

# What Is There to Worry About? An Introduction to the Computer Security Problem

Donald L. Brinkley and Roger R. Schell

---

This essay provides an overview of the vulnerabilities and threats to information security in computer systems. It begins with a historical presentation of past experiences with vulnerabilities in communication security along with present and future computer security experiences. The historical perspective demonstrates that misplaced confidence in the security of a system is worse than having no confidence at all in its security.

Next, the essay describes four broad areas of computer misuse: (1) theft of computational resources, (2) disruption of computational services, (3) unauthorized disclosure of information in a computer, and (4) unauthorized modification of information in a computer. Classes of techniques whereby computer misuse results in the unauthorized disclosure and modification of information are then described and examples are provided. These classes are (1) human error, (2) user abuse of authority, (3) direct probing, (4) probing with malicious software, (5) direct penetration, and (6) subversion of security mechanism. The roles of Trojan horses, viruses, worms, bombs, and other kinds of malicious software are described and examples provided.

In the past few decades, we have seen the implementation of myriads of computer systems of all sizes and their interconnection over computer networks. These systems handle and are required to protect credit data, justice information, computer vote tabulation, consumer billing, health data, insurance data, military and intelligence data, and computer and human communications, as well as countless other types of information. It is likely that readers of this essay have heard about some computer or network break-in at some time in the past few years.

Such events have been the subjects of popular movies and books, and reports about them are appearing ever more frequently in the press. Even telephone service has been disrupted by what has been described as “electronic vandalism.” Finding the source of such problems can amount to the electronic equivalent of looking for a needle in a haystack, so often the source of the problem is never discovered. Such is the nature of what we refer to in this essay as the computer security problem.

The first step in this introduction to the computer security problem will be a brief look at the historical parallels with a related subject — the communication security problem. The aspect of communication security addressed here is the protection of information while it is being communicated electronically from one place to another.

## **Historical lessons**

The computer security problem has grown with the computer industry. For roughly the first two decades of the use of electronic digital computers, the problem of security really was not noticed. The early computers were used to process sensitive information, both in environments involving national security and in commercial applications. But the size of the computers and the nature of their applications allowed any security problems to be solved outside the computer. If the entire system was dedicated to a single user, protection consisted of the user simply picking up his tapes and cards and clearing CPU memory when the job was finished. If one had sensitive information on a computer, one locked the computer in a room. Basically the user had complete control over his processing environment, including his data and programs. The computer itself was not really part of the security problem or its solution.

In the 1960s, users began demanding better utilization of computing resources, and the security environment surrounding computer systems started to change. The response to the demand for more efficiency gave birth to multiplexing techniques, resource-sharing operating systems, multiprogramming, and various other techniques of the age. One could build a time-sharing computer system to serve many users simultaneously. Users suddenly found not only a lack of control over the processing environment but a lack of control over their data and programs as well. While some of the early time-sharing systems were used in so-called benign environments where there was no exposure to security problems, others were not.

Users of time-sharing computers — for example, in academic environments — rapidly found that there was a real threat of unauthorized disclosure or modification of any sensitive information processed on the early time-sharing systems. There arose the problem of defending independent software structures from each other, as these were often im-

plemented on the same physical resources. Thus, multiprogramming operating systems began to enforce some sort of isolation of simultaneously executing processes. Since efficiency was the main consideration in computer systems, design criteria limited the “defending” and “isolation” primarily to the containment of accidents and errors.

Organizations desiring to utilize the increased capacities of resource-sharing systems demanded assurances that sensitive and nonsensitive information could be processed concurrently. Responding to customer pressure, computer systems manufacturers at first claimed that hardware and software mechanisms supporting resource sharing would also (with perhaps minor alterations) provide sufficient protection and isolation to permit multiprogramming of sensitive and nonsensitive programs and data.

This claim was soon discredited in the early 1970s with the introduction of several penetration tiger teams that were specifically tasked to test the protection offered by several major operating systems. Even those systems that underwent “retrofitting” to correct known implementation errors and design oversights were penetrated with only moderate amounts of energy. A Consensus Report published in the proceedings of the 1979 National Computer Conference [AFIP79] states:

It is a fact, demonstrable by any of several studies, that no existing commercially-produced computer system can be counted upon to protect any of its moderately knowledgeable users from having complete and undetectable access to any information in the system, no matter what kinds of so-called security features or mechanisms have been built into the system.

Despite the fact that little substantive reduction in the basic vulnerabilities occurred throughout the 1970s and 1980s, vast numbers of computer systems serving every imaginable user population and processing information of every possible degree of sensitivity have been put in operation. These computer systems are connected together into communities via various types of networks. The connection to networks introduces a need for communication security (often utilizing cryptography) to counter the possibility of an attacker tapping into the communication lines used by the networks. Further, by providing more ways of accessing the systems (from other interconnected systems, from remote terminals, and so on), the connection to networks gives potential attackers greater accessibility to the computer systems, thus compounding the computer security problems. However, the fundamental computer security problems are little affected by the interconnection of systems via networks. That is, the networks themselves provide few fundamentally new computer security problems, other than enhanced accessibility of the interconnected systems to potential attackers. Thus, the primary computer secu-

rity implication of connecting a computer system to a network is the increased exposure of the system to those individuals with the opportunity and the means to exploit the underlying computer security problems.

As noted in the preceding paragraph, however, communication security technology was developed to counter a different threat — that of an attacker tapping into communication lines. Communication security technology has a much longer history of development and use than that of computer security. Those concerned with computer security will be well served to learn from some of the errors made in using communication security. The remainder of this section briefly reviews some of the lessons from the history of communication security in an attempt to help those concerned with computer security avoid the mistakes encountered there.

Communication specialists, especially in the military, early recognized the vulnerability of electronic transmissions to interception (for example, through wiretaps or surreptitious listening to radio signals). The solutions were simple but drastic — restrict transmissions only to relatively unimportant (and nonsensitive) information or to transmission paths that were physically protected from intrusion. Obviously, these solutions limited use of the communications technology where it was most needed — in potentially hostile situations.

The communication security restrictions eventually gave rise to various cryptographic devices. These devices encode information into an unintelligible form so that protection of the entire transmission path is not required. But (of paramount importance) this dramatically changed the very nature of the security problem itself from a question of physical protection to a question of technical efficacy. Unfortunately, a problem arose with this new technology. When selecting a cryptographic device to use, rather than determining its effectiveness through careful technical analysis, selection was often based on the apparent absence of a known way to attack it.

Technically weak cryptographic devices found widespread military use because of misplaced confidence and the pressing operational need for electrical communications. One notable example was the Enigma machine used by the Germans during World War II. Their high-level national command and control network used it for communication security throughout the war. The Germans considered the Enigma completely safe. Yet before the war really got started, the British (perhaps aided by others) had in fact “solved the puzzle of Enigma” [WINT74].

The Enigma example also shows how the tendency to defend previous decisions (to accept and use techniques that were plausible but turn out to be insufficient) assures those so inclined of opportunities for exploitation. Ultra — what the broken Enigma signals were called — “not only gave the full strength and disposition of the enemy, it showed that the Allied (troops) could achieve tactical surprise.” In fact, General Dwight Eisenhower stated that “Ultra was decisive” [WINT74].

To be sure, the Germans “must have been puzzled by our knowledge of their U-boat positions, but luckily they did not accept the fact that we had broken Enigma” [WINT74]. There was a similar misplaced trust by the Japanese. The Japanese “hypnotized themselves into the delusion that their codes were never seriously compromised” [KAHN67]. Both the Japanese and the Germans, it seems, would not acknowledge their security weaknesses without direct confirming counterintelligence — and this came only after they had lost the war.

Technical experts eventually provided a sound technical basis for countering these sorts of communication security problems. Today our military and the commercial sectors make widespread use of cryptographic devices with confidence. For computers, as for communications, the nub of the problem is the effectiveness of the security mechanism and the assurance of its implementation.

Applying this history further to computer security, important lessons become clear. Misplaced confidence in the security of a system is worse than having no confidence at all in its security. Each (formal or de facto) decision to use computer systems without proper computer security controls permits a technical weakness to become a vulnerability in the same way the broken codes were a vulnerability to the Germans and the Japanese. An implication for computer security in military systems is that the lack of hard counterintelligence on exploitation should not be offered as evidence of effective security, even though the absence of war against an enemy capable of exploiting such weaknesses in our military computer systems has precluded ultimate exploitation.

The thrust of this historical review is captured in the maxim, “Those who cannot remember the past are condemned to repeat it.” The main lesson to be learned is this: Do not entrust security to technology unless that technology is demonstrably trustworthy, and the absence of demonstrated compromise is absolutely not a demonstration of security.

## **Computer misuse**

As we observed earlier, today there are vast numbers of computer systems and networks serving every imaginable user population and processing information of every possible degree of sensitivity. There is also a large and growing threat to the security of much of this information and the resources that handle it. The threat stems from the potential for computer misuse.

With respect to the potential damage done, computer misuse can be categorized into four broad areas:

1. theft of computational resources,
2. disruption of computational services,
3. unauthorized information disclosure, and

#### 4. unauthorized information modification.

Within these areas of computer misuse, we are concerned about misuse effected by either authorized users of the computer or those not authorized to use the computer. However, the former is generally a much more difficult problem to counter. As we shall discuss further, countering unauthorized information disclosure and modification depends as heavily on preventing the illicit access of bona fide users as preventing illicit access of nonusers. On the other hand, countering theft of computational resources and disruption of computational services primarily requires keeping nonusers from using (and abusing) the resource.

Note that computer misuse may arise in situations where there are legal penalties for the attackers if they are caught and convicted of these acts (for example, in countries or states that have specific laws against them). However, misuse may also occur in situations in which there is not a reasonable chance that the attackers will be captured and punished (for example, in espionage or warfare between two countries). In the latter situations, of course, there will be more of a concern with protecting against the threat being carried out; in the former, determining the damage and identifying the attacker after the fact are important as well.

**Theft of computational resources.** The theft of computational resources is one area of computer misuse. Much as some people connect into electric utility lines and divert power for their use without payment, computer resources can be diverted. For example, the systematic use of computational resources on a large scale by unauthorized users can have both economic and production impacts. Likewise, the use of systems by authorized users for unauthorized purposes (game playing, office football pools, bowling league administration, and so on) is a theft of the computational resource, as it represents a drain upon the computer system. However, such theft by authorized users is usually on a limited scale and is much harder to control, since, in general, it is difficult to determine whether authorized users are using the computer for authorized purposes only. Hence, preventing theft of computational resources typically focuses on protecting against actions by unauthorized users. Not only is prevention easier than apprehension, but also it is the unauthorized users who most likely perpetrate a theft on a major scale with little evidence to permit apprehending the thief.

Unauthorized programs that extensively use the hardware and the operating system can consume many thousands of dollars of computer services and compete with authorized programs. In addition to lost time, hardware subsystems — memory, for example — may be filled by the unauthorized programs to the exclusion of valid data, causing failure or delay of authorized programs.

If measures are taken to exclude unauthorized users, say through passwords, the persons responsible may misrepresent themselves as valid users, for example, by using a valid password they guessed or observed. In this way, they can continue to run their programs and charge any expenses to some legitimate user. However, stronger measures than passwords can be used to make it harder for an unauthorized user to masquerade as a legitimate user. Furthermore, other technical controls (for example, what is known as a “trusted path,” described in the next essay) can be used to substantially enhance the effectiveness of passwords and other measures.

**Disruption of computational services.** The second area of computer misuse is the disruption of computational services. Consider the situation of two companies competing for a contract. Company A has an automated cost/schedule system, while Company B does not. In the last several days before bidding closes, Company A’s computer (upon which its management relies) is unavailable due to a series of failures. Company A is forced to develop its bid without benefit of its system and thus proposes a contract with a cost 5 percent higher than the more accurate, computerized figures would have suggested. Company A loses in its bid on the basis of cost. The computer failures were caused by saboteurs from Company B.

Such disruptions of service can be extremely expensive if the timing is correct. Interruptions can be caused by attacks on any of the computer system’s components: hardware, software, or data. Physical damage to the hardware, confusion or modification of the operating system, subtle changes in the applications software, or modifications to the data or interface can all cause such problems. This form of attack does not necessarily use significant amounts of computer time, nor does it require either access to or modification of data. Nevertheless, access to the system by authorized users can be denied through these subtle attacks.

In such situations the misuse often goes undetected. The failures may be blamed on “bad luck” or on inadequate preventive maintenance. Since resources accessed may be minimal, few “tracks” may be left by the attackers, and proof of malicious acts or intent is often difficult to obtain. Identification of the perpetrators is even more difficult, since it is relatively easy to cover what “tracks” are left behind. Like preventing theft of computational resources, protecting against disruption of computational services typically focuses on protecting against actions by unauthorized users and on attempting to detect any misuse after the fact by authorized users by performing a “best effort” audit on their actions.

**Unauthorized information disclosure.** The third area of computer misuse is the unauthorized disclosure of data stored in the computer.

This type of act, much akin to our popular view of espionage, is well understood when we speak of unauthorized access to conventional data files stored on paper in filing cabinets. Computers add a new dimension to this problem. The enormous storage capability and the fast access times make the stakes more costly. A criminal might normally have to search an office for hours to find the data he wants (if he finds it at all). The long periods required for such nefarious searches greatly increase the chances of being “caught in the act.” Using a computer, the attacker can search databases equivalent to several offices’ filing systems in a matter of seconds. The fact that computer data systems often store data used for critical decision-making processes (such as the hypothetical situation of Company A above) usually implies that the data is entered in a more timely manner than in paper-based systems. The complete and up-to-date information in a computer system makes it a tempting target. “Outsiders” with full access to “inside” information pose a significant threat to business and society.

Furthermore, a computer used to implement a cryptosystem to protect information from unauthorized disclosure (for example, by encrypting the files it stores) presents a particularly attractive target, since cryptosystems typically depend on the nondisclosure of the encryption or decryption keys for the protection of the data being encrypted. That makes the keys themselves a small but extremely valuable target. When such keys are included in the data processed by the computer, such a small target is far easier to “slip out” of the system than all of the data that the keys can be used to compromise. This presents a significant motivator to those bent on attempting to gain unauthorized access to such data to try to implant some malicious software in the system (for example, through a penetration of the system). The malicious software could covertly leak the keys once the system is operational. Perhaps even more so than other forms of computer misuse, unauthorized disclosures are difficult to detect, since there may be no apparent tracks left behind.

**Unauthorized information modification.** We have mentioned the advantage of knowing the facts upon which a competitor is basing his decisions, and of knowing the decisions as soon as they are made. Of far greater use is the ability to “feed” data (erroneous, misleading, or incomplete) into the competing system and thus control or influence the decisions to one’s benefit. With such ability, one can virtually “change the facts” to suit the situation. Consider also the ability to “feed” keys into a cryptosystem such as that described in the preceding section (for example, where the keys are stored or manipulated by the computer that may be penetrated). If he had the ability to control the keys in a cryptosystem, an attacker could virtually control the cryptosystem. With this power to substitute his keys for the intended keys or to control the generation of the keys used, the attacker could not only read and write

the data flowing through the cryptosystem as would be possible even if he only knew the key (see the preceding section), but he could also generally introduce new data and read and write any data of the sort protected by any of the keys in the system.

The classical cases of computer operators modifying their credit ratings or erasing the records of their outstanding debts are small examples of this area of misuse. Also included in this area are modifications made to programs. Modification of accounting routines to prevent charges to one's credit card from being billed to one's account is an example. In view of our ever-increasing dependence on computerized systems in banking and commerce, the potential for large-scale fraud is enormous.

### **Computer misuse techniques**

The latter two areas of computer misuse discussed in the previous sections are threats to the information that computers handle rather than to the computers themselves. Threats to information (unauthorized information disclosure and modification) are fundamentally different from threats to resources (theft of computational resources and disruption of computational services). For one thing, countering information-oriented computer misuse depends as heavily on preventing the illicit access of bona fide users as preventing illicit access of nonusers, whereas countering resource-oriented computer misuse primarily requires keeping nonusers from using (and abusing) the resource. Further, resource-oriented computer misuse is countered using control mechanisms different from those necessary for countering information-oriented misuse. Generally, resource-oriented misuse is most effectively countered with measures such as maintaining separate rooms for computers, registering users of computers and networks, requiring use of passwords, and other physical and administrative security measures. Appropriate additional measures may include those that fall into the areas of computer system reliability and human engineering.

For the most part, this book in general and this essay in particular deal with information-oriented computer misuse. In thinking about how information-oriented computer misuse may be brought about, we have come to characterize "what there is to worry about" into six different classes. These classes are

1. human error,
2. user abuse of authority,
3. direct probing,
4. probing with malicious software,
5. direct penetration, and
6. subversion of security mechanism.

Common to all six of these classes are two elements:

1. A **vulnerability**: A quality or characteristic of the computer system (for example, a “flaw”) that provides the opportunity or means of exploitation.
2. A **threat**: The possible existence of one who participates in the exploitation by gaining unauthorized disclosure or modification of information such as accompanies information-oriented computer misuse.

The product of threat and vulnerability is considered the risk one faces in operating a system. Each of these two elements may in theory be brought about either accidentally or deliberately. For example, a vulnerability may be introduced through either the accidental introduction of a “bug” into the system or the deliberate introduction of malicious software. However, although a threat of accidental exploitation of a vulnerability may exist, we are concerned in this essay only with deliberate exploitations of vulnerabilities (that is, intentional acts attempting unauthorized disclosure or modification of information), since these are the purview of computer security, as described in Essay 2.

Therefore, the remainder of this essay discusses the six classes of information-oriented computer misuse listed above, which we call “computer misuse techniques.” We have used the rubric “computer misuse techniques” for all six classes, since what we are talking about with each is the threat of deliberate exploitation of an accidentally or deliberately introduced vulnerability. This distinction is clear in considering the first class — human error. The “error” allows the accidental introduction of a vulnerability, which may then be deliberately exploited, making it a “technique” for computer misuse.

We will now introduce the six classes of computer misuse techniques and provide examples for each class. Note that these classes of computer misuse are not mutually exclusive, as attackers may combine techniques to more effectively achieve their nefarious goals.

**Human error.** Human errors that lead to unauthorized disclosure or modification are basically probabilistic in nature. They may involve several human, hardware, and timing factors that when combined could allow an unauthorized disclosure or modification of information. Simple examples of this method are a computer operator inadvertently mounting the wrong tape, a user typing sensitive information into a file that is thought to be nonsensitive, or (assuming the computer has some sort of access controls for files) a user setting the authorized mode of access for a file to an inappropriate value. Users receiving information from this kind of disclosure or modifying information in this manner are often victims of circumstances and may not be malicious in their intent. How-

ever, even though the success of this method relies on probabilistic events that one cannot control, the method can be used by a determined attacker.

The basic approach used by an attacker in this method is to sit and wait for the proper set of circumstances to occur. Upon detection of a breach in the protection of information, the attacker acts to exploit the breach and make use of this accidentally introduced vulnerability.

This method could prove profitable to a malicious user, particularly if the system under attack has a history of human errors. Although this method is viable, other methods will be discussed that do not rely on these probabilistic circumstances and would thus be far more attractive to an attacker. Better human engineering reduces the probability of human error.

**User abuse of authority.** Al Capone's bookkeeper is reported to have said, "I can steal more with a pencil than ten men with machine guns." That was probably a conservative estimate. Regardless of the bookkeeper's actual capabilities, his chances of escaping detection and apprehension by law enforcement authorities would have been significantly better than those of a gang of gunslinging outlaws. If we can replace a hundred or a thousand crooked bookkeepers with a single vulnerable computer, consider the possibilities. The computer significantly increases the speed and possible scope of criminal acts.

Newspaper readers will occasionally encounter reports of "computer misuse" or "computer crime" at some financial institution that would probably be just as happy to avoid the publicity. The cases reported range from simple to fiendishly complex. In a simple case, perhaps a bank teller records, by a few keystrokes on a terminal, the receipt of a few thousand or hundred thousand dollars of money that "isn't there." An accomplice withdraws most or all of the money from another branch of the bank and both flee to some warmer and more hospitable clime. The daily audit of account balances and the real amount of cash on hand soon reveal the shortage, but by then the culprits are likely to be long gone.

More complex instances of this sort of computer misuse tax the imagination of the financial institution and the credulity of the reader. In one widely reported case, a bank officer is alleged to have submitted transactions with values in the millions over a period of more than a year. In this case, the strategy was allegedly to exploit the delay of funds or paperwork traveling from bank to bank or branch to branch, build up a large "float" of money in transit, and siphon off some of that float to the goals of personal enrichment and early retirement. The details of such a scheme are apparently intricate and difficult for the culprit as well as the reader. If the culprit slips up or there is some sort of accident, the float may sink, as it were, and the entire scheme could be discovered. In

one case, the fatal flaw was apparently a computer outage that required a batch of bogus transactions to be processed by hand.

Computer crimes such as these make interesting reading, and it seems likely that they will become more common in the years to come, as computers become more common in every nook and cranny of life. There is already a vast literature on such incidents, and one might be tempted to conclude that these cases typify the computer security problem. In fact, however, a review of the cases shows that they have little to do with computers. Whether they are simple cases involving the entry of a fictitious transaction or complicated ones involving the management of vast “floats,” these crimes could as well be carried out in a world of quill pens and green eyeshades as one that used the latest computer systems. The key common factor is the irresponsible action of an authorized individual who, in some manner, abuses a trust and abuses the authority granted to perform some task.

These essays have little to say about the problem of controlling authorized users of an application beyond confining the authorized users to their domains of authority and providing the capability to automatically audit the users’ security-relevant actions. Computers cannot reach into users’ minds and determine whether they are abusing the trust placed in them. We believe, in fact, that restricting the users to their domains of authority is a worthwhile goal for any system that processes valuable data, but the legitimacy of the authorized users’ actions is better measured by auditors or experts in the area of action than controlled by computer systems.

**Direct probing.** If cases of user abuse of authority are as old as banks and ledger papers, the computer misuse technique that we have called “direct probing” is as old as shared computer systems. We use the term probing to distinguish those cases where an individual uses a computer system in ways that are certainly allowed, but not necessarily intended, by the system’s operators or developers. It is important to realize that the individual who is attempting probing is deliberate in his attempts. This introduces a class of “user” that computer system designers may not have seriously considered. Often, designs reflect that the systems are expected to operate in a “benign environment” where violations of the system controls are presumed to be accidental. Because systems are presumed to be in a benign environment, the attacker may not have to exert much effort to succeed.

We present two examples because these cases are important. The second example bears many similarities to the story told by C. Stoll in *The Cuckoo’s Egg* [STOL89], but it does describe a different event.

*Students and sloppy security.* The growth of computer literacy among high school (and younger) students has resulted in a large population

interested in seeing “what systems are out there” and what information and programs are available. More frequently, there seems to be significant interest in gaining unauthorized access to other people’s computers and information.

A widely reported incident involved students from a school in New York who learned a telephone number that would give them access to a time-sharing computer operated by a small business in another city. The students dialed the number and recognized the “banner” that the system displayed on their terminal as belonging to the same operating system used by their school’s computer. They knew from experience with their school computer that copies of the operating system were distributed by the manufacturer with user accounts predefined for the system manager, who was expected to log in and create new accounts for his facility’s users. The system manager account had privileges to access any information on the computer. It was always distributed with the same “secret password,” and the documentation for the system directed the system manager to change the password as soon as he had installed the system.

The students tried the login sequence for the system manager’s account and gave the “distributed” password. Although the small business system had been in place for a long time, the system manager’s password had never been changed, and the students found themselves logged in with a fully privileged account. They read files, deleted files, and played games. They were able to create new accounts for themselves (they were, after all, the system manager for the computer) and made fairly general nuisances of themselves.

Eventually they asked for a “ransom” to stop playing on the system and leave its owners and users alone, and the ransom demand revealed enough information to allow the authorities to intervene. The students wound up the exercise with gently slapped hands, and the victims were not (very) much worse off for the experience.

The key aspect of this case is that the students did not use the computer or its software in any way that was unexpected by its developers or owners. They merely took advantage of sloppy administration by the system’s operators and inserted themselves where they had no business being.

*Network adventure.* We expect that many of our readers will have played or heard of some form of the computer game Adventure. In this game, one is presented with the computer’s representation of a maze of interconnected chambers. Some contain treasures that the player can collect and bring back to a “home base” to gain points in the game. Other chambers contain various kinds of monsters, miscreants, and unfortunate circumstances. The topology of the maze is left to the player to infer. The object, of course, is to navigate the maze and collect the

treasures without being destroyed by one of the unfortunate circumstances.

Some persons unknown played a form of real-life Adventure in the large computer network operated by a major manufacturer. They dialed an “800” number that gave access to an account that was supposed to be used for reporting software problems. The account was improperly set up, and the “players” found that they could issue user commands to the operating system in question. In particular, they could try remote terminal logins to other systems in the manufacturer’s network and could copy files from other machines.

In exploring the network, the adventurers discovered that it was easy to find the names of users on other machines by using the network file access facilities to list directory names, and by assuming that user names were the same as directory names. They tried guessing passwords for the user names they had found, and discovered that occasionally user name and password were the same. Then they could log in on a remote machine as one of its users.

As they wandered through the network, the adventurers discovered more user names and more passwords. Many files were unprotected (readable to any user on the network), and some gave clues to the names and passwords of yet more users. In addition, by logging into remote systems, the players were able to read files that were protected from “world” or general access — they now had legitimate local accounts that might belong to a group that had access to the files, or might own some files. Some of the files they found in this manner even contained stored passwords. In addition, some users had accounts on several machines. When those accounts all had the same password, it was a bonus to the now experienced adventurers.

Eventually the adventurers got access to some “privileged” accounts on a few machines. These accounts had the authority to bypass the normal system controls and read or write any information on the systems. In particular, the players could read lists of user names and passwords, and now had unrestricted access to many more “chambers” in the “maze.” The adventurers were able to compromise numerous accounts and to read an “awesome” number of files during their explorations.

The only misfortune that the adventurers suffered was early discovery. They were logged into the machine that was their starting point using a user name and password that should not have been associated with that communication line at that time of day, and an alert user realized that fact. The user developed a program that could monitor all transactions from the “800” number, and the manufacturer was able to record all the adventurers’ activities. Because they were apparently students on a low budget, the adventurers were unwilling to stop using the system through the all-important (and free) “800” number. Eventually,

through a combination of telephone traces, rumors, and good luck, the management of the network got an idea of the identities of the players and the game was canceled.

This case is like the previous case in one key respect. The adventurers, like the student vandals, were using the computer roughly as it was intended. They were looking at files that were there for the reading, logging in through the normal paths with passwords that were legitimately installed in the systems, and issuing legitimate commands to operating systems and network software. Their ability to gain access to a huge quantity of information resulted from a combination of sloppiness on the part of system users and managers and the fact that most of the systems they were exploring had fairly “coarse” protection, so that files were frequently left accessible to the “world.”

The results of this fairly unsophisticated game of Adventure are nonetheless frightening. Had the players had a sufficient budget to dial directly to a machine where they had gained access to an account (instead of reusing the “800” number and then exploring by use of the intercomputer network’s communication), they need never have been discovered or stopped. They had access to a large volume of sensitive information, but were apparently more interested in seeing how far they could go than in making malicious use of the information they could find.

**Probing with malicious software.** This computer misuse technique is quite similar to the previous technique, in that it is a form of probing—it involves using a computer system in ways that are allowed, but not necessarily intended. With this technique, however, specially developed software is used by the attacker for the express purpose of carrying out the probing. The use of such malicious software in probing makes this technique unique and significant.

*The Trojan horse.* Ancient Greek mythology supplies us with the story of the Trojan horse, which when brought within the fortified walls of Troy, opened to reveal hostile Greek soldiers, who attacked the city’s defenses from within. The term Trojan horse for software is widely attributed to Daniel Edwards [SALT75], an early computer security pioneer, and it has become standard in computer security. Like its mythological counterpart, it signifies a technique for attacking a system from within, rather than staging a frontal assault on well-maintained barriers. However, it does so without circumventing normal system controls (in the same manner in which the Trojans opened the doors of the city to bring in the horse). A Trojan horse is a program whose execution results in undesired side effects, generally unanticipated by the user. A Trojan horse will most often appear to provide some desired or “normal” function. In other words, a Trojan horse will generally have both an overt

function (to serve as a lure to attract the program into use by an unsuspecting user) and a covert function (to perform clandestine activities).

The overt or “lure” function of a Trojan horse can, for example, be mathematical library routines, word processing programs, computer games, compilers, or any program that might be widely used at an installation. Because these programs are executing on behalf of the user, they assume all access privileges that the user has. This gives the covert function access to any information that is available to the user.

The covert function is exercised concurrently with the lure function. An example of this kind of malicious software might be a text editor program that legitimately performs editing functions for the unsuspecting user while browsing through his directories looking for interesting files to copy. Attackers have used seemingly harmless computer games (for example, backgammon) to set all of the player’s files to “world read” so the game’s author can copy the files without the knowledge of the files’ owner.

This is a particularly effective option for the attacker due to the fact that as far as any internal protection mechanism of the computer system is concerned, there are no “illegal” actions in progress. The Trojan horse (for example, text editor) is simply a user program, executing in user address space, accessing user files, performing perfectly legitimate system service requests such as giving another user (for example, the attacker) copies of files.

An example comes from the security test of an Air Force system that was used to process sensitive information in the early 1970s. The installation in question was processing classified magnetic tapes using a computer and operating system that were widely known for the ease with which a hostile individual could access any information processed. The installation’s solution was to use only a selected set of programs to process the classified tapes, while any user was allowed to submit any unclassified program that he or she wished. The programs used to process the classified tapes not only did the requisite processing, but also took special precautions to label the classified information that appeared on the line printers. They even erased the main memory areas that had been used to store the sensitive data before terminating processing and returning the memory areas to the operating system for reallocation.

A security test team (a tiger team) realized that the classified processing programs could be used to ease the attacker’s job. By exploiting the operating system’s weaknesses to access it, they modified the program used to print the contents of a classified magnetic tape to serve as a Trojan horse. The Trojan horse program completed the print job when requested, but also hid a copy of the classified data, lightly encrypted, in an “invisible” location on disk. A later unclassified job could be submitted to read the hidden data, print it out (still encrypted) for a member of

the tiger team, and erase the hidden copy. In this case, a security solution actually made a security problem worse, since the use of the classified processing programs served to locate and save for the tiger team exactly those files and jobs that they wished to steal.

To reinforce the subtle nature of Trojan horses and the reality of the opportunities to plant them, the interested reader should read the story by Ken Thompson [THOM84], one of the codevelopers of Unix, of what he describes as “the cutest program [he] ever wrote.” This program is a Trojan horse that he introduced into the C compiler. When the Trojan horse in the C compiler determined that it was compiling the “login” code for the Unix operating system, it would generate code to accept not only the valid password but also a fixed password that he had previously selected and built into the Trojan horse. He also planted another Trojan horse in the compiler. This Trojan horse added the code for the two Trojan horses to the object code each time it recompiled subsequent versions of the compiler, without the Trojan horse code having to be present in the compiler source code. In this way, he was able to cover his tracks for years, while his employer possibly continued to unknowingly crank out copies of the operating system and compiler containing his Trojan horses.

While our stable full of Trojan horses may seem quite different from the network Adventure or password exploitation cited above, there is a technical point that all three cases hold in common. That is, just as one can guess a password or read an unprotected file without doing violence to the mechanism of the underlying computer and operating system, one can install a Trojan horse in a program that will be used by an intended victim, and that Trojan horse can function within the normal rules and mechanisms of the computer and its operating system. By issuing operating system directives to reset the access controls on a file or make a new copy, the Trojan horse can take advantage of standard mechanisms to do its dirty work without detection.

Two things make Trojan horses particularly attractive to the hostile attacker. First, as a practical matter, there is no effective procedure for detecting whether a piece of software contains a Trojan horse, especially if the designer devoted a reasonable effort to hide it. Second, almost all computer users are compelled to use software (for example, operating systems and application programs) developed by persons completely unknown to them. The route this software takes to the user provides numerous opportunities for insertion of a Trojan horse.

*Viruses, time bombs, logic bombs, and worms.* These four eye-catching names have received some notoriety in the popular press in the past few years. Actually, these are four types of Trojan horses, with special characteristics.

A *virus* is a self-replicating Trojan horse that attaches itself to other programs in order to be executed. This method of self-replication by at-

taching to another program to be executed is the primary distinguishing feature of a virus. The primary covert function of some viruses may be simply to replicate and spread, performing no other harmful action. However, others may take such actions as to modify, copy, or destroy other files or entire disks. Viruses are carried from one personal computer to another by unsuspecting users sharing software that has already been infected. They spread among other computer systems in a similar manner via networks. Note that a virus's method of replication by attaching itself to another program is, in itself, an unauthorized modification of data (the program to which it is attached).

Since each copy of a virus may replicate itself, a virus's ability to spread quickly is one of its most distinctive qualities. For example, suppose one copy of a virus is introduced into a system through the execution of an infected text editor. If that virus is able to attach itself to a new program just once per day, and each of its copies does likewise, after a week there will be more than 50 copies. After about a month, there will be around a billion copies. Although this is not our serious prediction, it is interesting to note that at the rate currently identified viruses seem to be spreading, by 1995 every computer in existence could potentially have a virus. (Do not be overly frightened, however, as this book will describe the technology for arresting Trojan horses; it is only necessary to employ it.)

A *time bomb* is simply a Trojan horse set to trigger at a particular time. For example, time bombs have been set to trigger on Friday the 13th and on the anniversary of events that were significant to the attacker.

A *logic bomb* is a Trojan horse set to trigger upon the occurrence of a particular logical event. For example, the Trojan horse might be set to trigger at the worst possible time. An event such as the need to correct a temperature imbalance in a power plant might trigger a logic bomb that modifies data to make the imbalance worse. Another example of a logic bomb is a "letter bomb" — contained in electronic mail and triggered when the mail is read.

Another form of Trojan horse is known as a *worm*. A worm is a program that distributes multiple copies of itself within a system or across a distributed system either through the exercise of a flaw that permits it to spread or through normally permitted actions (for example, mailing copies of itself to other systems, compiling them on the remote systems, and initiating their execution). Once in place, the worm may attack in any number of ways, through the methods described above or in the paragraphs below. A good example of a worm in action is the "Internet worm" [SPAF89]; it invaded a large, nationwide network of computers called the Internet, spreading to thousands of machines and disrupting normal activities and connectivity of the machines for several days. Fortunately, the primary objective of this worm was simply to spread to more machines, rather than to do any particular damage once it was

established on a new machine. Otherwise, the damage would have been far more significant.

**Direct penetration.** Probing relies on the fact that a computer's security controls are being used sloppily, or that the controls are so poorly designed that a user cannot control the sharing of information in a way that corresponds to his or her needs. Penetration, in contrast, involves the bypassing of intended security controls. In many cases, an attacker finds a single flaw in the implementation of an operating system or hardware, writes a program, and has the entire computer at his or her command.

Penetration typically involves the use of malicious software such as a Trojan horse, to confirm and exploit the flaw. It is not necessarily difficult or costly. Three historical examples will illustrate the point.

*The best is not good enough.* Honeywell Information Systems and MIT conducted a cooperative research project during the late 1960s and early 1970s to develop the Multiplexed Information and Computing Service (Multics). This time-sharing system incorporated hardware and operating system software "designed with security in mind" and was widely touted as the most secure operating system of its time. An Air Force tiger team was evaluating Multics [KARG74] as a candidate for use in a Pentagon application where the computer itself would be required to protect sensitive information from authorized system users who were not "cleared" to see all the information in the computer. Only the Multics operating system would stand between classified information and users without the clearances to see it.

In examining the Multics operating system programs, the tiger team found one place where an ordinary user program could branch to any location in a supervisory (executive mode) program. Such instances were forbidden by the Multics design concepts, but the implementation had made this program directly accessible for reasons of efficiency. The tiger team examined the program listings and found two instructions that would store a word at any location they specified, then return to their program. The privileges of the supervisory program were such that the store instruction could operate even on the programs or data used by the operating system itself.

The tiger team also found a place where Multics first checked a user's authorization for access to a file and, when the request proved valid, executed the request. However, in this case, the user could change the request after the validity check and before the execution of the request. Again, the vulnerability could be exploited to operate on any information in the computer.

The tiger team tested the first apparent vulnerability on an Air Force laboratory computer, then tried it on the Multics development computer

at MIT. They were able to use a small “hole” in the operating system to change the access privileges for a penetration job and gain complete and unrestricted access to any information on the computer. Specifically, they were able to change the “user ID” for their process from that of an ordinary user to that of the system manager who maintained the system programs and files. Given this level of access, the tiger team had effective ownership of the MIT Multics computer.

Once they had a way to penetrate Multics, the tiger team provided a number of additional demonstrations of the thoroughness of their work. (These will be described in a later section.)

*When the hardware is soft.* A major concern in computer security during the late 1960s and early 1970s dealt with security-related hardware flaws. There was a fear in that era that processor hardware might fail in such a way that the processor would keep running but security-related hardware checks would no longer be made. For example, the failed hardware might allow a privileged instruction to be executed from a user program. The people who hypothesized the problem also invented a form of solution: An unprivileged, interactive program they called “Subverter” would check periodically to see if any security-related hardware failures had occurred and, if they had, sound a suitable alarm.

The tiger team that penetrated Multics developed the Subverter program to check for flaws in the Multics processor. This program would awaken once every minute or so and try a few illegal operations, then go back to sleep. The illegal operations ranged from commonplace to obscure and were chosen to invoke the complexity of the Multics processor hardware. These tests included

1. trying to run privileged instructions,
2. attempting to violate read and write permission on segments,
3. testing of all instructions marked illegal, and
4. taking out-of-bounds faults on zero length segments.

During several hundred hours of execution while the tiger team’s members were using Multics, Subverter never detected a security-related hardware failure. When the hardware broke, the system went down, and there was no opportunity for subtle security exploitation. The tiger team did, however, discover that Subverter would occasionally crash without apparent cause.

On investigation, it became clear that Subverter was crashing because its read-only program segment was being modified. The test case that Subverter used for illegal memory access was to try to write in itself, and in these cases it was succeeding. The cause was traced to a test that used a combination of register, indexed, and indirect addressing that spanned several of the segments that make up a Multics process’s vir-

tual memory. When the locations involved met certain requirements (one indirect address word had to be in location three of a specific segment in the address chain), the instruction that started the operation would succeed without regard to the process's access rights for the final target address. When the target address was in the read-only Subverter code segment, Subverter "clobbered itself."

The flaw that allowed Subverter to write in itself was not random. Every time the combination of addressing modes involved was used, the problem would occur. It could in fact be used to write anywhere in virtual memory on any Multics processor. Thus the hardware flaw was as exploitable from a security penetration standpoint as any of the flaws in the operating system. It was a speculation that there might be such a flaw that caused the tiger team to write Subverter so that it emphasized the testing of obscure and complex instructions and addressing modes. The flaw was found to have been introduced by a field change to the processor that had the side effect of removing a special-case security check.

The implications of the Multics hardware vulnerability are as frightening as those of the software flaws. The designer of a secure operating system usually assumes a known hardware base. In this case, the hardware was "almost" what was expected, but the difference was capable of rendering the system's controls ineffective.

*Job security instead of computer security.* An early instance of penetration illustrates the tremendous difficulties of penetration and security repair. Penetrate and patch, as this was called, was once thought to be an effective security method, but it is now known to be unreliable. In theory one could test all possible programs to find any that led to a security penetration. This method of exhaustion would be effective if it were possible, but it is far beyond the realm of feasibility. For any real computer, it would take so long that before the penetration and patch work was finished, the sun would literally have burned out! Essays 2 and 6 describe more productive schemes for designing and demonstrating the security of computer systems. See Essay 11 for additional information on penetration testing.

A large aerospace contractor operated a major computer center that was used to support both government contracts and a commercial service bureau. The government contracts required that the contractor process classified information, while the service bureau had a large population of customers who could access the computer center by telephone.

The contractor proposed to allow the service bureau users access to the computer that was processing classified information. It was well understood (by the contractor and government security officials) that the operating system used by the computer center could be penetrated by a

hostile programmer, so the contractor proposed to fix the system's vulnerabilities. To that end, the contractor assigned a team of system programmers to review the operating system code, find the ways in which the controls could be subverted, and then repair the vulnerabilities. The team labored for several months and produced a system that they pronounced secure.

At this point the government brought in an independent tiger team to assess the contractor's work. The team found, in about two weeks, a set of ways to gain control of the computer in supervisor state and obtain access to any file stored or information processed in the system.

One would think that such an incident might be the end, but the contractor responded by mounting a new team that was directed to find all the remaining exposures and fix them as well. After a few more months, the tiger team was brought back with the same quick and successful results. The cycle may have been repeated once more — the history becomes vague around this point. However, the point that is not vague is that the contractor eventually gave up, bought a separate computer, and secured it by locking it up. The tiger team left the field victorious after the final "game."

Note, however, that this was a relatively happy ending only because the tiger team was fortunate enough to repeatedly find flaws with little effort. If they had not, it may well have been the case that the system would have been declared "secure," and the flaws would have been found and exploited only by malicious attackers, while the contractor and the government proudly used their "secure" computer.

**Subversion of security mechanism.** Subversion of a computer system's security mechanism involves the covert and methodical undermining of internal system controls to allow unauthorized and undetected access to information within the computer system. Such subversion is not limited to on-site operations, as in the case of deliberate penetration. It includes activities that spread over the entire life cycle of a computer system, including design, implementation, distribution, installation, and use.

The legitimate activities that are carried on during the various life-cycle phases offer ample opportunities for the subverter to undermine system components. The activities in the first four life-cycle phases identified above are basically not sensitive in nature and are carried out at relatively open facilities. Therefore, the subverter would have little difficulty in subverting the system components under development. Later in the use phase, these same components would be involved in the protection of information. By this phase the subverter would have an "environment" purposefully constructed for the unauthorized and undetected exploitation of a system and the information it contains.

The subverter is not an amateur. To be able to carry out subversive operations, the subverter must understand the activities that are performed during the various phases of a computer system's life cycle. But none of these activities is beyond the skill range of the average undergraduate computer science major. In fact, much of the activity involved with subversion can be carried out by individuals with much less technical knowledge. The subverter can utilize a diverse group of individuals who may or may not be aware of the subversive activities they are performing. One needs only to imagine the vast number of people who will have access to the various computer system components prior to their being installed at a site with sensitive information.

The subverter could, and undoubtedly would, use various methods to circumvent the control features of a computer system. But the subverter is concerned with the long-term return on his subversive efforts. To rely on a design oversight or an implementation flaw that might be eventually corrected would not be sound "business" practice. Rather, the subverter constructs his own clandestine mechanisms that are inserted into the controlling hardware or software during one of the various phases of a computer system's life cycle. Such clandestine mechanisms have historically been called *artifices* [LACK74]. These *artifices* can be implemented as either malicious hardware or malicious software. The most common forms of *artifices* used in subversion are known as *trap doors* [KARG74].

A key characteristic of a *trap door* is that, since it is installed in the controlling portion of the system (for example, operating system) and is therefore capable, it circumvents the system's normal control features. Another key characteristic is that a *trap door* is exercised under the direct control of an activation stimulus.

As the name implies, *trap doors* have a means of activation (like the latch on a door). This activation key is under the direct control of the attacker. A simple example of an activation key is a special sequence of characters typed into a terminal. A software *trap door* program embedded in the operating system code can recognize this key and allow the user of the terminal special privileges. This is done by the software circumventing the normal control features of the system. It is important to realize that the only purpose of a *trap door* is to "bypass" internal controls. It is up to the attacker to determine how this circumvention of controls can be utilized for his benefit.

*Undetectable trap door.* The attacker can construct the *trap door* in such a manner as to make it virtually undetectable to even suspecting investigators. The penetration of the MIT Multics computer by the tiger team that was described earlier led to further demonstrations of the significance of their work. Specifically, they installed a small *trap door* so undetectable that the manufacturer's personnel could not find the

clandestine code, even when they were told it existed and how it worked.

The Multics system internally encrypted its password list so that even if the list was printed out, the passwords were not intelligible. When a user presented his or her password, it was encrypted and then compared with the user's entry in the encrypted list. The tiger team retrieved the encrypted password list, then broke the cipher at their leisure to obtain all of the passwords for MIT's Multics computer system. The MIT Multics computer was used as the development site for future versions of the Multics operating system.

The tiger team modified Honeywell's master copy of the Multics operating system by installing a trap door: a set of instructions to bypass the normal security checks and thus ensure penetration even after the initial flaw was fixed. The trap door was small (fewer than 10 instructions out of about 100,000) and required a coded password for use. As we said, the manufacturer's personnel could not find it, even when they knew it existed and how it worked. Furthermore, since the trap door was inserted in the master copy of the operating system, the manufacturer automatically distributed the trap door to all Multics installations. Multics kept an "audit trail" of accesses to files by users. The tiger team's activities were duly audited. However, the audit trail mechanism itself was subject to "repair" by an authorized system manager. Since the tiger team appeared to be the system manager, they merely had to modify the record to remove all traces of their actions, such as the insertion of the trap door.

The full effect of the tiger team's project was eventually demonstrated to Honeywell and Air Force management, and a series of projects was initiated to improve the system's security to allow it to be used in the Pentagon application. Of perhaps as much interest, though, are the depth and breadth of the impact that the tiger team's penetration and subversion had on the system's security.

The key point that distinguishes this subversion from the other forms of attacks described above is that the tiger team examined the mechanisms used to provide operating system security, then installed permanent artifices to bypass them all. Once the team had done that, they were able to access any information in the system repeatedly and undetectably, despite later efforts that might close the initial vulnerability they exploited.

*A hardware trap door.* The implications of the Multics hardware vulnerability described earlier are as frightening as those of the software flaws. The prospect for maliciously installed trap doors is presumably as great in hardware as in operating system software. It is arguable that, given the complexity of modern integrated circuits, such trap doors are even harder to find than their software brethren. While the tiger team's Subverter program described earlier was designed to find such an obscure case, all

involved acknowledged that there was a certain amount of luck in the fact that the case of interest was one that Subverter tested in a finite amount of time.

**Putting it all together.** It is easy enough to imagine scenarios for attacks against the security of information in computer systems in which the use of techniques such as those described in the preceding paragraphs gives an attacker a decisive advantage. The number of avenues available for inducing security compromises in a typical computer system and the range of activities and capabilities that can be exercised once a flaw is located and exploited are frightening to consider.

*Future attack scenario?* To illustrate the range of subtlety and the indirect nature of such threats, we offer the following fictional scenario [SHOC88], which is at least technically realizable. The purpose of this scenario, it is emphasized, is to prompt a more careful consideration of the threat. For that reason, we deliberately present a “worst-case” scenario.

In 1995, the NATO nations undertook several technical initiatives in the area of command and control. Among these initiatives, a project was undertaken to interconnect many preexisting command and control systems located at various sites with high-speed, dedicated communications links. The preexisting systems were all thought to be “secure enough” and were accredited to process information of identical classification ranges. The systems nominally provided for the proper confinement of classified information. For that reason, the sponsors of the project assessed as minimal the additional risk to security induced by the new connectivity.

The successful completion of this project proved, in the ensuing conventional hostilities in the Middle East, to be one of the decisive factors leading to the losses suffered by the NATO alliance. Unknown to NATO engineers, technical saboteurs under the control of the enemy had managed to penetrate the system by installing a Trojan horse at one of the sites as early as 1989 to exploit a flaw in the command and control system. This obscure flaw provided a means to modify arbitrary system instructions, by sending a particular I/O device an undocumented sequence of control instructions, causing the device to modify an arbitrary memory location, bypassing the usual memory management access controls. The Trojan horse was introduced in a virus-infected graphics package distributed as “freeware” on a publicly accessible bulletin board system. An unsuspecting staff system programmer, who was a frequent contributor to the bulletin board, noticed the package, downloaded it into his personal home system, and transported it to the command and control system for evaluation, as it offered several features that were currently required for an application being developed at the site. Al-

though the graphics package itself, after evaluation was completed, was rejected and removed from the system, the virus by this time had relocated itself and become a permanent resident of the command and control system.

The Trojan horse had been specifically designed to penetrate any command and control system site it should chance to find itself on. Its placement on the public bulletin board was a carefully targeted attack on the command and control system programming staff, who were known to be frequent contributors and subscribers. Because they placed the Trojan horse on a public bulletin board, there was little risk to the saboteurs of personal exposure even if the Trojan horse was eventually discovered. The Trojan horse had been carefully engineered in advance by the technical sabotage team, based on an exact understanding of the particular flaw exploited. This understanding was gained by lengthy experimentation on an identical computer system purchased off the shelf by the sabotage team in support of the penetration. Among the functions built into the Trojan horse was the ability to accept covert software “upgrades” to its own program once it had installed itself. In effect, the Trojan horse, once installed, gave its creators complete access to the system, provided a means of communication with the Trojan horse could be established.

When first executed at the command and control system site, the virus contained in the Trojan horse activated itself, confirmed that it indeed was now executing at one of the targeted sites, and relocated itself into the message processing subsystem, where it could monitor (without perceptible impact on ongoing operations) all incoming message traffic. By placing redundant “watchdog” processes in installed intelligent peripherals, the Trojan horse was able to ensure its continued existence past system maintenance and regeneration episodes.

The Trojan horse was designed to communicate with its creators via unclassified message traffic, such as that originated at (or addressed to) a remote diplomatic post to which the enemy had low-risk human access, at the unclassified level. In particular, the human agent had the ability to originate and receive routine unclassified messages of an administrative nature. The entire NATO communication system was available to ensure that such messages, once originated by either the human agent or the software Trojan horse, would be delivered unchanged to the recipient in the normal course of operations. The human operator could signal and control the Trojan horse by including a preselected string of code words in any unclassified message emanating from the post, followed by a “program” for the Trojan horse to execute, encoded as numeric table data. The enabling trigger and program code were carefully designed to mimic a routine report originating from this and similar posts, while ensuring that the risk of the “trigger” actually occurring in a genuine message was low. Signals from the Trojan horse to

the operator could be sent using unclassified messages, similarly encoded, composed by the Trojan horse and transmitted from the command and control system site to the diplomatic post. The messages appeared, to superficial examination, to be routine logistics accounting messages containing tabular data. In fact, this data could be used to encode binary information (such as software upgrades for the Trojan horse itself).

The first action the Trojan horse took after installing itself was to announce its presence and location to its operator. It was decided to minimize the risk of detection by not further exercising the Trojan horse unless its operation would yield a decisive military or diplomatic advantage. Thus, for most of the years of its existence, the Trojan horse was inactive and remained undetected.

The interconnection of all of the command and control system sites increased the potential value of the Trojan horse to the enemy, as the means now existed to subvert the entire command and control system network. The enemy now felt prepared to commence hostilities, based, in part, on the successful (and undetected) exercise of the Trojan horse several weeks earlier, verifying that it still existed and was operable. It is probable that the Trojan horse was reprogrammed at this time so it could utilize the newly available remote access to other command and control system sites and could perform the precise intelligence and disinformation tasks needed for successful completion of the enemy mission.

The actual use of the Trojan horse during hostilities was carefully crafted to avoid detection. Prior to planned enemy thrusts, the Trojan horse transmitted selected allied order of battle information to the operator, who then passed the information to the enemy intelligence system. During enemy attack phases, the Trojan horse was used to introduce small distortions in enemy track and locating data. These modifications to system behavior were subtle enough to escape detection, but provided decisive intelligence and disinformation advantages, leading to the attainment of the enemy objectives during the hostilities.

*The opportunity for exploitation exists.* This scenario is, of course, intentionally alarming. (Similar scenarios have been described in articles [SCHE79, GRAN83] published in military professional journals, indicating that the tactical significance of computer security problems has not been lost on military professionals.)

Though this scenario was from the military context, the issues are equally relevant to any context where there are determined attackers. The credibility of such scenarios is based on the following factors.

Contemporary systems are vulnerable to attack by any individual with access to their hardware and/or software components at any point in their life cycle. The exposure of systems to attack has dramatically increased as systems are interconnected. The increasing use of preexisting and commercial off-the-shelf (COTS) software (for example, graphics

packages), an increasingly attractive option, also significantly increases the exposure of systems, as the developers of such software must be regarded as having access to the system.

Potential attackers are capable of exploiting opportunities to penetrate mission-critical systems. No particular skills beyond those of normally competent computer professionals are required. In fact, such skills are now within the publicized repertoire of amateur “hackers.” Trojan horses designed to penetrate particular systems can often be located “off the shelf” from underground bulletin board systems.

It must be assumed that potential attackers are motivated to exploit available opportunities. It would be imprudent to assume that potentially hostile interests will be restrained, particularly where decisive advantages are at stake.

We must assume that a serious penetration attempt will be indirect in nature, will not require direct physical access by the penetrator and/or operator to the penetrated target, and will not advertise its presence or cause easily observable disturbances to the system’s behavior. In short, a serious penetration attempt will be quite unlike those of amateur penetrators (hackers) occasionally receiving media publicity.

This follows simply from the difference in goals assumed for the threats: Hackers generally have motivations related to the need for self-esteem and notoriety. A professional penetrator is motivated to remain undetected and effective for lengthy periods of time. What malicious software does after initial system penetration is influenced primarily by the goals of its author, not by technical difficulty. The nature of a professional threat may be assessed, even by computer nonprofessionals, by taking any of the recently publicized cases and estimating whether the penetration would have ever been detected if its author had wished it to remain indefinitely covert.

### **Perspective on the computer security problem**

If we review the computer security “war stories” cited above, a number of facts become clear. The problems of human error are significant but not of substantial interest to the determined, hostile attacker. There are other, higher payoff methods of attacking the information in a computer system that render this problem less interesting. Proper training of users and good system administration are prerequisites to solving this problem. Further, protection against probing and penetration is necessary.

The computer is almost irrelevant to the issues of user abuse of authority. The problem is as old as the storage and manipulation of information. The solutions that applied in the era of ledger papers apply as well to equivalent records stored in a computer. To the extent that it can automate such methods as audits, cross-checks, and consistency

checks, the computer can actually improve security against user abuse of authority.

User probing involves exploitation of computer security controls that have insufficient power, or controls that are improperly used. The user who probes the system is acting as a normal user and will be controlled by security measures that have adequate flexibility and are carefully applied.

Penetration depends on weakness in the implementation of a system's hardware or software security controls. Even if a system incorporates rich and flexible security features that are carefully used and conscientiously maintained, it may be vulnerable to penetration. To resist penetration, the controls themselves must be built "right." History has taught us that the challenge of building a penetration-proof system is very great indeed.

Finally, subversion of the security mechanism itself is most difficult to prevent. Not only must controls be built "right," as mentioned in the preceding paragraph, but they must also be built nonmaliciously and be simple and small enough for analysis to determine that they perform as expected (or at least do not perform as not expected). Essays 2 and 6 introduce measures that have been found effective against probing, penetration, and subversion of security mechanism. The reference monitor concept introduced in Essay 2 gives us a set of principles that can be applied to the design or selection of security features and to their implementation in ways that provide a high degree of resistance to penetration and a high degree of assurance that they are not subverted.

### **For further reading**

More real examples of computer misuse can be found in the literature [BLOO90, HAFN91, SPAF89, STOL89]. In addition, Neumann and Parker [NEUM89] summarize and discuss classes of computer misuse techniques. These references, as well as the others cited in this essay, can help the reader gain a broader understanding of "what there is to worry about."

### **Acknowledgments**

We would like to give a special acknowledgment to the contributions of Steven B. Lipner of Trusted Information Systems. Substantial portions of the text as well as several key insights for classifying threats are drawn from material he previously prepared and he has graciously permitted us to use in this essay.

Acknowledgment is also given to P.A. Myers, from whose work [MYER80] we have drawn excerpts for incorporation into this essay.

Myers' fine thesis also contributed significant ideas to the discussion of malicious software and subversion presented here.

In addition, we acknowledge (and have excerpted) two early articles [COX79, SCHE79] that reached deeply into the cultures of their respective audiences to awaken them to the computer security problem.

We also gratefully acknowledge the substantial constructive comments and suggestions provided by James P. Anderson on an earlier version of this essay.

