IJIS (2001) 1: 3–13 / Digital Object Identifier (DOI) 10.1007/s102070100003

Computer security

Carl E. Landwehr

Mitretek Systems, 7525 Colshire Drive, McLean, VA 22102, USA

Published online: 27 July 2001 –

Abstract. A strong factor in the early development of computers was security – the computations that motivated their development, such as decrypting intercepted messages, generating gunnery tables, and developing weapons, had military applications. But the computers themselves were so big and so few that they were relatively easy to protect simply by limiting physical access to them to their programmers and operators. Today, computers have shrunk so that a web server can be hidden in a matchbox and have become so common that few people can give an accurate count of the number they have in their homes and automobiles, much less the number they use in the course of a day. Computers constantly communicate with one another; an isolated computer is crippled. The meaning and implications of "computer security" have changed over the years as well. This paper reviews major concepts and principles of computer security as it stands today. It strives not to delve deeply into specific technical areas such as operating system security, access control, network security, intrusion detection, and so on, but to paint the topic with a broad brush.

Keywords: Computer security – Vulnerability – Security principles – Security policy – Security mechanisms

1 What is computer security?

Secure has its etymological roots in *se* – without, or apart from, and *cura* to care for, or be concerned about [1]. So, in the broadest sense, we might say a computer is secure if it is free from worry and if it is safe from threats, and computer security is the discipline that helps free us from worrying about our computers. Of course one might be foolishly secure, simply out of ignorance (this is another of the dictionary definitions for the word). In fact, people have worried about the security of their computers for many years, and computer security concerns are a significant factor today in the development and application of computer technology throughout society.

Today, securing a computer for an e-commerce application may mean first assuring that the system will be available for use and will deliver uncorrupted information. Assuring the confidentiality of the information delivered may not be important at all if the system is simply acting as an online catalog of merchandise, though of course if it is used to accept credit card numbers, they will require protection. This emphasis reverses the traditional focus of some military and intelligence organizations on preserving confidentiality.

In military systems, the first generation of computer security measures aimed to *prevent* security violations, and researchers developed technologies that could be counted on to prevent computers from leaking sensitive data. The market adopted few of these technologies, however, and a second generation of security technologies, characterized by firewalls and intrusion detection systems, aimed to at least *detect* and *limit* security violations that could not be prevented. Efforts are now underway to develop a third generation of security technologies and architectures that will have the ability to *tolerate* attacks and continue to provide critical functions, albeit in a degraded mode, while an attack is in progress.[1]

We view computer security as a narrower topic than information security, which would cover all forms of information storage and processing. Nevertheless, our focus is not so much on securing computers themselves (against theft or other physical threats, for example), but on se-

[1] For the characterization of history, I am indebted to Jay Lala of the U.S. Defense Advanced Research Projects Agency.

curing the data that they receive, store, and retransmit, and securing the processes they perform on those data. At the same time, we recognize that security is a system property and that it is a "weak-link" property. The easiest path for an enemy to get information from a targeted system may be to steal a backup tape, retrieve hard-copy output from a dumpster, or simply steal the victim's laptop. Anyone concerned about overall information security for the system must not ignore such possibilities, and indeed there are technical countermeasures for some of them. For example, we may encrypt the backup tape and secure the encryption key so that the backup tape is of no practical use to the enemy, or we may encrypt the entire hard drive of the laptop. This paper introduces the basic terminology and concepts of computer security and provides a general background for the other papers in this inaugural issue of IJIS.

2 Security policy

Security policy provides the rules of the game in computer security, and this fact explains its prominence, which often surprises newcomers. A policy is simply a set of rules defined to meet a particular goal, in this case to secure a computer or the information it processes. A computer without a security policy is like a society without laws; there can be no illegal acts in the later and no security violations in the former. Events like the "I-LOVE-YOU" virus incident [2], in which the likely perpetrator went unprosecuted because there were no laws against his acts in his country, support this simile, and in fact are stimulating lawmakers worldwide.

The local legal and regulatory framework can itself play a significant role in the security policy for a computer system. A nation or a collection of nations may impose policies on the protection and distribution of information that motivate elements of a system's security policy. The need for forensic information to prosecute computer crimes and the European Union's privacy directive [3] provide current examples, as do regulatory policies on the protection and distribution of healthcare information in the U.S. [4].

Laws may also discourage the development of security measures in computer systems. The U.S.'s Digital Millenium Copyright Act [5], by making any attempt to circumvent technical access controls on copyrighted material illegal, discourages the development of technically strong controls. In the long run, it may also encourage the development of technology to detect violations of the Act and support successful prosecution of violators.

2.1 Security, privacy, and confidentiality

Security, privacy, and confidentiality are sometimes used interchangeably, but for our purposes it is important to distinguish them. Security, as we have already discussed, is the most general of these terms and may, depending on a stated policy, incorporate privacy or confidentiality concerns. Privacy, when used carefully in the context of security policies, connotes matters that can be kept entirely to oneself, whereas confidential matters are those we may entrust to others with the understanding that they will not be further disclosed except in accordance with some implicit or explicit policy. I may entrust my doctor, lawyer, or banker with certain confidential information that is to be disclosed only under proper authority; other matters I may keep entirely private. In this sense, automated systems are generally concerned with preserving confidentiality, at most. The meaning of a private key in a public-key cryptosystem preserves this distinction, in that disclosing a private key to anyone renders it no longer private. However, many discussions of privacy in the popular technical press use the word in a sense much closer to confidentiality. For example, records of video rentals are considered "private" in the U.S., in that the video store may not disclose them, but of course the video store has the record.

2.2 Commercial, military, and other policies

Security policies will depend on the concerns of the owner of the system and the owner of the information processed by it. For many years there were two rather distinct communities concerned about computer security: commercial and military. Commercial concerns naturally focus on the flow and protection of financial assets. The desire to prevent, detect, and prosecute commercial fraud motivated both security policy and technology development in this area and led to controls on application-level programs, so that only authorized individuals could invoke certain operations, and to the generation and preservation of audit trails, so that potential fraud could be identified and the perpetrators identified, prosecuted, and convicted. This class of commercial systems benefits in general from the ability to count its losses and so to determine on a fairly rational basis how much to invest in preventive measures.

The military world[2] suffers from the difficult problem of deciding how to assess in financial terms the damage done by a particular disclosure of sensitive information. A disclosure might compromise an expensive intelligence collection system or a particular military operation and have a tremendous cost in dollars and lives, or it may cause little actual damage. This fact makes it much more difficult for the military to determine how much to invest in mechanisms to prevent such disclosures. For many years, military concerns focused primarily on preserving the confidentiality of sensitive information. Security policies designed to protect such information existed long before computers, but placing sensitive information into computer systems opened many potential vulnerabilities

[2] For purpose of this paper, we lump intelligence systems with military systems.

for its disclosure or corruption. Military investment has fueled much of the research and development in this area since its inception, though perhaps a declining fraction over the past decade or so, as commercial interests and concerns have escalated. The focus of this research has most often been on securing the lower levels of the infrastructure – the operating systems, for example, rather than the applications, both because of the diversity of military applications and the belief that securing the applications without securing the infrastructure would be like building on a foundation of sand.

It is worth noting that the commercial world also has assets that it wishes to protect against disclosure to its competitors and on which it may be difficult to place a specific dollar value. Examples include research and development results, product plans, and corporate financial information. Thus the distinction between military and commercial concerns and policies is not always sharp.

In addition to conventional military and commercial security policies, there are other areas in which security policy definition is crucial and can be contentious. Healthcare information is a clear example: patients may have a strong interest in maintaining the confidentiality of all or some aspects of their medical records, yet they may wish to disclose much or all of it to the physician treating them, and they may have to disclose portions of it to an insurance company for reimbursement, or to a pharmacy in order to fill a subscription. Government regulations may constrain how such information is to be protected and how it may flow [4]; and the security policies for such systems must take these regulations into account. Census information and data generated and processed during elections require similar specialized policies and protections. Confidentiality policies for the vast quantities of data that are typically recorded by commercially interested parties as individuals use the Internet for web browsing and e-commerce are largely unstated and even where stated, compliance is largely voluntary. As is frequently the case with both healthcare and financial data, the data collected concern specific individuals and can be linked to them, but the collector owns the data and controls the security policies applied to them.

Finally, as computers are increasingly used as critical components in control systems, security policies need to take account of the consequences of delayed or denied service as well as corruption or exposure of information. For at least 40 years, computers have played key roles in military systems, but the early systems were isolated and driven to meet specific functional requirements that were divorced from security requirements. Today, computers are highly networked and are part of mission-critical activities not only in military systems, but in a broad spectrum of industries and throughout critical infrastructures that control energy distribution, transportation, water systems, and virtually every aspect of economic life. Security policies must consequently take the role of the system within the enterprise into account.

3 Bounding the system

A security policy has a domain of application that may transcend many individual computer systems. In securing any particular system, a key first step is to identify what is in the system and what is not. This activity defines the *security perimeter* for the system and limits the scope of what needs to be protected and what can be controlled. If we limit this discussion strictly to *computer* security the problem may seem easier, since presumably the perimeter of an individual computer is easy to see.

But today, individual computers are invariably connected to other computers, either intermittently or permanently, and exchange information with them, and it is this information and the functions of these interconnected systems that we want to secure. Further, the configuration of individual client machines that are directly controlled by users can have a significant effect on system security. For example, a user may install a modem on a desktop PC connected to an internal corporate network in order to gain access to that system from a residence. While perhaps well justified in the user's mind, this act opens a path for an intruder to enter the corporate network without going through the carefully configured firewall that the corporation uses to protect itself from outside attacks.

Further, computers increasingly exchange information that is actually intended to be executed, and possibly to be installed permanently, on the other systems with which they are communicating. The security implications of the free exchange of programs and program fragments are significant, as reflected by the recently released U.S. Department of Defense policy on the use of mobile code [6]. When mechanisms intended to improve a system's security posture, such as virus scanners and system administration tools, use mechanisms like these to distribute and install patches and updates, it becomes difficult to simply forbid the use of these mechanisms and essential to verify the source and the integrity of the information received.

One of the purposes of defining a system's security perimeter is to distinguish intruders from legitimate users. An *intruder* is an individual who crosses a system's security perimeter without authorization. Since an intruder may gain access to a system by stealing a legitimate user's identity, and the intruder's behavior on the computer system may be difficult to distinguish from an ordinary user, detecting such intruders is a hard problem [7].

4 Vulnerability, threat, and risk

A *vulnerability* is a weak point in a computer system. It may be a flaw in a piece of software that runs in a privileged mode, a poorly chosen password, or a misconfigured rule enforced by a firewall. It could even be a depen-

dence on a service or piece of information external to the system. Vulnerabilities exist in all of today's commercial operating systems; new vulnerability announcements appear regularly in bulletins from vendors and from incident response organizations such as the Computer Emergency Response Team [8] around the world.

A *threat* is an intent to inflict damage on a system. Different individuals and groups have different abilities to carry out a threat (through *attacks*), and the determination of the nature of threat against which a system must be defended should drive the decisions about its *security architecture* – its structure from the security perspective. The threat posed by a legitimate user of a system is referred to as the *insider* threat. Precisely because they have some degree of authorized access to a system and typically can gain a detailed understanding of how their systems operate without arousing suspicion, the insider threat is a major concern for many systems.

The *risk* assumed by the owner or administrator of a system is the likelihood that the system will not be able to enforce its security policy (including the continuation of critical operations) in the face of an attack. Thus risk is a function of both the exposure of the system's vulnerabilities in the context of its security architecture and the level of threat manifested against the system at a given time. As such, its risk can change over time even though an organization's system and operations have not changed at all. In particular, the decision of a particular group to target a system will increase that system's risk, as will the release of information about a previously undisclosed vulnerability in a piece of software that that system uses, and, even more, the release of a script that makes the exploitation of the vulnerability easy [9].

Organizations can take different approaches toward risk; a *risk avoidance* philosophy focuses on removing every vulnerability and defeating every attack. The adoption of widely used and inherently vulnerable systems has made such an approach untenable for most systems, causing them to adopt what they call *risk management* approaches, in which a degree of risk is accepted as normal and more attention is focused on measures to deal with successful attacks. To reduce the risk posed by insiders, for example, an organization may partition roles and privileges within its computer systems so that no single individual can authorize a large financial transaction. Of course, in some cases, the adoption of a risk management strategy is hard to distinguish from what one might simply call *risk acceptance*.

5 Security properties

For many years, the conventional definition of computer security required that the computer system maintain three properties:

- *confidentiality*: assuring that computer-based information is not disclosed without proper authorization;
- *integrity*: assuring that computer-based information is not modified without proper authorization; and
- *availability*: assuring that computer-based information is accessible to legitimate users when required.

More recently, it has become common to add two more properties:

- *authentication* (or sometimes *identification and authentication*): assuring that each principal is who they claim to be; and
- *non-repudiation*: assuring that a neutral third party can be convinced that a particular transaction or event did (or did not) occur.

Authentication is really a prerequisite for the first three properties, since without proper authentication it is not possible to determine whether a disclosure or modification has been properly authorized. Non-repudiation is primarily of interest in the context of communication protocols, particularly for legal or financial transactions. Non-repudiation, when offered as a service that can be provided to assure the origin, transport, delivery, etc., of a message usually depends on the use of cryptography, and the compromise of one of the keys involved in providing the service could nullify its guarantees in a particular case.

6 Security principles

If you want to keep a secret, don't tell anyone and don't write it down. Or, as Benjamin Franklin's Poor Richard put it, "Three may keep a secret, if two of them are dead [10]." Some security principles like these carry over to the world of computer security, where they can affect the design of security functions provided by a system, the user interfaces used to invoke those functions, and the way those functions are implemented within the computer system. This section reviews a few such principles that have proven relevant to computer security. When considering a particular point of design, some of these principles can conflict, requiring an informed choice of tradeoffs.

6.1 Accountability

People behave better (or at least more in accordance with laws or policies) if they know that they can later be called to account for their actions. For computer systems, this observation has several implications: first, adequate *authentication* mechanisms are needed so that responsibility can be conveyed from a human principal to a process within the computer system. Second, functions within the computer system that are controlled by the security policy must be *authorized*. Since the authorization decision typically must take into account the identity and role of the requestor, authentication is a prerequisite for proper authorization decisions. Finally, security critical actions need to be *audited* so that, after the fact, the responsibil-

ity for initiating the action can be reliably traced back to an individual.

Implementing authentication, authorization, and audit procedures in a way that could, for example, provide the basis for a criminal prosecution is a significant technical challenge that spans many layers of a computer system. Audit trails generated by recording the invocation of operating system functions, for example, are often too voluminous and too fine-grained to be of very much use in monitoring system activity for security violations and can only with difficulty be used to reconstruct the details of a security violation detected through other means. An audit trail generated by the application itself can be more effective, since the application presumably captures the semantics of the security sensitive operation. However, the application must rely on the file system and the operating system to safeguard the audit trail against fraudulent modification.

6.2 Least privilege

The principle of least privilege asserts that each process (or other entity) in a computer system should be granted only those privileges needed for it to accomplish its designated function [11]. In some respects, this is a correlate of the need-to-know principle in systems to protect classified information – even though an individual may have the clearance for a set of information, the information should only be provided if that person has a specific need to know it. Otherwise, a secret is being revealed to another individual, increasing risk, without any particular benefit.

Implementing this principle in computer systems always involves tradeoffs. Perfect application of the least privilege principle would require that each process could read and write only those files, or those parts of particular files, essential to its particular purpose, for example. Simply defining this policy for each program in a system would be a significant burden, and the mechanisms needed to enforce it would likely be cumbersome. Further, there would be a constant flow of policy changes as programs were revised or added new functions. On the other hand, today this principle seems largely forgotten or at least ignored. Typically, every application run on a personal computer has full access to the entire file system and can cause great damage to it, either accidentally or maliciously. Many viruses exploit violations of this principle in order both to damage systems and to propagate themselves.

6.3 Minimize the variety, size, and complexity of trusted components

Any component of a system, be it human or computer, that has the authority or ability to compromise the security of a system must be trusted not to do so. To establish that a component is worthy of such trust requires effort, and if the component is a computer program or a piece of hardware, the effort is generally proportional to its size and complexity. Minimizing the variety, size, and complexity of the trusted components in a system design can reduce both assurance cost and risk, because it is easier to assure that they are correctly specified and implemented. This principle aligns well with the least privilege principle, and is fundamental to the Trusted Computer System Evaluation Criteria (TCSEC) [12], which provide some of the earliest and strongest guidance to those trying to build computer systems to meet strict security (and particularly confidentiality) requirements. The highest TCSEC evaluation levels require the most stringent minimization of the Trusted Computing Base – that portion of a hardware/software system responsible for enforcing security properties.

Some care is required in the application of this principle, however, for it can lead to performance penalties. If, for example, a design centralizes all security checking in one component that runs in a separate, protected domain, it may be very frequently invoked and require a large number of domain-crossing operations. Centralized, general purpose security kernels [13] generally suffered from this problem [14]. Current work to develop language-based security mechanisms aims for simple enforcement mechanisms, but ones that are distributed throughout the program [15, 16].

6.4 Default security

This principle says that system defaults should be arranged with a preference for secure operation. According to this principle, a computer system should be configured, at delivery, so that its operation will be secure and that relaxing the security controls should require a conscious act on the part of an administrator. Systems that come with widely known default user IDs and passwords already installed, or with defaults that permit system wide sharing of all files, or administrator privileges for every user violate this principle. Default security could be understood as an application of the least privilege principle.

6.5 Defense in depth

This is the "don't put all your eggs in one basket" principle. It recognizes that security mechanisms may be imperfect and fail; consequently designs should employ multiple, diverse, and complementary mechanisms, so that attacks that exploit the failure of one mechanism will be stopped by a different one. This principle may seem to be at odds with minimizing the variety and complexity of trusted functions, and indeed the management of diverse defensive measures poses a significant problem to system administrators today. However, the limited scope of many security mechanisms, as well the potential that they contain flaws, makes defense in depth a prudent principle for today's computer systems.

7 Security mechanisms

The need for competing interests to share a common set of computing resources lies at the heart of many computer security requirements. Although a personal computer may be owned and operated by a single individual, its operating system, file system, and applications in effect represent a variety of competing and cooperating interests.

7.1 Defining domains

The primary security mechanisms used to construct computer systems that can enforce specific security policies effectively set up fences (and gates) within computers. A fenced area in a computer system corresponds roughly to a *domain*, which we define as a collection of data and a set of authorizations for manipulating that data within a computer system [17, 18]. When a system boots up, establishment of an initial domain that meets security requirements is crucial; witness the many viruses that attempt to install themselves in such a way that they will gain control whenever the system is restarted. In addition to mechanisms for creating domains, mechanisms to control communication between domains are required.

Domains can be built using hardware features of the underlying computer system, using software only, or, as is usually the case, some combination of software and hardware. Mechanisms to control access to privileged instructions, such as those controlling hardware level input-output, changes to the privilege state of the machine, and the mapping from virtual to physical memory addresses are key to defining domains at the most basic level. The development of sophisticated mechanisms for defining protection domains, including hardware support, is part of the history of the development of time-sharing systems, with the MULTICS hardware and operating system designs [19, 20] representing a high point. The Intel 80286 microprocessor and its successors actually incorporated scaled-down versions of mechanisms developed for MULTICS, but most operating systems and applications developed for it have exercised relatively few of the features provided.

Separation that is based only on software-implemented mechanisms tends to be somewhat more fragile than separation that has hardware assistance, because if the code, once initiated, is not prevented by the hardware from accessing other domains, then some certification is needed that each piece of code that is executed will not, in fact, violate the intended domain boundaries.[3] This approach was reasonably successful in some Burroughs machines [21], and has reappeared in current Java language systems, which attempt to certify that a program of Java bytecode is safe to run before dispatching it. As the introduction of lightweight processes and threads has weakened the boundaries between computations in order to reduce the time needed to swap contexts, methods for introducing inline checks to assure security properties have gained interest, as reflected by the work of Wahbe et al. [15] on software fault isolation and Schneider [22] on security automata. Certifying code before it is run has also gained interest, as reflected by work on proof-carrying code [23] and static flow analysis [24] of computations.

[3] Of course hardware-protected domains depend on software to properly set up the data structures on which the hardware bases its enforcement, but this is a more restricted problem.

7.2 Linking users with domains

As we have seen, security requires that actions be attributable to individuals. This principle leads to a requirement for mechanisms that link users and domains. Identification (the user asserting who she/he is) and authentication (providing evidence that the assertion is valid) are the processes needed to establish this link. Providing a user ID and password continues to be the most common method for identification and authentication, but the use of tokens (smart cards or other devices that support challenge/response protocols, for example) and biometrics for these purposes is growing. These different forms of authentication depend on three factors: something a user *knows* (a password), something a user *has* (a token) and something a user *is* (a fingerprint, iris scan, or other biometric). In general, the more factors provided in an authentication, the stronger the authentication is considered to be, though of course the strength of the particular factors and mechanisms involved should be taken into account.

A key element in an authentication process is that there is a trustworthy path between the source of the authentication data and the decision-making component – if an untrustworthy component can either introduce or collect information that is crucial to the authentication process, authentications could be spoofed. In order to foil eavesdroppers, some authentication protocols incorporate a challenge–response sequence in which the correct response differs for each authentication.

Today, an authentication process typically succeeds or fails – the level of evidence presented is not usually captured or available for input to subsequent policy decisions. The development of public-key infrastructures in which certificates (certified public keys) are distinguished based on different levels of authentication of the individual by the issuing organization, or by different levels of confidence in the issuing organization, could change this.

Once an individual identity is linked with a domain, the activities of programs executing in that domain typically inherit the privileges of that user. It is worth noting at this point that if a user ID is shared by several different people, the link between the user and the domain is substantially weakened. For example, it may be convenient to

set up a "sysadmin" ID that has full privileges for the system and is to be shared by several system administrators. The ID exists so that administrators can conduct routine tasks from a less privileged account and thereby be less at risk if they should make a mistake or invoke a malicious program. Nevertheless, it will be quite difficult to identify which individual has taken a particular action.

7.3 Authorizing operations

Once a domain is created and linked to a user, it becomes possible to decide whether an attempt by a program executing in that domain to gain access to a particular resource is permitted or not – whether a given access is authorized. A *reference monitor* [25] is a component that decides, based on some data structure embodying a security policy, whether a given reference is authorized under that policy. If the object is a file, for example, it may have an associated *access control list* that specifies which users are permitted to read, write, or execute that file. When a program attempts to open a file for reading, the reference monitor would be invoked to determine whether the user associated with the requesting domain is authorized to perform the requested operation. If so, the file is opened and the operation proceeds; if not the operation is denied and an error of some sort is reported.

To be effective, a reference monitor must be able to mediate every access request (complete mediation), it must be small and simple enough so that there is high assurance that it performs its functions correctly (correct), and it needs to be able to protect itself from being corrupted (tamperproof). These requirements [25] typically lead to a small, centralized reference monitor that operates in a domain separate from the requesting program. Such designs also incur frequent crossing of domain boundaries, raising performance issues. Recent research [22] seeks to overcome these problems by having a post-processor insert the reference monitor checks throughout the code to be executed. This approach avoids the domain-crossing overhead, but requires high confidence in the mechanisms used to determine what checks to insert and in assuring that all code executed has been modified in this way.

Authorization policies and mechanisms typically exist at a variety of levels within a computer system. A login process may determine whether a given user is authorized to use the system at all, the file system can decide whether the logged-in user is authorized access to particular files, and an application such as a database management system may decide which fields of which records in a file the user may view. If the user can gain read access to the files used by the database without going through the database application, the database's security controls are ineffective – the database fails to achieve the status of being a reference monitor for its data because its mediation is not complete. So, it is critical that the developer who wishes to enforce application-specific security controls understand the context in which the application will execute and the operation of the underlying authorization mechanisms in the operating system. The development of role-based access control policies and mechanisms [26] represents an attempt to organize the complexity of application-based policies and to associate access privileges with job-related functions (roles) rather than individuals (users). Users can then be authorized to assume certain roles that carry with them complex, prespecified sets of application-specific authorizations.

Network security is beyond the scope of this paper, but we note in passing that firewalls implement authorization policies on the users of a computer system as well. The firewall may permit or prohibit traffic to or from particular ports and addresses. Similarly, filtering software operating on a user's computer tries to play the role of a reference monitor with respect to browser access to websites, barring access to sites that violate the policy represented by the software vendor's filter list.

Some security policies, such as those that call for certain restrictions on information flow among domains or objects within computer systems, cannot be enforced precisely by authorization mechanisms like reference monitors [27]. In practice, however authorization mechanisms are used to enforce an approximation of such policies.

7.4 Auditing operations

Audit logs are broadly useful for system administration and maintenance in case of software or equipment failures. Their use in computer security raises different issues, in that they can be an important link in assuring accountability for security-relevant actions.

Normally, one doesn't expect system users to attempt to turn off or corrupt audit logs, but if an intruder gains access to a system and wishes to cover his tracks, defeating the audit mechanism will be as important to him as avoiding surveillance cameras is to an ordinary thief. The audit mechanism itself, then, must be able to resist attacks. There is an apparent conundrum here: if we could protect the audit mechanism, couldn't we protect the entire machine? And if we can't protect the audit mechanism, what good is it? In fact, not all intruders will be aware of or knowledgeable enough about security audit mechanisms to disable them. Moreover, there are mechanisms, such as generating audit trails on a write-once device, or sending it to a remote, protecting machine, that can make it very hard for an intruder to destroy the audit log, even if she manages to prevent further writes to it.

The real difficulties with security audit arise in identifying the events worth auditing and preserving them in a way that they can be reviewed effectively. Auditing done at the level of the operating system can be consistent, but will likely consist of a huge record of relatively tiny events. Such logs can be used to reconstruct an intruder's activities once the intrusion has been identified, but reviewing the log to determine whether an intrusion

has occurred or not is likely to be impractical. Auditing conducted at the level of an application makes it easier to identify the meaning of a sequence of system calls, but each application will generate its own particular kind of audit log. Nevertheless, this is probably the more promising avenue to pursue. Today, security audit logs are probably most useful in keeping honest people honest, and secondarily as a source of input to automated intrusion detection systems.

7.5 Cryptography

A cryptographic algorithm transforms cryptographic key and readable (plaintext) data into cyphertext that can only be understood by applying another (possibly the same) cryptographic key and crypto-algorithm to it. If the keys and algorithms are the same, we have a *symmetric* or *secret-key* crypto system. If the algorithm involves two different keys, one for enciphering and the other for deciphering, we have an *asymmetric* or *public-key* algorithm. Public-key algorithms have the benefit that one of the two keys can be openly broadcast, and as long as the other key is kept private, information enciphered under the public key can be deciphered only by the holder of the private key. Further, information enciphered under the private key can be deciphered by anyone, but can have been generated only by the holder of the private key. This feature can be used to generate a digital signature, binding the holder of the private key to the contents of a document. In general, public-key algorithms are much more computationally intensive than private-key algorithms, so they are usually applied only to relatively small pieces of data (such as the keys for symmetric algorithms).

The past decade has seen increasing application of cryptography to computer security problems. Today, some commercial operating systems offer the user the option of keeping the entire hard drive encrypted, for example. Taking advantage of this feature offers the user some protection against the theft of a hard drive or laptop, but can also make loss of the encryption key a catastrophe.

Cryptography is probably finding its widest use today (and perhaps its widest use in history) in securing and validating information in e-commerce applications. Secure sockets layer (SSL) and transport layer security (TLS), used to encrypt business transactions conducted over the World Wide Web, rely on public-key cryptosystems to generate and communicate a shared session key that is used with a private (symmetric)-key cryptosystem to prevent eavesdroppers from stealing, for example, credit card numbers in transit over the Internet. The protocols do nothing, however, to secure the computer systems at either end. As Gollman [28] and others have observed, cryptography can transform a communication security problem into a key management problem. If the key management problem is simpler than the communication security problem, this is a benefit. However, the question of protecting the key, if it is stored on a computer, brings us back to the domain of computer security mechanisms.

Cryptographic mechanisms are finding their way into intellectual property protection schemes [29] and a variety of other applications. In each case, the security of the scheme overall must rest on the secrecy of the key to some cryptographic algorithm. If protecting this key can be made easier than protecting an entire operating system, we will have realized a net gain in security. It seems increasingly likely that if public-key infrastructures become widespread, there will be a significant demand for tokens such as smart cards that can store a user's private keys and encrypt data using that key as well, so that the private key never leaves the token. Although tokens are themselves vulnerable to attack, they are probably less vulnerable than the average operating system. They can be constantly with the user, and a token can be constructed to authenticate its user, so the effect of theft is reduced.

8 Assurance

When security mechanisms work properly, they should be invisible, or nearly so, to their users. If they get in the way when there is in fact no security problem, people will find a way to work around them or will at least stop paying attention to them. The dialog box that pops up too frequently ceases to be read, or even thought about, by its respondent. The implication is that it can be very difficult to distinguish a situation in which the security mechanism functions properly and no security violations are occurring and a situation in which the security mechanism is simply not working. How does the user of a cryptographic device have confidence that the stream of bits emerging from it has in fact been encrypted using the intended key?

The answer is that she relies on the assurance supplied by the crypto provider that the device functions as specified. Providing this assurance for a straightforward, single-purpose hardware encryption device is hard enough; providing it for a computer system is much more challenging. In general, assurance that a program or device functions as intended can rest on three kinds of evidence: evidence that the device was built by well-trained, motivated, and knowledgeable people; evidence that the process used to build the device is sound and when properly followed will produce a device that meets its specification; and evidence from testing and analyzing the product directly. All of these kinds of evidence presuppose that the desired behavior of the device (at least, for our purposes, from a security standpoint) is well specified.

The strongest of these three kinds of evidence comes from examination of the actual device. If the mechanisms work properly, it doesn't really matter who built it

or what process they used. However, today these mechanisms are frequently computers, and both the hardware and software may be quite complex. Any direct examination of the artifact produced will either not be exhaustive or will be extremely labor intensive. Further, the standard of specification required to resolve whether the device exhibits the proper security behaviors is likely to be far beyond commercial practice for ordinary software and hardware. For the highest level of assurance, formal models of the security policy that the machine is intended to enforce, as well as corresponding formal specifications for the enforcement mechanisms, may be required [30].

The first general approach to dealing with the security assurance problems for general-purpose computing systems is documented in the U.S. *Trusted Computer System Evaluation Criteria* (the "Orange Book") [12]. This pioneering work laid down a structure that grouped both security mechanisms and assurance requirements into a set of increasingly rigorous levels. The intent was that the government (and other organizations) would specify the level of security they required in their computer systems according to these levels. At the same time, computer vendors would submit their systems to the government for evaluation, with the results to be published and then used to determine which systems met procurement requirements. Although carefully thought out, the plan turned out to be both expensive and time-consuming for both the government and the vendors. See [31] for a thorough treatment of criteria and evaluations.

Subsequently, other nations developed different sets of evaluation criteria and processes that explored slightly different approaches, for example splitting the requirements for security function and assurance, and using commercial firms to conduct product evaluations. The result has been the development of the "Common Criteria" approach [32]. Several nations have now adopted this approach and agreed to recognize each other's evaluations, at least up to a specified level of assurance.

The Common Criteria approach calls for the development of a Protection Profile to capture a common set of security requirements (e.g. for a firewall) and a Security Target, which is the baseline against which a particular product (the Target of Evaluation) is judged. In this way, a product developer can specify, and pay to have his product evaluated against, a Security Target that could conform to several different Protection Profiles. The customer can specify or select a Protection Profile that meets its needs and identify which products could satisfy them according to the Security Targets that have been successfully evaluated against.

9 System considerations

That security is a system problem has never been more apparent than it is today. A variety of e-mail lists and websites announce vulnerabilities and corresponding patches in a steady stream. The shortage of system administrators with security training has prompted the U.S. government to organize new programs of incentives to induce universities to develop training courses and students to enroll in them.

It is easy to argue that the bulk of reported security incidents today are the consequence of inadequate system administration. Indeed it is a daunting task to be faced with managing the security of a typical office environment where individual users can modify the hardware and software configuration of their client machines, each new software release can be expected to bring a new crop of vulnerabilities, and each new week a new set of viruses. Recent reports [9] document that the bulk of incidents reported to the CERT occur after scripts are released that automate the exploitation of particular vulnerabilities, and that patches for these vulnerabilities have been available well before the scripts are distributed. Administrators simply have not been able to assure that these patches are installed promptly on the machines that need them.

Managing the configuration of a collection of systems includes the hardware as well as the software. Even if an institution's Internet connections are protected by carefully configured firewalls, the existence of a machine behind the firewall with a forgotten modem attached to it can open the system to a dangerous attack. The only real way to manage this kind of problem is through a combination of software tools to probe the configuration regularly for vulnerabilities [33], administrative procedures to assure that changes to the configuration are well understood and in accordance with overall security policies, and training to assure users are aware of security issues and the consequences of unsafe practices. Even so, as the I-L-Y virus incident showed graphically [2], the defenses provided by most systems to a targeted attack that deviates even slightly from known patterns are startlingly weak.

Software is increasingly delivered to systems in small units, downloaded from Internet sites. Application-level macros, scripts, plug-ins, Java applets, Active-X controls, Javascript, and more stream into a client through a web browser, are installed, and run. Because this mode of operation is routine for many users, and because the integrated application packages they use often lack internal barriers or security controls, these users are vulnerable to a wide variety of attacks if their operating systems function perfectly. Perhaps we should be more surprised at how few incidents there are, rather than the costs of the ones that occur. In any case, our growing willingness to accept and install software with relatively weak assurance of its pedigree or value makes the risk profile of the current computing environment quite different from what it was 20 years ago. The combination of worldwide networking, mobile code, and integrated applications is a volatile mix that can allow an attack originated in a far corner of the earth to have worldwide effects in a matter of hours.

At the system level, the only real countermeasure available to protect against these kinds of attacks is virus scanning deployed both on the desktop systems and at the corporate firewall. Although the companies marketing this technology do their best to keep up and are attempting to develop quick response methods to deploy new filters, when an outbreak occurs, the attacker continues to hold the advantage. There is, however, some technology in the research pipeline [34–36] that holds the promise of limiting both the damage certain types of malicious mobile code can cause and also its ability to propagate an attack.

In addition to being susceptible to attacks from mobile and malicious code, many Internet-connected systems are vulnerable to a variety of attacks against well-known vulnerabilities in operating systems or applications. These attacks typically take the form of probes to determine system configuration information, from which potential vulnerabilities can be identified, followed by the sending of crafted packets and messages that try to exploit the hypothesized vulnerabilities.

A well-administered firewall is an essential first line of defense against such probes and attacks, but it will never be a complete solution. In fact, as more services are multiplexed over port 80, which few systems wish to block, the effectiveness of firewalls may diminish. The increasing use of encryption to cover Internet traffic could have a similar effect, since it may make it impossible for the firewall to monitor content.

Intrusion detection systems are another weapon in the system administrator's arsenal, Today, these systems work primarily by looking for signatures of known attacks, and so they are subject to the same problem with novel attacks as virus scanners. In fact, since most attacks are sequences of operations that may occur over a period of time, interspersed with normal traffic, they can be much more difficult to recognize than a virus. Consequently, many users of the systems experience high rates of false alarms, which predictably cause them to turn down the sensitivity of the detector, which in turn increases the probability of false negatives.

Intrusion tolerance is a new approach to system architecture that takes as a given that some attacks against a system will succeed and strives to organize system resources so that even in the face of a partially successful attack, critical system functions can be maintained, perhaps in a degraded mode. Although research is active in this area [37, 38], it is too early to assess results.

To summarize, today's approach to software development, testing, and distribution, combined with current operating systems and system architectures, presents significant vulnerabilities and a very challenging security problem for any system administrator. Not only more and better trained administrators, but also a technically more manageable and securable infrastructure, is needed.

10 Unsolved problems

We could list a number of detailed open technical problems in computer security: easier ways to specify security policies; more efficient, more flexible enforcement mechanisms; faster encryption algorithms; and so on. Computer security problems have been the target of research and development for 30 years. Principles for building secure computing systems have been identified and mechanisms have been developed that implement them. There is a thriving market in firewalls, intrusion detection systems, and the like. Why do we see an increasing number of computer security problems?

1. Unsafe programming practices. For example, the major source of exploitable security flaws continues to be code that writes into a buffer without checking its length first. This fact should be a major embarrassment to the entire software development community. Solutions to this problem are nearly as old as programming languages.
2. Unsafe design choices, particularly at the application layer. Developers increasingly integrate features across applications, so that an e-mail package can automatically open and display a document or spreadsheet attached to an incoming message. They incorporate scripting languages capable of executing arbitrary programs. These mechanisms can support many convenient services, and users often embrace them. But providing such services with so little regard for security that they simultaneously enable automatic worldwide distribution of destructive software is like heating a room by setting the rug on fire.
3. Complex, hard-to-manage system architectures. As long as it takes an expert to configure systems and networks so that their security flaws are not exposed, there will be plenty of opportunities for intruders.

One view of this situation is that the fundamental research and development challenges have been met, and that the problem is transitioning what we have learned into the systems we build. There is some truth in this view, but a more even-handed assessment might be that the research community has not, so far, been able to generate solutions that are both simple enough to implement, low enough in cost, and easy enough to use that they are embraced by customers.

In either case, today's information security engineer is faced with the problem of building a trustworthy system from untrustworthy components [39], which is akin to building a silk purse from the proverbial sow's ear. The only workable solutions to date demand some minimum number of trustworthy components to be used in conjunction with the untrustworthy ones. It is the components that provide critical security services, such as cryptographic devices, authentication devices, firewalls, virtual private network components, and so on, that are being relied on to provide overall system security and that

therefore could justify the extra investment in design and assurance that the security technologies entail.

Acknowledgements. I first thank the editors for giving me the opportunity to write this paper and for their patience during its drafting. I also thank Fred Schneider for comments on an early outline that helped me shape the structure and contents. I acknowledge, though I cannot list, the numerous conversations and papers from my friends and colleagues in the computer security community that have contributed to my present understanding of the field, on which I based this paper. Finally, I gratefully acknowledge the recent books by Dieter Gollman [28] and Bruce Schneier [40] which I found helpful in clarifying some details during the writing.

References

1. Murray J (1971) Compact edition of the Oxford English dictionary. Oxford University Press, New York
2. Markoff J (2000) An 'I love you' virus becomes anything but. The Week in Review, New York Times, 7 May 2000
3. European Parliament (1995) European Union Directive 95/46/EC: On the protection of individuals with regard to the processing of personal data and on the free movement of such data. url: `http://wwwdb.europarl.eu.int/oeil/oeil_ViewDNL.ProcedureView?lang=2&procid=59`
4. U.S. Department of Health and Human Services (2000) Health insurance reform: standards for electronic transactions, 45 CFR Part 160 and 162. Office of the Secretary of Health Care Finance Administration, 20 August 2000. url: `http://aspe.hhs.gov/admnsimp/final/txfinal.pdf`
5. Congress of the United States of America (1998) U.S. Digital Millenium Copyright Act. 28 October 1998, Public Law 105-304. url: `http://thomas.loc.gov/cgi-bin/query/z?c105:H.R.2281.ENR:`
6. U.S. Department of Defense (2000) ASDC3I – Memorandum of 7 November 2000. Policy guidance for use of mobile code technologies in Department of Defense information systems
7. McHugh (2001) Intrusion and intrusion detection. Int J Inf Secur, this volume
8. CERT Coordination Center. Vulnerability notes database. Software Engineering Institute, Carnegie Mellon University, Pittsburgh. url: `http://www.kb.cert.org/vuls`, `http://www.cert.org`; Forum of Incident Response and Security Teams. url: `http://www.first.org`
9. Arbaugh WA, Fithen WL, McHugh J (2000) Windows of vulnerability: a case study analysis. IEEE Comput 33(12):52–59
10. Franklin B (1968) Poor Richard's almanac. In: Bartlett J; Beck EM (eds.) Bartlett's Familiar Quotations, 14 edn. Little, Brown, Boston, p 421b
11. Saltzer JJ, Schroeder MD (1975) The protection of information in computer systems. Proc IEEE 63(9): 1278–1308
12. U.S. Department of Defense (1985) DoD trusted computer system evaluation criteria (The "Orange Book"). DoD 5200.28-STD
13. Schell RR, Downey PJ, Popek GJ (1973) Preliminary notes on the design of secure military computer systems. MCI-73-1, ESD/AFSC, L.G. Hanscom Field, Bedford, Mass. url: `http://csrc.ncsl.nist.gov/publications/history/sche73.pdf`
14. Landwehr CE (1983) Best available technologies for computer security. IEEE Comput 16(7): 86–100
15. Wahbe R, Lucco S, Anderson TE, Graham SL (1993) Efficient software-based fault isolation. In: Proc. 14th ACM Symp. on Operating Systems Principles, ACM, Ashville, NC, pp 203–216
16. Urlingsson U, Schneider FB (2000) IRM enforcement of Java stack inspection. In: Proc. 2000 IEEE Symp. on Security and Privacy. IEEE CS Press, Los Alamitos, CA
17. Lampson BW (1971) Protection. In: Proc. 5th Princeton Symp. on Information Science and Systems, Princeton U., pp 437–443; (1974) ACM SIGOPS Operat Sys Rev 8(1): 18–24
18. Landwehr CE, Carroll JM (1984) Hardware requirements for secure computer systems: A framework. Proc. 1984 IEEE Symposium on Security and Privacy, Oakland, CA, April 1984, pp 34–40
19. Saltzer JH (1974) Protection and the control of information sharing in MULTICS. Commun ACM 17: 388–402
20. Organick EI (1972) The MULTICS system: an examination of its structure. MIT Press, Cambridge, Mass.
21. Organick EI (1973) Computer system organization: the B5700/B6700 series. Academic Press, New York
22. Schneider FB (2000) Enforceable security policies. ACM Trans Inf Syst Secur 3(1): 30–50
23. Necula GC, Lee P (1998) The design and implementation of a certifying compiler. In: Proc. of the ACM SIGPLAN '98 Conf. on Programming Language Design and Implementation, ACM, pp 333–344
24. Myers AC (1999) JFlow: practical mostly-static information flow control. In: Proc. 26th ACM SIGPLAN-SIGACT Conf. on Principles of Programming Language, San Antonio, pp 228–241
25. Anderson JP (1972) Computer security technology planning study, ESD-TR-73-51, ESD/AFSC. 01731 [NTIS AD-758 206], Hanscom AFB, Bedford, Mass. url: `http://csrc.ncsl.nist.gov/publications/history/ande72.pdf`
26. Sandhu RS, Coyne EJ, Feinstein HL, Youman CE (1996) Role-based access control models. IEEE Comput 29(2): 38–47
27. McLean JD (1994) A general theory of composition for trace sets closed under selective interleaving functions. In: Proc. 1994 IEEE Symp. on Research in Security and Privacy, Oakland, CA. IEEE CS Press, Los Alamitos, CA, pp 79–93
28. Gollman D (1999) Computer security. Wiley, Chichester
29. Secure Digital Music Initiative (1999) Portable device specification, version 1. Part 1, version 1.0, 1999. url: `http://www.sdmi.org/download/port_device_spec_part1.pdf`
30. McLean J (2001) IJIS. To appear
31. Lipner S (2001) IJIS. To appear
32. Common Criteria Implementation Board (2000) Common criteria for information technology security evaluation, version 2.1. Aligned with ISO/IEC International Standard (IS) 15408, September 2000. url: `http://csrc.ncsl.nist.gov/cc/ccv20/ccv2list.htm#CCV21`
33. Ram P, Rand DK (1995) Satan: double-edged sword. IEEE Comput 28(6): 82–83; nmap security scanner: manual page, url: `http://www.nmap.org/nmap/nmap_manpage.html`
34. Balzer RM, Goldman NM (2000) Mediating connectors: a non-bypassable process wrapping technology. In: Proc. of the DARPA Information Survivability Conference and Exposition (DISCEX 2000). IEEE CS Press, Los Alamitos, CA, Vol. 2, pp 361–368
35. Fraser T, Badger L, Feldman M (2000) Hardening COTS software with generic software wrappers. Proc. of the DARPA Information Survivability Conference and Exposition (DISCEX). IEEE CS Press, Los Alamitos, CA, Vol. 2
36. Ghosh A, Burrer D, Hollebeek T, Pelican M (2001) Sandboxing mobile code execution environments. DARPA Contract #F30602-99-C-017. url: `http://www.cigitallabs.com/research/sandboxing.html`
37. Pal P, Loyall JP, Schantz RE, Zinky JA, Webber F (2000) Open implementation toolkit for building survivable applications. In: Proc. of the DARPA Information Survivability Conference and Exposition (DISCEX 2000), IEEE CS Press, Los Alamitos, CA, Vol. 2, pp 197–210
38. Wylie JJ, Bigrigg MW, Strunk JD, Ganger GR, Kiliccote H, Khosla PK (2000) Survivable information storage systems. IEEE Comput 33(8):61–68
39. Schneider FB (ed) (1999) Trust in cyberspace. Committee on Information Systems Trustworthiness, National Research Council. National Academy Press, Washington, D.C.
40. Schneier B (2000) Secrets and lies. Wiley, New York. 1 For this characterization of history, I am indebted to Jay Lala of the U.S. Defense Advanced Research Projects Agency. 2 For purposes of this paper, we lump intelligence systems with military systems. 3 Of course hardware-protected domains depend on software to properly set up the data structures on which its enforcement is based, but this is a more restricted problem