Reflections on System Trustworthiness

PETER G. NEUMANN*

Computer Science Laboratory
SRI International
EL-243, 333 Ravenswood Ave
Menlo Park, CA 94025-3493
USA

Abstract

We examine here a range of concerns relating to computer systems and networks, with particular attention to difficulties in system development, and the resulting vulnerabilities, threats, and risks. We examine some approaches that might achieve dramatic improvements in the ability to develop, operate, and use trustworthy systems. The problems and their solutions typically require a combination of technology and social policy.

*Principal scientist.

ADVANCES IN COMPUTERS, VOL. 70

ISSN: 0065-2458/DOI: 10.1016/S0065-2458(06)70006-7

1. A Total-System Perspective

Each of the following items presents pressing challenges relating to the constructive use of information technology. The totality of all the interrelated challenges requires concerted efforts that transcend the individual problems.

• *Trustworthiness*. Trustworthiness implies simply that something is worthy of being trusted to satisfy its expected requirements. Users often trust systems that are not worthy of being trusted, with respect to attributes such as system and network security, system reliability and survivability, human safety, interoperability, predictable system behavior, and other important attributes that are for the most part not receiving enough concerted attention. Computer-communication infrastructures are typically riddled with flaws. In the absence of more serious attacks, governments and system developers seem to have been lulled into a false sense of security. At present, neither proprietary nor source-available system developers are sufficiently militant in satisfying critical needs. In mass-market software, the patch mentality seems to have won out over well-designed and well-implemented systems.

• *Total system life-cycle issues*. Developing and operating trustworthy systems is inherently difficult today. Typically, a system is not likely to be trustworthy unless the relevant attributes were explicitly recognized from the beginning of system development, reflected in sound system architectures and software development, explicitly addressed in system procurements, and their fulfillment mandated throughout system operation.

• *System development practice*. Costly failures have occurred in developing large systems, such as the modernization efforts for the US Internal Revenue Service, US and UK air traffic control systems, the FBI Virtual Case File, and German TollCollect, to name just a few. Procurement and development of large-scale hardware/software systems remains a high-risk activity, with cost overruns, delays, and even abandonment of entire projects.

• *The Internet*. Increasingly, many enterprises are heavily dependent on the Internet, despite its existing limitations. Internet governance, control, and coordination create many contentious international problems. The Internet infrastructure itself is susceptible to denial-of-service attacks and compromise, while the lack of security and dependability of most attached systems also creates problems (e.g., penetrations such as open relays being used to host zombies and "bots"). Worms, viruses, and other malware are often impediments, as are ubiquitous problems of spam e-mail and phishing attacks that may result in identity theft.

• *Critical national infrastructures*. Despite some past recognition of the pervasiveness of serious vulnerabilities, critical national infrastructures such as electrical power, energy, telecommunications, transportation, finance, and government continuity are typically still vulnerable to attacks and accidental collapses. For example, massive power outages keep recurring, despite supposed improvements. Telecommunications outages can have severe consequences, as can transportation shutdowns and fuel shortages. Furthermore, these infrastructures have interdependencies that result in widespread system failures.

• *Accountability*. Oversight of computer activities is often as weak as oversight of corporate practices. On the other hand, audit mechanisms must also respect privacy needs. As one example that fails on both counts, today's unauditable all-electronic voting systems are lacking in accountability in a would-be effort to protect voter privacy. In fact, without the addition of some sort of voter-verified audit trail, they provide no meaningful assurances that votes are correctly recorded and processed. (For example, see the October 2004 special issue of the *Communications of the ACM*, devoted to the integrity of election systems.)

• *Privacy*. Privacy is something that many people do not value until after they have lost it. Personal privacy is relevant pervasively in our lives, especially in financial matters and health care. Some advocates of homeland security have postulated the need to sacrifice privacy in order to attain security, although the necessity of this tradeoff is highly debatable. Sacrificing privacy does not necessarily result in greater security. (Benjamin Franklin's quote is apt in this regard: "Those who would sacrifice liberty for security deserve neither.") Furthermore, serious inroads to privacy protection have occurred that may be difficult to reverse. Surveillance is becoming more widespread, but often without adequately respecting privacy. Legitimate needs for anonymity or at least pseudoanonymity (for example, to protect victims and legitimate whistle-blowers) must not be suppressed or dismissed as dangerous.

• *Education*. In many countries, university curricula in software engineering and trustworthiness inadequately reflect the needs of critical systems. Instruction is often aimed at programming in the small, while more or less ignoring systems in the large. This situation has potentially serious long-term implications worldwide.

As noted above, it is the totality of these problems that is of primary concern. Simplistic local approaches are not effective. Greater foresight and pervasive system-oriented thinking are urgently needed, along with greater private-public cooperation.

2. Anticipating Disasters

As Henry Petroski noted over twenty years ago [42], we generally learn less from successes than from failures. The ACM Risks Forum [32] and *Computer-Related*

Risks [34] include a startling number of failures and risks, and provide a goldmine of opportunity for anyone who wants to learn from past mistakes. Intriguingly, or perhaps ironically, most of the content of [34] is still as relevant today as it was in 1995. The same types of failures continue to recur, and the range of causes remains much the same. Indeed, the scope and extent of the risks has increased steadily. For example, the ACM Risks Forum continues to report computer system development fiascos and operational failures of aircraft, air-traffic control, defense systems, train crashes, electrical power, telecommunications, medical health systems, and financial problems. These difficulties include problems in reliability, system survivability, security, privacy, and human well-being. Some of these problems have been escalating dramatically, such as spam, phishing attacks, identity thefts, and financial losses.

In recent years, some unusual natural disasters have occurred, such as the 9.0-magnitude Indonesian earthquake that triggered a tsunami killing more than 200,000 people in 11 countries around the Indian Ocean, the exceptionally heavy 2006 hurricane season in the Caribbean area including the devastating effects of Katrina, and a major mudslide in La Conchita, California. Although failures of information technology obviously had no role in *triggering* these disasters, IT systems could play significant roles in *anticipating, detecting, monitoring, and responding to* such events, minimizing losses of life, injuries, and consequential damages. What have we learned from such events, especially with respect to the need for proactive contingency plans?

For example, a tsunami detection and early-warning system such as had already been deployed in the Pacific Ocean could also have been used in the Indian Ocean. Such a system could have given timely warnings to millions of people, and could have saved many lives *if* local authorities had citizen alerts and evacuation plans in place. Early warnings and preparedness for hurricanes and typhoons are improving as computer prediction of possible storm paths is becoming more accurate and as many authorities prepare disaster response plans and train for their deployment. However, preparedness tends to improve only after disasters have occurred (and then often only temporarily). In the case of the mudslide in the hills above La Conchita, which followed an awesome sequence of rainstorms, sensors in the hills were designed to trigger advance warnings, which evidently were not taken seriously enough. (A similar slide had occurred in an adjacent area nine years earlier, and insurance companies had already declined to provide future coverage.)

Several problems arise in connection with developing detection and warning systems.

• Institutions (especially governments, corporations, and defense departments) tend to fashion response plans for past situations rather than for potentially devastating future situations. Unless a similar disaster has recently occurred in a similar venue under similar conditions, few people worry about low-probability high-impact events. A comparable tendency holds for trustworthy computing. A computer networking event not unlike a tsunami occurred in 1988—namely, the Internet Worm [47,54] that affected about 10% of the 60,000 Internet hosts active at the time. As a result, an emergency response team (now US-CERT) was formed to help coordinate responses and warn of vulnerabilities. Prior to the year 2000, a large upgrade effort to avoid the Y2K crisis was generally successful; the situation could have been much more serious without the intensive remediation efforts. Today, many new threats such as malware and terrorist attacks could easily disable critical infrastructures and the Internet. However, because the cybersecurity equivalent of a tsunami seems extremely unlikely to many people unfamiliar with the nature of the vulnerabilities, there is little interest in mounting efforts to increase system trustworthiness and engage in other preventive measures. The consequences of major meltdowns could be very dramatic, especially if accompanied by terrorist attacks.

- Institutions tend to optimize short-term costs and ignore long-term consequences. Also, farsighted analyses of what might happen are always subject to poor assumptions, faulty reasoning, and mandates to reach self-serving conclusions. This is discussed further in Section 8.

- People generally do not like to make unnecessary preparations, and often resent taking sensible precautions. Repeated false warnings tend to inure them, with a resulting loss of responsiveness. Even justifiable warnings that are heeded (such as the Y2K remediation or boarding up for an oncoming hurricane) are often denigrated if the resulting effects are only relatively minor.

It is clear that much greater attention needs to be devoted to predicting, detecting, and ameliorating both natural catastrophes and unnatural computer-related misuse, attacks, disasters, and outages. Efforts are needed to dramatically improve the trustworthiness of those systems on which many lives depend, and to make those systems more tolerant to human misbehavior as well as malfunctions and natural causes.

3. Trustworthiness

Estimates of system trustworthiness ultimately depend on having some sort of logical basis for confidence that a system will predictably satisfy its critical requirements. Measures of trustworthiness are particularly important for information security, system integrity and reliability, human safety, fault tolerance, and overall enterprise survivability in the face of wide ranges of adversities (including malfunctions, deliberate attacks, and natural causes).

Many lives increasingly depend on critical national infrastructures—all of which in turn depend in varying degrees on the predictable behavior of computer-communication resources. Indeed, these infrastructures often depend on the Internet for information and control and may be vulnerable to attacks from any attached computer systems.

Unless critical information system resources are sufficiently trustworthy, other systems are at risk from failures and subversions. Unfortunately, for many of the key application domains, the existing information infrastructures are lacking in trustworthiness. For example, power grids, air-traffic control, high-integrity electronic voting systems, the emerging US Department of Defense Secure Global Information Grid, national infrastructures, and many collaborative and competitive Internet-based applications all need systems that are more trustworthy than we have today or are likely to have in the foreseeable future.

Numerous steps are needed to develop trustworthy systems. Consider an analogy with the planet's natural environment—expectations for which are somewhat similar to expectations for trustworthy information systems. For example, pure air and uncontaminated water are vital, as are the social systems that ensure them.

Although poorly chosen analogies can be misleading, the analogy with the natural environment is appropriate. Each of the following items is applicable to both trustworthy information systems and natural environments.

- Their critical importance is generally underappreciated until something goes fundamentally wrong—after which undoing the damage can be very difficult if not impossible.
- Problems can result from natural circumstances, equipment failures, human errors, malicious activity, or a combination of these and other factors.
- Dangerous contaminants may emerge and propagate, often unobserved. Some of these may remain undetected for relatively long periods of time, whereas others can have immediately obvious consequences.
- Once something has gone recognizably wrong, palliative countermeasures are typically fruitless—too little, too late.
- Your own well-being may be dramatically impeded, but there is not much you as an individual can do about aspects that are pervasive—perhaps international or even global in scope.
- Detection, remediation, and prevention require cooperative social efforts, such as public health and sanitation activities, as well as technological means including increased trustworthiness.
- Up-front preventive measures can result in significant savings and increased human well-being, ameliorating major problems later on.

- As discussed further in Section 8, long-term thinking is relatively rare. There is frequently little governmental or institutional emphasis on proactive prevention of bad consequences. Many of the arguments against far-sighted planning and remediation are skewed, being based on faulty, narrowly scoped, or shortsighted reasoning—especially relating to short-term profits rather than long-term savings and other benefits.
- Commercial considerations tend to trump human well-being, with business models sometimes considering protection of public welfare to be detrimental to corporate and enterprise bottom lines.

In some contexts, pure water is becoming more expensive than oil. Fresh air is already a crucial commodity. Short- and long-term effects of inadequately trustworthy information systems can similarly be costly. Proactive measures are as urgently needed for system trustworthiness as they are for breathable air, clean water, and environmental protection. It is difficult to remediate computer-based systems that were not designed and implemented with trustworthiness in mind. It is also difficult to remediate serious environmental damage.

Anticipating and responding to compelling long-term needs does not require extraordinary foresight, whether for air, water, reversing global warming, or trustworthy systems upon which to build infrastructures. Our long-term well-being—perhaps even our survival—depends on our willingness to consider the future and commitment to taking appropriate actions.

4. Risks in Trusting Untrustworthiness

The Internet provides ample opportunity for proving the age-old truism, "There's a sucker born every minute." Carnival-style swindles and other confidence games once limited to in-person encounters are now proliferating electronically, worldwide, at low cost and effort. Blatantly obvious pre-Internet examples are the so-called Nigerian-style postal scams that requested use of one's bank account to help move money; hoping for a proffered generous commission, the suckers are then separated from their assets. These scams have been updated to today's e-mail phishing and e-mail scam attacks that efficiently harvest personal information from vastly more people, and are considerably more sophisticated—for example, replicating a legitimate website in every respect except for perhaps just one hard-to-detect bogus URL. Indeed, it is becoming increasingly difficult to distinguish the real from the bogus, and people continue to be victimized.

Many other kinds of scams, stings, and misrepresentations also exist. Deceptive unsolicited e-mail (spam) offering bogus goods and services opens up new avenues for fraud and identity theft. Online activities are emerging with glaring opportunities for swindles, manipulations, and assorted malfeasance, such as online auctions (with irregularities such as nondelivery and secondary criminality), an alarming increase in highly sophisticated phishing attacks, Internet gambling, and fraudulent websites (e.g., with deceptive URLs creating the appearance of legitimacy). Any of these and other situations could result in inordinate risks, such as financial ruin, blackmail, compromised democracy, or even loss of life. But it is perhaps not surprising that people fall for such schemes, particularly when the technology superficially appears genuine.

Today's unauditable paperless all-electronic voting systems present significant risks (see Section 9). The risks are even greater for voting over the Internet. With independent accountability seriously lacking today, e-voting can be likened to using an off-shore gambling site not subject to any regulation and managed by unknown and unaccountable agents.

We tend to trust third-party relationships with banks, telephone companies, airlines, and other service providers whose employees have in some way earned our trust, collectively or individually. But what about untrustworthy third parties? Some computer-based applications rely critically on the putative integrity and noncompromisibility of automated trusted third parties, with little if any easily demonstrated human accountability. Examples include digital-certificate authorities, cryptographic servers, surveillance facilities, sensitive databases for law enforcement, and credit-information bureaus. With appealing short-term cost incentives for pervasive use of outsourcing, the need for demonstrably trustworthy third-party institutions becomes even more important. However, security, privacy, and accountability are often ignored in efforts to save money.

Is placing trust in offshore enterprises inherently riskier than using domestic services? Not necessarily. Corruption and inattention to detail are worldwide problems. The deciding factor here is the extent to which comprehensive oversight can be maintained.

Is domestic legislation enough? Of course not. Any legislation should not be overly simplistic; for example, it should avoid seeking solely technological fixes or purely legislative solutions to deeper problems. Besides, serious complexities arise from the fact that such problems are international in scope and demand international cooperation.

Is there a role for liability (for flagrant misbehavior or injurious neglect) and differential insurance rates—for example, based on how well a purveyor is living up to what is expected of it? Such measures have significant potential, although they will be strongly resisted in many quarters.

So, how can we provide some meaningful assurance that critical entities such as direct or third parties are sufficiently trustworthy? Ideally, institutions providing, controlling, managing, and monitoring potentially riskful operations should be decoupled from other operations, avoid conflicts of interest, and be subject to rigorous independent oversight. Enronitis and collusion must be avoided, even if it means that the costs are greater. Furthermore, the people involved need altruism, sufficient foresight to anticipate the risks, and a commitment to effectively combat those risks. At the very least, their backgrounds should be free of criminal convictions and other activities that would cast serious suspicions on their trustworthiness. In addition, legislators need to be able to see beyond the simplistic and palliative, to approaches that address the real problems. Above all, the entire populace must become more aware of the risks and the concerns outlined above, especially to the inherent combination of technology and policy issues.

This conclusion should not be a surprise. Overall, there are many risks that must be addressed. The old Latin expression "Caveat emptor" (Let the buyer beware) is even more timely today.

5. Principles for Developing Trustworthy Systems

Everything should be made as simple as possible—but no simpler.
Albert Einstein

Developing trustworthy systems with complex requirements is inherently a complex challenge. In general, simple solutions are hopelessly inadequate in such cases. On the other hand, enormously complex systems—even if they purport to be trustworthy—are likely to be unmanageable, from the perspective of developers, system administrators, application operators, and end-users.

Ideally, there should be some middle ground. In particular, the recommended approach, considered in Section 6, is to develop trustworthy systems as conceptually sound predictable compositions of simpler components, perhaps even with provably sound combinations of provably sound components.

In anticipation of that approach, a relevant set of principles can be helpful in increasing trustworthiness—if the principles are used intelligently as guidelines for system development and operation.

5.1 Saltzer–Schroeder Security Principles

The ten basic security principles formulated by Jerry Saltzer and Mike Schroeder [51] in 1975 are all still relevant today, in a wide range of circumstances. They are actually of broader interest than just with respect to security. For example, each one is also relevant to reliability, survivability, and human safety. In essence, these principles are summarized as follows (overly tersely), for present purposes:

- *Economy of mechanism*: Seek design simplicity (wherever and to whatever extent it is effective).
- *Fail-safe defaults*: Deny accesses unless explicitly authorized (rather than permitting accesses unless explicitly denied).
- *Complete mediation*: Check every access, without exception.
- *Open design*: Do not assume that design secrecy will enhance security.
- *Separation of privileges*: Use separate privileges or even multiparty authorization (e.g., two keys held by different entities) to reduce misplaced trust.
- *Least privilege*: Allocate minimal (separate) privileges according to need-to-know, need-to-modify, need-to-delete, need-to-use, and so on. The existence of powerful mechanisms such as *superuser* is inherently dangerous.
- *Least common mechanism*: Minimize the amount of mechanism common to more than one user and depended on by all users.
- *Psychological acceptability*: Strive for ease of use and operation—for example, with easily understandable and forgiving interfaces.
- *Work factors*: Make cost-to-protect commensurate with threats and expected risks.
- *Recording of compromises*: Provide nonbypassable tamper-resistant and tamper-evident audit trails of evidence.

These are of course basic guidelines, not hard-and-fast rules—especially in light of some potential mutual contradictions. Two fundamental caveats must be recognized. First, each principle by itself may be useful in some cases and not in others. Second, when taken in combinations, groups of principles are not necessarily all reinforcing; indeed, they may in some cases conflict with one another. Consequently, development must consider appropriate use of each principle in the context of the overall effort. Examples of a principle having both positive and negative aspects are scattered through the following discussion.

The Saltzer–Schroeder principles grew directly out of the MIT/Honeywell/BellLabs Multics experience (e.g., [40]) begun in 1965 and discussed further later in this section. Each of these principles has taken on almost mythic proportions among the security elite, and to some extent buzzword cult status among many fringe parties. Therefore, we do not explain each principle in detail—although considerable depth of discussion is needed for successful application of each principle. Careful reading of the Saltzer–Schroeder paper [51] is recommended if it is not already a part of your library. Matt Bishop's security books [7,8] are also useful in this regard, placing the principles in a more general context.

Various caveats are considered in Section 12.

TABLE I
APPLICABILITY OF SALTZER–SCHROEDER PRINCIPLES

Principle	Trustworthiness	Assurance
Economy of mechanism	is a vital aid to sound design. Exceptions must be handled completely.	can simplify local analysis.
Fail-safe defaults	simplifies design, use, operation, maintenance.	can simplify analysis.
Complete mediation	is vital, but beware of compromise from below.	can simplify analysis locally.
Open design	Secret designs do not preclude compromise. Open design can inspire stronger system security.	Open designs do not preclude compromise. Open design encourages independent analysis.
Separation of privileges	avoids many types of common flaws.	focuses analysis more precisely.
Least privilege	limits effects of flaws; simplifies operation.	focuses analysis more precisely.
Least common mechanism	avoids certain common flaws.	modularizes analysis.
Psychological acceptability	is relevant to usability and operations.	Ease of use is helpful, must anticipate crises.
Work factors	are misleading if systems can be compromised from outside/within/below.	give a false sense of security if unexpected compromises are ignored.
Compromise recording	is an after-the-fact diagnostic and deterrent.	is only an indirect contributor.

Table I summarizes how each of the Saltzer–Schroeder principles can contribute to the goals of trustworthiness and assurance, particularly with respect to security, reliability, and other survivability-relevant requirements. Intriguingly, most of these principles can also be helpful in enhancing composability.

In particular, complete mediation, separation of privileges, and allocation of least privilege are beneficial to composability and trustworthiness. Open design can contribute significantly to composability, when subjected to internal review and external criticism. (See Section 6.) However, conflicts persist about the importance of open design with respect to trustworthiness, with some people still clinging tenaciously to the notion that security by obscurity is sensible—despite risks of many flaws being so obvious as to be easily detected externally, even without reverse engineering. Indeed, the recent emergence of good decompilers for C and Java, along with the likelihood of similar reverse engineering tools for other languages, suggests that such attacks are becoming steadily more practical. Overall, the pretense of keeping a design secret and the supposed unavailability of source code are realistically not significant deterrents, especially with ever-increasing skills among black-box system analysts. However, there are cases in which reliance on security by obscurity is unavoidable—as in the hiding of private and secret cryptographic keys, although the cryptographic algorithms and implementations can be public.

Fundamental to trustworthiness is the extent to which systems and networks can avoid being compromised by malicious or accidental human behavior and by events such as hardware malfunctions and so-called acts of God. In [35], we consider *compromise from outside, compromise from within*, and *compromise from below*, with fairly intuitive meanings. These notions appear throughout this report.

In theory, there are various cases where certain weak links can be avoided (such as zero-knowledge protocols that can establish a shared key without any part of the protocol requiring secrecy, Byzantine algorithms, and k-out-of-n cryptography), although in practice they may be undermined by compromises from below (involving trusted and supposedly trustworthy insiders subverting the underlying operating systems) or from outside (involving penetrations of the operating systems and masquerading as legitimate users).

From its beginning, the Multics development was strongly motivated by a set of principles—some of which were originally stated by Ted Glaser and Neumann in the first section of the first edition of the Multics Programmers' Manual in 1965. (See http://multicians.org.) It was also driven by extremely disciplined development. For example, no coding effort was begun until a written specification had been approved by the Multics advisory board; also, with just a few exceptions such as low-level device drivers, all the code was written in a subset of PL/I just sufficient for the needs of Multics, for which the first compiler (early PL, or EPL) had been developed by Doug McIlroy and Bob Morris.

In addition to the Saltzer–Schroeder principles, further insights on principles and discipline relating to Multics can be found in a paper by Fernando Corbató, Jerry Saltzer, and Charlie Clingen [12] and in Corbató's Turing lecture [11].

5.2 Further Principles

An earlier view of principled system development was given by Neumann in 1969 [33], relating to what is often dismissed as merely "motherhood"—but which in reality is both profound and difficult to observe in practice. The principles under consideration in that paper included automatedness, availability, convenience, debuggability, documentedness, efficiency, evolvability, flexibility, forgivingness, generality, maintainability, modularity, monitorability, portability, reliability, simplicity, and uniformity. Some of those attributes indirectly affect security and trustworthiness, whereas others affect the acceptability, utility, and long-term future of systems.

Considerable discussion in [33] was also devoted to (1) the risks of local optimization and the need for a more global awareness of less obvious downstream costs of development (e.g., writing code for bad—or nonexistent—specifications, and having to debug really bad code), operation, and maintenance (see Section 8); and (2) the benefits of higher-level implementation languages (which prior to Multics were rarely used for the development of operating systems [11,12]).

In the context of developing predictably trustworthy systems, an expanded set of principles is listed below. Although most of them might seem more or less obvious to advanced developers, there are interpretations, hidden issues, and potential pitfalls for their successful implementation. As a result, a seemingly paradoxical situation arises: understanding and experience are required in order to make optimal use of the principles. Thus, the learning experience is essentially iterative.

- *Sound architecture*. Recognizing that it is better to avoid design errors early than to attempt to fix them later, composable architectures inherently capable of evolvable, maintainable, robust implementations are required. Furthermore, good interface design is as fundamental to good architectures as is their internal designs. Both the architectural structure and the architectural interfaces (particularly the visible interfaces, but also some of the internal interfaces that must be interoperable) can benefit from careful specification.
- *Abstraction*. The primitives at any given logical or physical layer should be relevant to the functions and properties of the objects at that layer, and should mask lower-layer detail where possible. Ideally, the specification of a given abstraction should be in terms of objects meaningful at that layer, rather than requiring lower-layer (e.g., machine-dependent) concepts. Abstractions at one layer can be related to the abstractions at other layers in a variety of ways, thus simplifying the abstractions at each layer rather than collapsing different abstractions into a more complex single layer. Particularly useful examples of abstraction include trustworthiness kernels, virtual machine monitors, and similar layered defenses.
- *Modularity*. Modularity relates to the characteristic of system structures in which different entities (modules) can be relatively loosely coupled and combined to satisfy overall system requirements, whereby a module could be modified or replaced as long as the new version satisfies the given interface specification. In general, modularity is most effective when the modules reflect specific abstractions and provide encapsulation within each module (see the next item).
- *Encapsulation*. Details that are relevant to a particular abstraction should be local to that abstraction and subsequently isolated within the implementation of that abstraction and the lower layers on which the implementation depends. One example of encapsulation involves information hiding—for example, keeping

internal state information inaccessible to the visible interfaces [41]. Another example involves masking the idiosyncrasies of physical devices from higher-layer system interfaces, and from the user interfaces as well.

- *Layered and distributed protection*. Protection (and generally defensive design for security, reliability, and so on) should be distributed to where it is most needed, and should reflect the semantics of the objects being protected. With respect to the reality of implementations that rely on—and perhaps pass through—entities of different trustworthiness, layers of protection are vastly preferable to flat concepts such as single sign-on (i.e., where only a single authentication is required). With respect to psychological acceptability, single sign-on has enormous appeal; however, it can leave enormous security vulnerabilities as a result of compromise from outside, from within, or from below, in both distributed and layered environments. Overall, psychological acceptability can conflict with other principles, such as complete mediation, separation of privileges, and least common privilege.
- *Constrained dependency for integrity*. Dependencies on less trustworthy entities should be avoided unless potential negative effects can be somehow confined or constrained. However, it is possible in some cases to surmount the relative untrustworthiness of mechanisms on which certain functionality depends—as in various types of trustworthiness-enhancing mechanisms (see [36]). In essence, do not trust anything on which you must depend—unless you are adequately satisfied with demonstrations of its trustworthiness or the ability to surmount its relative untrustworthiness. This intuitive extension of Biba's notion of multilevel integrity [6] is considered further in Section 6.
- *Architectural minimization of what must be trustworthy*. Appropriate trustworthiness should be situated where it is most needed, suitable to overall system requirements, rather than required uniformly across widely distributed components (with potentially many weak links) or totally centralized (with creation of a single weak link and forgetting other vulnerabilities). Trustworthiness is expensive to implement and to ensure. Thus, significant benefits can result from minimizing what has to be trustworthy. This principle can contribute notably to sound architectures. In combination with economy of mechanism, this provides avoidance of both bloatware and adverse dependence on less trustworthy components. For example, in some cases a simple end-to-end check can determine the presence of intermediate compromises and avoid the necessity of trusting everything else for integrity (apart from denial-of-service attacks).
- *Object orientation*. The OO paradigm bundles together abstraction, encapsulation, modularity of state information, inheritance (subclasses inheriting the attributes of their parent classes—e.g., for functionality and for protection), and

subtype polymorphism (subtype safety despite the possibility of application to objects of different types). This paradigm facilitates programming generality and software reusability, and if properly used can enhance software development. This is a contentious topic, in that most of the OO methodologies and languages are sloppy with respect to inheritance.

- *Separation of policy and mechanism*. Statements of policy should avoid inclusion of implementation-specific details. Furthermore, mechanisms should be policy neutral where that can be advantageous in achieving functional generality. However, this principle must never be used in the absence of understanding about the range of policies that needs to be implemented. There is a temptation to avoid anticipating meaningful policies, deferring them until later in the development—and then discovering that the desired policies cannot be realized with the given mechanisms. This is a characteristic chicken-and-egg problem with abstraction.
- *Separation of duties*. In relation to separation of privileges, separate classes of duties of users and computational entities should be identified, so that distinct system roles can be assigned accordingly. Distinct duties should be treated distinctly, as in activities of system administrators, system programmers, and unprivileged users.
- *Separation of roles*. Concerning separation of privileges, the roles recognized by protection mechanisms should correspond in some readily understandable way to the various duties. For example, a single all-powerful superuser role intrinsically violates separation of duties, separation of roles, separation of privilege, and separation of domains. The separation of would-be superuser functions into separate roles (as in Trusted Xenix) is a good example of desirable separation. Once again (as with single sign-on), there is a potential conflict between principles: the monolithic superuser mechanism provides economy of mechanism, but violates other principles. In practice, all-powerful mechanisms are sometimes unavoidable, and sometimes even desirable despite the negative consequences (particularly if confined to a secure subenvironment). However, they should be avoided wherever possible.
- *Separation of domains*. Concerning separation of privileges, domains should be able to enforce separate roles. For example, a single all-powerful superuser mechanism is inherently unwise, and is in conflict with the notion of separation of privileges. However, separation of privileges is difficult to implement if there is inadequate separation of domains. Separation of domains can help enforce separation of privilege, but can also provide functional separation (as in the Multics ring structure, a kernelized operating system with a carefully designed kernel, a capability-based architecture, or a virtual machine monitor). The prin-

ciple of least common mechanism is also somewhat related. It is desirable to avoid sharing of trusted multipurpose mechanisms, including executables and data, thereby minimizing the use of all-powerful mechanisms such as *superuser* and shared buffers (such as the historically seminal FORTRAN `common`). As one example of the flaunting of principles, exhaustion of shared resources provides a huge source of covert storage channels, whereas the natural use of a common calendar clock provides a source of covert timing channels.

- *Sound authentication*. Authentication is a pervasive problem. Nonbypassable authentication should be applicable to users, processes, procedures, and in general to any active entity or object. Authentication relates to evidence that the identity of an entity is genuine, that procedure arguments are legitimate, that types are properly matched when strong typing is to be invoked, and other similar aspects.
- *Sound authorization and access control*. Authorizations must be correctly and appropriately allocated, and nonsubvertible. Crude all-or-nothing authorizations are often riskful (particularly with respect to insider misuse and programming flaws). In applications for which user-group-world authorizations are inadequate, access-control lists and role-based authorizations may be preferable. Finer-grained access controls may be desirable in some cases, such as capability-based addressing and field-based database protection. However, knowing who has access to what at any given time should be relatively easy to determine.
- *Administrative controllability*. The facilities by which systems and networks are administered must be well designed, understandable, well documented, and sufficiently easy to use without inordinate risks.
- *Comprehensive accountability*. Well-designed and carefully implemented facilities are essential for comprehensive monitoring, auditing, interpretation, and automated response (as appropriate). Serious security and privacy issues must be addressed relating to the overall accountability processes and audit data.

Similar to the summary in Table I the additional principles also tend to contribute to the goals of achieving composability, trustworthiness, and assurance.

At this point in the analysis, it should be no surprise that these and other principles can contribute in varying ways to security, reliability, survivability, and other -ilities. Furthermore, many of the principles and other "ilities" are linked. We cite just a few of the interdependencies that must be considered.

For example, authorization is of limited use without authentication, *whenever identity is important*. Similarly, authentication may be of questionable use without authorization. In some cases, authorization requires fine-grained access controls.

Least privilege requires some sort of separation of roles, duties, and domains. Separation of duties is difficult to achieve if there is no separation of roles. Separation of roles, duties, and domains each must rely on a supporting architecture.

The comprehensive accountability principle is particularly intricate, as it depends critically on many other principles being invoked. For example, accountability is inherently incomplete without authentication and authorization—without which trace-back to the users or originating entities is doubtful. In many cases, monitoring may be in conflict with privacy requirements and other social considerations [16], unless extremely stringent controls are enforceable. Separation of duties and least privilege are particularly important here. All accountability procedures are subject to security attacks, and are typically prone to covert channels as well. Furthermore, the procedures themselves must be carefully monitored. Who monitors the monitors? *(Quis auditiet ipsos audites?)*

6. System Composition: Problems and Potentials

The challenge of developing systems with realistic trustworthiness requirements is inherently complex, despite persistent advice to keep it simple. However, consider the goal of building trustworthy systems using predictably sound compositions of well-designed components along with analysis of the properties that are preserved by, transformed by, or emerging from the compositions. Conceptually, that can greatly simplify and improve development. Indeed, composition is seemingly theoretically relatively straightforward to achieve—especially if we follow the guidance of David Parnas, Edsger Dijkstra, and others. Unfortunately, there is a huge gap between theory and common practice: system compositions at present are typically *ad hoc*, based on the intersection of potentially incompatible component properties, and dependent on untrustworthy components that were not designed for interoperability and whose behavior can undermine the compositions—often resulting in unexpected results and risks. In practice, it is particularly difficult to determine all potentially negative effects of compositions of arbitrary components that were not designed with composition explicitly in mind.

Composition is a concept that is meaningful with respect to many entities, including requirements, specifications, protocols, implemented components, and analytic results such as evaluations and formal proofs. In many cases, the composition of different entities may have unpleasant results.

Other problems may arise because of the order in which operations are carried out, even though the operations may be theoretically commutative or in some broader sense equivalent (perhaps producing different but nevertheless acceptable results).

For example, consider the combination of error-correcting coding (which adds redundancy), compression (which removes redundancy), and cryptography (which ideally makes meaningful content look essentially random). Compressing after encrypting makes little sense, because there is little apparent redundancy. Similarly, compressing after adding redundancy for error correction also makes little sense, because it vitiates the overall error correction. Thus, if such a combination were to be effective, compression should precede encryption, which then should be followed by error-correcting coding.

With regard to subsystem composition, the following are particular concerns.

- *Composability and compositionality*. A distinction is sometimes made between two concepts pertaining to composition. *Composability* relates to the predictability of the preservation or transformation of existing properties under composition. *Compositionality* refers to the predictability of properties that emerge as a result of compositions.
- *Inadequate requirements*. If stated requirements do not explicitly demand that subsystems and other components be developed in ways that encourage compatibility and interoperability, composability is likely to be difficult to achieve. Furthermore, poorly defined requirements are likely to hinder composability.
- *Nonexistent or inappropriate specifications*. If system and subsystem specifications do not adequately define the relationships among interfaces, inputs, internal state information and state transitions, outputs, and exception conditions, and if those specifications are oblivious to critical relationships with related functionality, determining to what extent composability is possible becomes much more difficult. Composition of underconstrained specifications is an inherent problem, because the extent to which the components compose is ill-defined; supposed demonstrations of composability may actually be meaningless. Overly constrained specifications (e.g., including unnecessarily low-level and possibly incompatible details) are also often an impediment to composability. Shared state information across components is a particular source of potential problems.
- *Properties that exist beyond what is defined by stated individual subsystem interface specifications*. Assuming the presence of meaningful specifications, inadequacies of the specifications and inconsistencies between specifications and implementations are characteristic problems. In general, specifications are always inherently incomplete with respect to defining what should *not* happen, even if they are fairly good at defining what should happen. (Abstraction is a very important technique for simplifying specifications, but it suppresses detail that may include undesirable aspects of behavior and may therefore negatively affect compositional properties.) In addition, programming languages and com-

pilers themselves provide very few if any guarantees that something beyond what is expected cannot occur. Examples include shared-buffer interactions and unanticipated information residues from one invocation of a subsystem to a subsequent or concurrent invocation of the same subsystem; buffer overflows and other cases of inadequate bounds checks and inadequate runtime validation; inadequate authentication; improper initialization and finalization; improper encapsulation, which can result in interference and other unexpected interactions; race conditions; covert channels; and intentionally planted Trojan horses. This list represents just the nose of the camel. All these problems can impair composability. As one example, various Windows operating systems are actually relatively modular (which is essential for orderly development), but the modules are not sufficiently encapsulated to prevent adverse effects resulting from composition.

- *Properties that manifest themselves only as a result of combinations of subsystems*. Examples include adverse *emergent* properties (i.e., disruptive or even constructive effects that are not evident in any of the individual subsystems but that arise only when the subsystems are combined); adverse feedback interactions between subsystems, such as infinite loops or dependence on functionality that is less trustworthy; emergent covert channels that do not exist in any of the subsystems in isolation; mutual incompatibilities in the interfaces—perhaps resulting from internal state interference; global failure modes resulting from local faults, as in the 1980 ARPANET collapse [48] and the 1990 AT&T long-distance collapse (e.g., see [34]); so-called "man-in-the-middle" attacks (which might alternatively be called untrustworthy interpositions), in which an interposer can simulate the actions of each component; and other failure modes that arise only in the overall system context. A fascinating noncomposability situation is noted in attempts to combine encryption with digital signatures [4]: signatures are composable with public-key cryptography, but *not* with symmetric cryptography, in which case security may break down. These impediments to composability can arise essentially everywhere throughout the development life cycle—for example, incompatibilities among different requirements and policies, undesirable interactions in specifications and implementations, and difficulties in reconfiguration and maintenance.
- *Multivendor and multiteam incompatibilities*. In the interest of having heterogeneous architectures that enable mixing and matching of alternative components, it may be desirable to use multiple system developers. However, incompatibilities among interface assumptions, the existence of proprietary internal and external interfaces, and extreme performance degradations resulting from the inability to optimize across components can all result in difficulties in composing components.

- *Scalability*. Composability typically creates many issues of scalability. For example, performance may degrade badly or nonpredictably as multiple subsystems are conjoined. Composability can lead to a wide range of expected performance implications—for example, linear, multiplicative, or exponential in the number of composed subsystems. In practice, even further degradations can result—for example, from design or implementation flaws or indirect effects of the composition, such as unrecognized dependence on substantively slow interactions. Obviously, infinite loops and standstill deadlocks ("deadly embraces") are limiting cases of degradation, and often arise as a result of subsystem compositions.
- *Human issues*. Above all, people are the ultimate source of many problems. The supposed "good guys" can accidentally have profoundly negative effects on composability, through poor system conception, inadequate requirements, lack of comprehensive and accurate specifications, bad software-engineering practice, misuse or bad choices of programming languages, badly managed development, and sloppy operational practice (for example). Insider "bad guys" can have various negative effects on the desired composability, such as installing Trojan horses during development, operation, and reconfiguration that impair interoperability and compromise security. Human activities can also directly impair enterprise interoperability [18]. Outsider "bad guys" are generally less likely to negatively affect composability externally, except as a result of penetrations (through which they effectively become bad insiders), subversion of the development process, tampering, and denials of service (often without any internal access required).

There are many desiderata for achieving predictably assured composition, relating to requirements, specifications, implementations, programming languages, configuration information, and analyses thereof. Several relevant issues are noted below.

- *Compatibility and interoperability*. Compatibility implies merely the ability to coexist within a common framework, whereas interoperability additionally implies the ability to work together without adverse side effects. Both are essential prerequisites for composability.
- *Web interoperability*. In recent years, considerable effort has been devoted toward establishing a common definition of a *Web portal* concept that would facilitate universal interoperability providing access to Web services. As one example, Michael Alan Smith [53] has proposed a hierarchical General Portal Model that attempts to unify seventeen different definitions from the literature. From the top, the layers address process interfaces (process identification, transformation), resource discovery (resource identification, resource location, resource binding), and network interfaces (security, network access). In this

context, a portal implies an "infrastructure providing secure, customizable, personalizable, integrated access to dynamic content from a variety of sources, in a variety of formats, *wherever it is needed*." Among other approaches is that of a service-oriented architecture (e.g., [24]).

- *Consistency and completeness of the interface specifications*. Externally discernible functional behavior should be precisely what is specified, implying bilateral consistency of behavior with respect to the functional specifications. That is, the subsystem must do what it is supposed to do, *and nothing else beyond what is specified*. However, because specifications are inherently incomplete, many system failures (in security, reliability, performance, and so on) can result from events that occur outside the scope of specifications and thus are undetectable by any analyses based on those specifications.
- *Independence of specification abstractions*. As noted above, abstraction can be an enormous aid to composability of specifications, as well as to assurance proofs. However, it is essential that the details not explicitly represented by each abstraction be independent of the details of other abstractions. Otherwise, composability will most likely be impaired. One elegant example of provable composability is seen in the orthogonality theorem of Chander, Dean, and Mitchell [9], which provides soundness and completeness proofs for a trust management kernel with a clean separation between authorization and structured distributed naming.
- *Timing and synchronization issues*. In general, Lamport-style safety properties (i.e., nothing bad happens) compose better than liveness properties (something good eventually happens with certainty) [25], but this boundary is blurred by the inclusion of timing constraints, which are technically safety properties, but generally not composable. It is also blurred by the existence of properties that are neither safety nor liveness—such as information flow. Furthermore, time (whether real time or relative time) is typically common to different abstractions, which is a reason that synchronization and timing constraints can present serious impediments to facile composition. For example, see Kopetz [23] on composability in the Time-Triggered Architecture.
- *Explicit state visibility and information hiding*. If a subsystem is stateless (i.e., it does not remember any of its own state information from one invocation to the next), then it is less likely to have adverse interactions when that subsystem is composed with other subsystems—although there are always issues such as noncommutativity of operations and interference during concurrent execution. In addition, nontrivial recovery, as in selective rollback, may be unnecessary. However, statelessness is often not a desirable goal—although stack disciplines effectively separate the internal state information from the subsystem itself and

simplify composability. Assuming that a subsystem is stateful (i.e., it retains at least some of its own state information from one incarnation to the next), there is a choice between the classical notion of information hiding and explicit external visibility of state information (which tends to make explicit any residues that might impair compositionality). On the other hand, because information hiding typically masks internal state information, it can hinder facile composability if there are any implicitly shared states. However, this should be detectable with sensible specifications and implementation. (For example, pointers, loosely bound aliases, and other indirect references tend to create problems.) Thus, the separation of common stateful entities can greatly facilitate composition. Information hiding is also very desirable for other reasons, including isolation, security, system integrity, and tamper resistance.

One interesting historical approach is found in the formal specifications of SRI's Provably Secure Operating System (PSOS [19,38,39]), in which certain state information is hidden but from which the state information that is explicitly visible at the module interface is derived. Because hidden state information could not be accessed outside of the module (information hiding), it could not be referenced in any other module specification. As a result, there can be no module state residues or other state information that can be accessible to other modules or subsequent invocations of the same module beyond what is explicitly declared as visible. This greatly increases the composability of modules and the analysis of potential interactions. It also rules out certain characteristic design flaws simply because it is impossible to represent them in the specifications, even accidentally! (Note that bad implementations can introduce bugs that are not definable in specifications.)

6.1 Other Manifestations of Composition

As noted at the beginning of this section, composition is not limited only to components. It has other manifestations as well.

- *Policy composition*. Serious problems can result when different policies are in conflict or otherwise do not compose properly—especially if that lack of composability is not discovered until much later in development. Furthermore, attempting to compose policies often results in emergent properties that are not evident from the constituent policies. For example, see work by Virgil Gligor et al. with respect to the composability of separation-of-duty policies [21] and application-specific security policies [20]. Gligor notes (among other things) that policy composability does not necessarily imply the usefulness of the resulting policies, and that existing compositionality criteria are not always realistic. Preventing denials of service is a particularly thorny policy; besides, policies that do not address denials of service are inherently incomplete. Of considerable interest is work by Heiko Mantel relating to the general composability of secure system policies and components [28] (e.g., flow properties that are preserved under refinement [27]). Many past efforts are of particular interest to the research community, such as [2,29].

- *Protocol composition*. There is also ongoing work on protocol composability—for example, see [13]. An interesting research challenge might be to consider a particular collection of protocols (e.g., for authentication, encryption, and integrity preservation) and prove that they are mutually composable, subject to certain constraints; the proofs could also be extended to demonstrating that their modular implementations would be composable.
- *Proof composition*. A book on compositionality of proofs [15] is worth careful reading for anyone interested in formal verification and high assurance of systems.
- *Certification composition*. Rushby [49] has characterized some of the main issues relating to the modular certification of an aircraft that is derived from separate certification of its components, based on an extension of a formal verification approach. The crucial elements involve separation of assumptions and guarantees (based on "assume-guarantee reasoning") into normal and abnormal cases.

6.2 Approaches for Predictable Composition

The following approaches can enhance the likelihood of predictable compositions.

- *Dependency analysis*. In many systems, unrecognized interdependencies among different components can hinder composability. Similar comments are relevant to contradictory or otherwise incompatible interdependencies among policies, models, separately compiled software, and even proofs. Identifying such dependencies and removing them or otherwise neutralizing them would be a considerable aid to composability,
- *Constrained and guarded dependency strategies*. The principle of constrained dependency for integrity is introduced in Section 5. Deterministic linearization or other suitable prioritization of intersubsystem dependencies (such as a lattice ordering) can avoid many adverse dependency problems, such as often result from misguided locking strategies and search strategies, compatibility mismatches in system upgrades, and unanticipated distributed interactions. For example, in Dijkstra's THE system paper [17], the use of a linearly ordered hierarchical locking structure guaranteed that no deadly embraces could occur *between* two different layers of abstraction (although in subsequent years a

deadly embrace was occasionally discovered *within* a particular layer). As another example, Biba's multilevel integrity [6] (MLI) requires in essence that no computational entity (e.g., user, program, process, or data) may depend on any other entities that are deemed less trustworthy (i.e., that are potentially less highly trusted) with respect to integrity. In the broadened sense of dependence considered here, the strict lattice ordering of multilevel integrity attributes implied by Biba may be relaxed if any relative untrustworthiness can be masked by creative system architecture or otherwise transcended—as in the trustworthiness-enhancing mechanisms enumerated in [36] as well as other architectural approaches such as isolation kernels and virtual machine monitors. Also, see Abadi et al. [1] for a formalization of dependency.

- *Functional consistency among layers of abstraction*. The 1977 Robinson–Levitt paper [45] on hierarchical formal specifications introduced the concept of formal mappings between different layers of functional specifications that represent abstract implementations of each layer as a function of the lower layers. Formal proofs at one layer can be derived by using the mapping functions together with the formal specifications at appropriate layers. The relatively unsung Robinson–Levitt mapping analysis is actually quite far-reaching, and can be used directly to relate properties of a composed system to individual properties of its subsystems. As noted above with respect to correctness and completeness of interface specifications, this approach is of course limited by any incompleteness in the functional specifications and mapping functions. The Robinson–Levitt approach was part of the SRI Hierarchical Development Methodology (HDM) [46] used in the Provably Secure Operating System [19, 38,39] project in the 1970s. An extremely impressive new application of this approach in a modern setting has been developed by John Rushby and Rance DeLong [50], which uses the interpretation mechanism of SRI's current formal methods environment (PVS), and applies it to high-assurance separation kernels (which explicitly provide both isolation and controlled sharing) as well as virtual-machine architectures. An earlier informal application of explicit interlayer relationships is found in the analysis of the interlayer dependencies in the Honeywell/Secure Computing Corporation (SCC) LOgical Coprocessor Kernel (LOCK) [52]. (PSOS's type-enforcement was the precursor of several SCC systems, including the Sidewinder firewall.)
- *Operating system and programming language approaches*. Program modularity, recursive and nested procedure-call protocols, clean stack disciplines, and the absence of unintended residues can all greatly enhance composability. Virtualized multiprocessing and rigorously enforced virtual machine separation has considerable possibilities in enabling extremely efficient distributed processing by abstracting out many of the usual pitfalls, especially when distributed

across networked systems. There is an important role for sound programming languages that naturally enforce modular separation with abstraction and encapsulation, compilers that efficiently enforce the programming-language modularity and strong typing, systems that provide efficient interprocedure and interprocess control flow, and optimizing compilers that do not throw out the baby with the bathwater (e.g., by prematurely binding entities that need to remain separated until later, creating less easily analyzed object code, seriously impeding debugging, or compromising security separations provided by architectural encapsulations and programming languages). (However, well-implemented aggressive optimizers are less likely to violate security than programmers are.) As one example, SPARK (the SPADE Ada Kernel, based on the Southampton Program Analysis Development Environment) provides a language-based approach to improving security and safety. Correctness-preserving transformations that survive compilation and optimization are another approach with significant promise. In particular, optimizing compilers must be fairly farsighted not to compromise the integrity of source code in the context of its system execution, although careful modularity with abstraction and encapsulation can diminish some of those possible effects. An alternative approach to assuring the soundness of the optimization is the translation validation approach considered at NYU [55], in which a validation tool confirms that the object code produced by the optimizer is a correct translation of the source code.

- *Principled designs, implementations, and use*. As a summary of this section, the Saltzer–Schroeder principles and the further principles discussed above are potentially extremely beneficial to the attainment of security. Techniques particularly relevant to composability include abstraction, hierarchical layering, encapsulation, design diversity, composability, pervasive authentication, and access control, as well as administrative and operational controllability, pervasive accountability and recovery, separation of policy and mechanism, assignment of least privilege, separation of concerns, separation of roles, separation of duties, and separation of domains. The object-oriented paradigm also has some merit, especially strong typing. (However, the would-be inheritance of implementations without strict inheritance of specification subclasses tends to impede composability. Every subclass instance must meet the specifications of all its superclasses, or else all verifications of uses of the superclasses are unsupported.)

Several recent proceedings are worthy of consideration with regard to composable system architecture and software engineering [43,22,26,14].

7. A Crisis in Information System Security

Section 4 considers risks in trusting entities that might not actually be trustworthy. Nevertheless, flawed systems that can cause more security and reliability problems than they solve are in widespread use.

Untrustworthy mass-market software might be used so extensively for various reasons, even if the source code is proprietary and the vendor can arbitrarily download questionable software changes without user intervention. Sometimes this is a path of least resistance (with few perceived alternatives) or obliviousness. Or perhaps it has the appearance of saving money *in the short term*. In some cases it is mandated organizationally—ostensibly to simplify procurement, administration, and maintenance, or because of a desire to remain within the monolithic mainstream. Often security, reliability, and the risks of networking are considered less important, or there is a belief that the free market will provide a cure. But the simplest answer is probably "because it's there." However, irrespective of any reasons *why* people might be willing to use flawed software, in certain cases it might be wiser *not to use it* at all—especially where the risks are considerable.

In my fourth testimony (August 2001) in five years for committees of the US House of Representatives, I made the following statement—amplifying similar statements made in earlier years:

> "Although there have been advances in the research community on information security, trustworthiness, and dependability, the overall situation in practice appears to continually be getting worse, relative to the increasing threats and risks—for a variety of reasons. The information infrastructure is still fundamentally riddled with security vulnerabilities, affecting end-user systems, routers, servers, and communications; new software is typically flawed, and many old flaws still persist; worse yet, patches for residual flaws often introduce new vulnerabilities. There is much greater dependence on the Internet, for Governmental use as well as private and corporate use. Many more systems are being attached to the Internet all over the world, with ever increasing numbers of users—some of whom have decidedly ulterior motives. Because so many systems are so easily interconnectable, the opportunities for exploiting vulnerabilities and the ubiquity of the sources of threats are also increased. Furthermore, even supposedly standalone systems are often vulnerable. Consequently, the risks are increasing faster than the amelioration of those risks."

In many respects, the situation does not seem to be getting better. The continuing flurry of viruses, worms, and system crashes raises the level of disruption to users and institutions. The incessant flow of identified vulnerability reports and the further existence of flaws that are not widely known suggest serious problems. The continual needs for installing copious patches in mass-market software (and the iterative problems they sometimes cause) suggest that we are not converging. Putting the blame on inadequate system administration seems fatuous. Various exploitations of flaws (such as worms and viruses) are further examples of endemic problems in vulnerable systems that can be exploited. Unfortunately, too many people seem to be oblivious to the underlying security problems.

Suggestions that we need to raise the bar may be countered with the argument that past attacks have not really been serious, and we have had few pervasive disasters of information system security, so why should we worry? Unfortunately, Murphy's Law suggests that if it can happen, it eventually will. Also, the general overemphasis on short-term costs allows long-term concerns to be ignored. (See the next section.)

The Free Software/Open Source movements have been touted as possible alternatives to the inflexibilities of closed-source proprietary code. Indeed, GNU-Linux/BSD Unix variants are gaining considerable credibility, and are seemingly less susceptible to malware attacks. However, by itself, availability of source code is not a panacea, and sound software engineering is still essential. Even if an entire system has been subjected to extremely rigorous open evaluation and stringent operational controls, that may not be enough to ensure adequate behavior.

Many users and application developers have grown accustomed to flaky software, perhaps because they do not have to meet critical requirements (as in nuclear power control, power distribution, and flight and air-traffic control) and suffer no liability for disasters. Perhaps it is time to follow the adage of "Just Say No" to bad software, and to demand that software development be dramatically improved.

Many different approaches to software system development can be found in practice, such as object-oriented programming, aspect-oriented programming, agile software development, service-oriented architecture, design patterns, model-based design, event-driven architecture, clean-room development, extreme programming, formal methods, a long list of methodologies named after their progenitors, and so on. The discipline of these and other approaches can be very helpful, but trustworthiness demands much more than conventional software. Principled approaches are just one more step forward, and need to be coupled with sound development practices.

8. Optimistic Optimization

Many people (corporate executives, managers, developers, and so on) tend to ignore the long-term implications of decisions made for short-term gains, often based on overly optimistic or fallacious assumptions. In principle, much greater benefits can result from far-sighted vision based on realistic assumptions. For example, serious environmental effects (including global warming, water and air pollution, pesticide toxicity, and adverse genetic engineering) are largely ignored in pursuit of short-term profits. However, conservation and environmental protection appear much more relevant when considered in the context of long-term costs and benefits. Furthermore, governments are besieged by intense short-sighted lobbying by special interests. Insider financial manipulations have serious long-term economic effects. Research funding has been increasingly focusing on short-term returns, to the detriment of the future.

Computer system development is a particularly frustrating example. Most system developers are unable or unwilling to confront life-cycle issues up front and in the large, although it is clear that up-front investments can yield enormous benefits later in the life cycle. In particular, defining requirements carefully and wisely at the beginning of a development effort can greatly enhance the entire subsequent life cycle and reduce its costs. This process should ideally anticipate all essential requirements explicitly, including (for example) security, reliability, scalability, and relevant application-specific needs such as evolvability, maintainability, usability, interoperability, and enterprise survivability. Many such requirements are typically extremely difficult to add once system development is well underway. Furthermore, certain types of requirements tend to change; thus, system architectures and interfaces should be relatively flaw-free and inherently adaptable without introducing further flaws. Insisting on principled software engineering (such as modular abstraction, encapsulation, and type safety), sensible use of sound programming languages, and use of appropriate support tools can significantly reduce the frequency of software bugs. All these up-front investments can also reduce the subsequent costs of debugging, integration, system administration, and long-term evolution—if sensibly invoked.

Consideration of the value of up-front efforts is a decades-old concept. However, it is often widely ignored or done badly, for a variety of reasons—such as short-term profitability, rush to market, lack of commitment to quality, lack of liability concerns, ability to shift late life-cycle costs to customers, inadequate education, experience and training, and unwillingness to pursue other than seemingly easy answers.

Overly optimistic development plans that ignore these issues tend to win out over more realistic plans, but can lead to difficulties later on—for developers, system users, and even innocent bystanders. The past is littered with systems that did not work properly and people who did not perform according to the assumptions embedded in the development and operational life cycles. (An example is seen in the mad rush to low-integrity paperless electronic voting systems with essentially no operational accountability, discussed in Section 9.) The lessons of past failures are widely ignored. Instead, we have a *caveat emptor* culture, with developers and vendors disclaiming all warranties and liability.

Many would-be solutions result in part from short-sighted approaches. Firewalls, virus checkers, and spam filters all have some benefits, but also some problems.

Firewalls would be more effective if they were not required to pass all sorts of executable content, such as ActiveX and JavaScript—but many users want those features enabled. (To date, viruses and worms have been rather benign, considering the full potential of really malicious code.) However, active content and malware would be much less harmful in a well-architected environment that could constrain executable content in some sort of "sandbox" that has rigidly limited effects.

Spammers seem to adapt very rapidly to whatever defenses they encounter. For example, they can test their current offerings against existing anti-spam products and adapt accordingly. Furthermore, domestic legislation may simply drive spammers offshore, without reducing the pain.

Better incentives are needed for far-sighted optimization, in larger contexts and over longer periods of time, with realistic assumptions and appropriate architectural flexibility to adapt to changing requirements. Achieving this will require many changes in research and development agendas, software and system development cultures, educational programs, laws, economy, commitment, and perhaps most important—in obtaining well-documented success stories to show the way for others. Particularly in critical applications, if it is not worth doing sensibly, perhaps it is not worth doing at all. But as David Parnas has said, let's not just preach motherhood; let's teach people how to be good mothers.

9. An Example: Risks in Electronic Voting Systems

The challenge of ensuring election system integrity provides a paradigmatic example of the considerations of the previous sections. The election process is an end-to-end phenomenon whose trustworthiness typically depends on the integrity of every step in the process. Unfortunately, each of those steps represents various potential weak links that can be compromised in many ways, accidentally and intentionally, technologically or otherwise; each step must be safeguarded from the outset and auditable throughout the entire process.

Irregularities reported in the 2000 and 2004 US national elections span the entire process, concerning voter registration, disenfranchisement and harassment of legitimate voters, huge delays in certain precincts, unbalanced distribution of voting equipment, absence of provisional ballots (required by the Help America Vote Act), mishandling of absentee ballots, and problems in casting and counting ballots for e-voting as well as other modes of casting and counting votes. Some machines could not be booted. Some machines lost votes because of programming problems, or recorded more votes than voters. Some touch-screen machines altered the intended vote from one candidate to another. The integrity of the voting technologies themselves is limited by weak evaluation standards, secret evaluations that are paid for by the vendors, all-electronic systems that lack voter-verified audit trails and meaningful recountability, unaudited post-certification software changes, even runtime system or data alterations, and human error and misuse. (Gambling machines are held to much higher standards.) Other risks arise from partisan vendors and election officials. Furthermore, statistically significant divergences between exit polls and unaudited results created questions in certain states. All these concerns add to uncertainties about the integrity of the overall election processes.

With modern technology, the voting process could be more robust. Whether or not the potential weak links are mostly technological, the process can certainly be made significantly more trustworthy. Indeed, it seems to be better in many other countries than in the US; for example, Ireland, India, and the Netherlands seem to be taking integrity challenges seriously. As technologists, we should be helping to ensure that is the case—for example, by participating in the standards process or perhaps by aiding the cause of available source code and publicly accessible evaluations. However, the end-to-end nature of the problem includes many people whose accidental or intentional behavior can alter the integrity of the overall process, and thus creates many nontechnological risks.

With respect to computers used in elections, the principles outlined here would enable considerable improvements in trustworthiness if they were observed in practice. For example, architecturally minimizing the parts of the total system that must be trusted would by itself be a huge improvement, thereby reducing the extent of the weak links. The same is true of the principle of separating policy and mechanism.

The importance of understanding the idiosyncrasies of mechanisms and human interfaces, and indeed understanding the entire process, is illustrated by the 2000 Presidential election—with respect to hanging chad, dimpled chad, uncleaned chad slots, butterfly-ballot layouts, and the human procedures underlying voter registration and balloting. Clearly, the entire election process has vulnerabilities, including the technology and the surrounding administration. Looking into the future, a new educational problem will arise if preferential balloting becomes more widely adopted, whereby preferences for competing candidates are prioritized and the votes for the lowest-vote candidate are iteratively reallocated according to the specified priorities. This concept has many merits, although it certainly further complicates ballot layouts and voter awareness!

Alternative approaches have been proposed to existing voting systems (which have typically been lever machines, optically scanned paper, and paperless unauditable direct-recording computer systems). In approximate order of increasing conceptual complexity, these include (with examples of each) paper-based systems (Ben Adida [3], David Chaum [10], Ron Rivest [44]), cryptographic solutions (Andy

Neff [31], Josh Benaloh [5]), and voter-verified paper audit trails (VVPATs) (as an add-on for existing all-electronic systems, as proposed by Rebecca Mercuri [30]).

The VVPAT approach attempts to overcome the lack of integrity in existing direct-recording systems, but creates further complexity in the process. It is primarily a short-term fix to the current situation, in which proprietary software and proprietary evaluations against inherently incomplete voluntary standards provide relatively little system integrity. Cryptographic approaches require considerable care in design, analysis, implementation, and assurance, but also have the potential to avoid paper records—if the end-to-end systems could be made sufficiently trustworthy. On the

TABLE II
APPLICABILITY OF PRINCIPLES TO ELECTIONS

Principle	Computerization	Human Procedures
Economy of mechanism (+ sound architecture)	Simplistic mechanisms are dangerous. Complex systems need extensive analysis and predictable composability.	Operational simplicity is essential for poll workers. Perspicuous risk assessment is desirable throughout.
Fail-safe defaults	can simplify operation and improve trustworthiness.	can mitigate against insider misuse, fraud, and errors.
Complete mediation	can be useful in principled system architectures.	Weakness in depth requires end-to-end oversight.
Open design (+ openness generally)	Proprietary closed-source software and evaluations are inherently suspect.	Diverse oversight is essential throughout the entire process, especially over weak links.
Separation of privileges	can reduce insider misuse, human error, system failures.	can avoid centralized vested control throughout.
Least privilege (+ reduced needs for trust) (+ constrained dependency)	eschews root-privilege misuse, bootload subversion, trusting untrustworthiness. Avoid software built on subvertible underpinnings.	is important throughout the entire process, obviates allocation of excessive trust. Do not trust potentially untrustworthy people.
Least common mechanism	Beware of common flaws and common fault modes.	Separate roles may simplify assurance.
Psychological acceptability	Voter- and official-friendly systems can be helpful.	Ease of use and operation can help if it is not simplistic.
Work factors (+ objective risk analyses)	must encompass all systems, not just limited to strength of cryptography/authentication.	must encompass the entire process end-to-end, including developers and operators.
Compromise recording (+ pervasive monitoring)	Tamper-resistant audit trails are critical whenever results are suspect, and may help disincentivize fraud.	Manual procedures need oversight against compromise from outside/within/below, not just when suspicions arise.

other hand, the proposed paper-based systems have considerable conceptual simplicity and avoid many of the integrity problems of computer-based systems. However, these approaches address primarily only the vote recording and counting parts of the election process. End-to-end integrity must also include voter registration, voter and vote authentication, and postprocessing.

Table II tersely summarizes the potential relevance of principles (left column) for overall system architectures and development, for both computer-related systems (middle column) and operational procedures (right column) throughout the election process. The table represents a broadening of the Saltzer–Schroeder principles to address some additional aspects (suggested by what follows the *plus* sign in parentheses in the *principle* column). It thus generalizes the original principles somewhat to include related concepts discussed herein that reach farther than what was originally covered by Saltzer and Schroeder. It also reflects on the fact that these principles are relevant to trustworthiness overall—including (for example) many types of human errors and system failures that are not just limited to security issues. However, it does not remind the reader that this set of principles is only part of what is needed. Ultimately, expertise, experience, and good judgment are essential.

10. The Need for Risk Awareness

Around the world, our lives are increasingly dependent on technology. What should be the responsibilities of technologists regarding technological and nontechnological issues?

• Solving real-world problems often requires technological expertise as well as sufficient understanding of a range of economic, social, political, national, and international implications. Although it may be natural to want to decouple technology from the other issues, such problems typically cannot be solved by technology alone. They need to be considered in the broader context.

• Although experts in one area may not be qualified to evaluate detailed would-be solutions in other areas, their own experience may be sufficient to judge the conceptual merits of such solutions. For example, demonstrable practical impossibility or fundamental limitations of the concept, or the existence of serious conflicts of interest of the participants, or an obvious lack of personal and system-wide integrity are causes for concern.

• Ideally, we need more open, holistic, and interdisciplinary examinations of the underlying problems and their proposed solutions. (For example, see [37].)

Many concerns arise in important computer-related application areas, such as aviation, health care, defense, homeland security, law enforcement and intelligence—with similar conclusions. In each area, a relevant challenge is that of developing and operating end-to-end trustworthy environments capable of satisfying stringent requirements for human safety, reliability, system integrity, information security, and privacy, in which many technological and nontechnological issues must be addressed throughout the computer systems and operational practices. Overall, technologists need to provide adequate trustworthiness in our socially important information systems, by technological and other means. Research and development communities internationally have much to offer in achieving trustworthy computer-communication systems. However, they also have the responsibility of being aware of the other implications of the use of these systems.

A deeper knowledge of fundamental principles of computer technology and their implications will be increasingly essential in the future, for a wide spectrum of individuals and groups, each with its own particular needs. Our lives are becoming ever more dependent on understanding computer-related systems and the risks involved. Although this may sound like a meta-motherhood statement, wise implementation of motherhood is decidedly nontrivial—especially with regard to risks.

Computer scientists who are active in creating the groundwork for the future need to better understand system issues in the large, especially the practical limitations of theoretical approaches. System designers and developers need broader and deeper knowledge—including those people responsible for the human interfaces used in inherently riskful operational environments; interface design is often critical. Particularly in those systems that are not wisely conceived and implemented, operators and users of the resulting systems also need an understanding of certain fundamentals. Corporation executives need an understanding of various risks and countermeasures. In each case, knowledge must increase dramatically over time, to reflect rapid evolution. Fortunately, the fundamentals do not change as quickly as the widget of the day, which suggests that pervasive emphasis on education and ongoing training is needed with respect to the concepts of this chapter.

An alternative view suggests that many technologies can be largely hidden from view, and that people need not understand (or indeed, might prefer not to know) the inner workings. For example, David Parnas's early papers on abstraction, encapsulation, and information hiding are important in this regard. Although masking complexity is certainly possible in theory, in practice we have seen too many occasions (for examples, see the ACM Risks Forum archives) in which the occurrence of inadequately anticipated exceptions resulted in disasters. The complexities arising in handling exceptions apply ubiquitously, to defense, medical systems, transportation systems, personal finance, security, to our dependence on critical infrastructures that can fail—and to anticipating the effects of such exceptions in design, implementation, and operation.

Thus, computer-related education is vital for everyone. The meaning of the Latin word "educere" (to educate) is literally "to lead forth." However, in general, many people do not have an adequate perception of the risks and their potential implications. When, for example, the information media tell us that air travel is safer than automobile travel (on a passenger-mile basis, perhaps), the comparison may be less important than the concept that both could be significantly improved. When we are told that electronic commerce is secure and reliable, we need to recognize the cases in which it is not.

With considerable foresight and wisdom, Vint Cerf has repeatedly said that "The Internet is for Everyone." The Internet can provide a fertile medium for learning for anyone who wants to learn, but it also creates serious opportunities for the unchecked perpetuation of misinformation and counterproductive learning that will need to be unlearned.

In general, we learn what is most valuable to us from personal experience, not by being force-fed lowest-common-denominator details. In that spirit, it is important that education, training, and practical experiences provide motivations for true learning. For technologists, education needs to have a pervasive systems orientation that encompasses concepts of software and system engineering, security, and reliability, as well as stressing the importance of suitable human interfaces. For everyone else, there needs to be much better appreciation of the sociotechnical and economic implications—including the risks issues. Above all, a sense of vision of the bigger picture is perhaps what is most needed.

11. Risks of Misinformation

The problems of online misinformation are evidently worsening, because of the growth of the Internet and our ever increasing dependence on online systems. Information technology is a double-edged sword—perhaps even more so than many other technologies. In the hands of enlightened individuals, institutions, and governments, its use can be enormously beneficial. In other hands, it can be detrimental. Unfortunately, the dichotomy is often in the eye of the beholder, perhaps depending on one's objectives (e.g., personal financial gains, corporate profits, global economic well-being, politics, privacy, and environmental concerns).

Given a collection of online information, many people behave as if it is inherently authentic and accurate. This myth applies not only to websites, but also to many types of special-purpose databases, such as those found in law enforcement, motor vehicle departments, medicine, insurance, social security, credit information, and homeland security. We have seen many cases in which misinformation (e.g., false flight data, erroneous medical records, undeleted acquittals, or tampered files) has resulted in serious consequences. The same is true of imprecise information (e.g., resulting in false arrests, or affecting everyone with a particular name such as "David Nelson" who attempts to board an airplane).

Although an individual can occasionally observe that personal information about one's self is incorrect, more typically such erroneous information is hidden from the individual in question, possibly in diversely inaccurate versions. Overall, it is usually impossible for one to ensure that all such instances are correct—especially when mirrored in unknown sites all over the world. Furthermore, it is difficult to determine whether or not online information about anything else is authoritative. Worse yet, the volume of questionable information is growing at an extraordinary rate, and attempts to update substantive misinformation often have little effect—especially with the persistence of incorrect cached versions.

We increasingly rely on the Internet for many purposes, including education and enlightenment, irrespective of whether the sources are accurate. Oft-repeated overly simplistic sound-bite mantras seem to be popular. Furthermore, some people seem eager to waste time and energy that could be better spent elsewhere—or to drop out. There is a tendency for entrenched positions to remain fixed. Are we losing our ability to listen openly to other views and engage in constructive thought?

Another problem involves the inaccessibility of vital information. We seem to have evolved into a mentality of "If it is not on the Internet, it does not exist." Even though there are many more data bytes available today than ever before, search engines reportedly find only a small percentage of those pages, almost none of the database-driven dynamic Web pages, and very little of what is in most public libraries. Copyright restrictions and proprietary claims further limit what is available. For example, professional society digital libraries tend to be accessible only to those members who pay to subscribe. Furthermore, overzealous filtering blocks many authoritative sources of information. Are our education and information gathering suffering from a lowest-common-denominator process?

The propagation of misinformation has long been a problem in conventional print and broadcast media, but represents another problem that is exacerbated by the speed and bandwidth of the Internet. In general, widely held beliefs in supposedly valid information tend to take on lives of their own as urban myths; they tend to be trusted far beyond what is reasonable, even in the presence of well-based demonstrations of their invalidity.

In the face of such rampant misinformation, the truth can be difficult to accept, partly because it can be so difficult to ascertain, partly because it can seem so starkly inconsistent with popular misinformation, and partly because people want to believe in simple answers. Thus, we are revisiting classical problems that might now be considered as E-Epistemology, involving the nature and fundamentals of online knowledge—especially with reference to its limits and validity. However, there are some possible remedies, such as epistemic educational processes that teach us how to evaluate information objectively. For websites, this might entail examining who are the sponsors, what affiliations are implied, where the information comes from, whether multiple seemingly reinforcing items all stem from the same incorrect source, whether purported website security and privacy claims are actually justified, and so on.

12. Boon or Bane?

Predicting the long-term effects of computers is both difficult and easy: it is easy to predict the future (often mistakenly), but very difficult to be correct. Here are some suggested possible visions of the future.

• Computers play an increasing role in enabling and mediating communication between people. They have great potential for improving communication, but there is a real risk that they will simply overload us, keeping us from really communicating. We already receive far more information than we can process. A lot of it is noise. Will computers help us to communicate or will they interfere?

• Computers play an ever-increasing role in our efforts to educate our young. In some countries, educators want to have computers in every school, or even one on every desk. Computers can help in certain kinds of learning, but it takes time to learn the arcane set of conventions that govern their use. Even worse, many children become so immersed in the cartoon world created by computers that they accept it as real, losing interest in other things. Will computers really improve our education, or will children be consumed by them?

• Computers play an ever increasing role in our war-fighting. Most modern weapon systems depend on computers. Computers also play a central role in military planning and exercises. Perhaps computers will eventually do the fighting and protect human beings. We might even hope that wars would be fought with simulators, not weapons. On the other hand, computers in weapon systems might simply make us more efficient at killing each other and impoverishing ourselves. Will computers result in more slaughter or a safer world?

• Information processing can help to create and preserve a healthy environment. Computers can help to reduce the energy and resources we expend on such things as transportation and manufacturing, as well as improve the efficiency of buildings and engines. However, they also use energy, and their production and disposal create pollution. They seem to inspire increased consumption, creating what some ancient Chinese philosophers called "artificial desires." Will computers eventually improve our environment or make it less healthy?

• By providing us with computational power and good information, computers have the potential to help us think more effectively. On the other hand, bad information can mislead us, irrelevant information can distract us, and intellectual crutches can cripple our reasoning ability. We may find it easier to surf the Web than to think. Will computers ultimately enhance or reduce our ability to make good decisions?

• Throughout history, many people have tried to eliminate artificial and unneeded distinctions among people. We have begun to learn that everyone has much in common—men and women of all colors, races, and nationalities. Computers have the power to make borders irrelevant, to hide surface differences, and to help us overcome long-standing prejudices. However, they also facilitate the creation of isolated, antisocial groups that may spread hatred and false information. Will computers ultimately improve our understanding of other peoples or lead to more misunderstanding and hatred?

• Computers can help us to grow more food, build more houses, invent better medicines, and satisfy other basic human needs. They can also distract us from our real needs and make us hunger for more computers and more technology, which we then produce at the expense of more essential commodities. Will computers ultimately enrich us or leave us poorer?

• Computers can be used in potentially dangerous systems to make them safer. They can monitor motorists, nuclear plants, and aircraft. They can control medical devices and machinery. Because they do not fatigue and are usually vigilant, they can make our world safer. On the other hand, the software that controls these systems and the people involved may actually be untrustworthy. Bugs are not the exception; they are the norm. Will computers ultimately make us safer or increase our level of risk?

Much of the accumulated wisdom summarized in this chapter is not particularly new. But it is also not widely practiced. Many people are so busy advancing and applying technology that they do not look either back or forward. We should look back to recognize what we have learned about computer-related risks (e.g., [34]). We must look forward to anticipate the future effects of our efforts, including unanticipated combinations of seemingly harmless phenomena. Evidence over the past decades suggests we are not responding adequately to the challenges. Predilections for short-term optimization without regard for long-term costs abound. We must strive to make sure that we maximize the benefits and minimize the harm. Among other things, we must build stronger and more robust computer systems while remaining acutely aware of the risks associated with their use. Perhaps disciplined observance of the content of this chapter can help provide an impetus for the considerable culture change that is required for the development of trustworthy systems, networks, and enterprises in the future.

ACKNOWLEDGEMENTS

This chapter was written in part under National Science Foundation Grant Number 0524111. Sections 5 and 6 are based on a report sponsored by Douglas Maughan when he was the US Defense Advanced Research Projects Agency (DARPA) Program Manager for the Composable High-Assurance Trustworthy System (CHATS) program. Joshua Levy contributed some incisive comments on the final draft.

The author is very grateful to members of the ACM Committee on Computers and Public Policy, who over the years have been extraordinarily constructive in guiding the Inside Risks columns of the *Communications of the ACM*, which provided a springboard for some of the material here. In particular, Peter Denning, Jim Horning, David Parnas, Jerry Saltzer, and Lauren Weinstein have been especially helpful, as has Tom Lambert at *CACM*. In particular, Section 12 is based on a column that appeared in the March 2001 *CACM*, coauthored with David Parnas.

Section 3 was inspired by an article by Tim Batchelder, "An Anthropology of Air", *Townsend Letter for Doctors and Patients*, pp. 105–106, November 2005. "Because [air] is negative space, it is difficult to see the value in preserving it."

Lindsay Marshall at Newcastle has been extremely helpful in providing a nicely searchable archive facility for the ACM Risks Forum. (The official archives of all issues of the ACM Risks Forum since its inception in August 1985 can be found at http://www.risks.org, which indirects to http://catless.ncl.ac.uk/Risks/, and at ftp://www.sri.com/risks/.)

Will Tracz has consistently encouraged the author to contribute salient risks-related highlights to the *ACM SIGSOFT Software Engineering Notes*, ever since Will took over the editorship from Neumann's founding stint (1976–1993).

The citations given herein represent just the tip of an enormous literature iceberg. Additional references relevant to this chapter can easily be gleaned by searching the Web or by browsing my website.

REFERENCES

[1] Abadi M., Banerjee A., Heintze N., Riecke J.G., "A core calculus of dependency", in: *POPL '99, Proc. of the 26th SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Antonio, Texas*, January 20–22, 1999, pp. 147–160.

[2] Abadi M., Lamport L., "Composing specifications", in: de Bakker J.W., de Roever W.-P., Rozenberg G. (Eds.), *Stepwise Refinement of Distributed Systems: Models, Formalisms, Correctness*, *REX Workshop, Mook, The Netherlands, May–June 1989*, in: *Lecture Notes in Comput. Sci.*, vol. 230, Springer-Verlag, Berlin, 1989, pp. 1–41.

[3] Adida B., Neff C.A., "Ballot casting assurance", in: *Workshop on Electronic Voting Technology Workshop, Vancouver, BC, Canada, August 2006*, USENIX.

[4] An J.H., Dodis Y., Rabin T., "On the security of joint signature and encryption", in: *Advances in Cryptology*, *EUROCRYPT 2002, Amsterdam, The Netherlands*, in: *Lecture Notes in Comput. Sci.*, Springer-Verlag, Berlin, May 2002, pp. 83–107.
[5] Benaloh J., "Simple verifiable elections", in: *Workshop on Electronic Voting Technology Workshop, Vancouver, BC, Canada, August 2006*, USENIX.
[6] Biba K.J., "Integrity considerations for secure computer systems", Technical Report MTR 3153, The Mitre Corporation, Bedford, Massachusetts, June 1975. Also available from USAF Electronic Systems Division, Bedford, Massachusetts, as ESD-TR-76-372, April 1977.
[7] Bishop M., *Computer Security: Art and Science*, Addison–Wesley, Reading, MA, 2002.
[8] M. Bishop, *Introduction to Computer Security*, Addison–Wesley, Reading, MA, 2004.
[9] Chander A., Dean D., Mitchell J.C., "Reconstructing trust management", *J. Computer Security* **12** (1) (January 2004) 131–164.
[10] Chaum D., "Secret-ballot receipts and transparent integrity: Improving voter confidence & electronic voting at polling places", Technical report, March 2002, http://www.chaum.org.
[11] F.J. Corbató, "On building systems that will fail (1990 Turing Award Lecture, with a following interview by Karen Frenkel)", *Commun. ACM* **34** (9) (September 1991) 72–90.
[12] Corbató F.J., Saltzer J., Clingen C.T., "Multics: The first seven years", in: *Proc. of the Spring Joint Computer Conference, vol. 40, Montvale, New Jersey*, AFIPS Press, 1972.
[13] Datta A., Küsters R., Mitchell J.C., Ramanathan A., Shmatikov V., "Unifying equivalence-based definitions of protocol security", in: *Proc. of the ACM SIGPLAN and IFIP WG 1.7 Fourth Workshop on Issues in the Theory of Security, Oakland, California*, IEEE Computer Society, April 2004.
[14] de Boer F.S., et al. (Eds.), *Formal Methods for Components and Objects*, *4th International Symposium*, *Lecture Notes in Comput. Sci.*, vol. 4111, Springer-Verlag, Berlin, November 2005.
[15] de Roever W.-P., de Boer F., Hanneman U., Hooman J., Lakhnech Y., Poel M., Zwiers J., *Concurrency Verification: Introduction to Compositional and Noncompositional Methods*, *Cambridge Tracts Theoret. Comput. Sci.*, vol. 54, Cambridge University Press, New York, NY, 2001.
[16] Denning D.E., Neumann P.G., Parker D.B., "Social aspects of computer security", in: *Proc. of the 10th National Computer Security Conference*, September 1987.
[17] Dijkstra E.W., "The structure of the THE multiprogramming system", *Commun. ACM* **11** (5) (May 1968).
[18] Faughn A.W., "Interoperability: Is it achievable?", Technical report, Harvard University PIRP report, 2001.
[19] Feiertag R.J., Neumann P.G., "The foundations of a Provably Secure Operating System (PSOS)", in: *Proc. of the National Computer Conference*, AFIPS Press, 1979, pp. 329–334, http://www.csl.sri.com/neumann/psos.pdf.
[20] Gligor V.D., Gavrila S.I., "Application-oriented security policies and their composition", in: *Proc. of the 1998 Workshop on Security Paradigms, Cambridge, England*, 1998.
[21] Gligor V.D., Gavrila S.I., Ferraiolo D., "On the formal definition of separation-of-duty policies and their composition", in: *Proc. of the 1998 Symposium on Security and Privacy, Oakland, California*, IEEE Computer Society, May 1998.

[22] Gorton I., et al. (Eds.), *Component-Based Software Engineering, 9th International Symposium*, *Lecture Notes in Comput. Sci.*, vol. 4063, Springer-Verlag, Berlin, June/July 2006.
[23] Kopetz H., "Composability in the time-triggered architecture", in: *Proc. of the SAE World Congress*, *Detroit, Michigan*, SAE Press, 2000, pp. 1–8.
[24] Krafzig D., Banke K., Slama D., *Enterprise SOA Service-Oriented Architecture Best Practices*, Prentice Hall, Upper Saddle River, NJ, 2004.
[25] Lamport L., "A simple approach to specifying concurrent program systems", *Commun. ACM* **32** (1) (January 1989) 32–45.
[26] Löwe W., et al. (Eds.), *Software Composition, 5th International Workshop*, *Lecture Notes in Comput. Sci.*, vol. 4089, Springer-Verlag, Berlin, March 2006.
[27] Mantel H., "Preserving information flow properties under refinement", in: *Proc. of the 2001 Symposium on Security and Privacy*, *Oakland, California*, IEEE Computer Society, May 2001, pp. 78–91.
[28] Mantel H., "On the composition of secure systems", in: *Proc. of the 2002 Symposium on Security and Privacy*, *Oakland, California*, IEEE Computer Society, May 2002, pp. 88–101.
[29] McCullough D., "A hookup theorem for multilevel security", *IEEE Trans. Software Engineering* **16** (6) (June 1990).
[30] Mercuri R., "Electronic vote tabulation checks and balances", PhD thesis, Department of Computer Science, University of Pennsylvania, 2001. http://www.notablesoftware.com/evote.html.
[31] Neff C.A., "A verifiable secret shuffle and its application to e-voting", in: *Proc. of the ACM Conference on Computer and Communications Security, Philadelphia, Pennsylvania*, November 2001, pp. 116–125.
[32] Neumann P.G., "Illustrative risks to the public in the use of computer systems and related technology, index to RISKS cases", Technical report, Computer Science Laboratory, SRI International, Menlo Park, California. Updated regularly at http://www.csl.sri.com/neumann/illustrative.html also in .ps and .pdf form for printing in a denser format.
[33] Neumann P.G., "The role of motherhood in the pop art of system programming", in: *Proc. of the ACM Second Symposium on Operating Systems Principles*, *Princeton, New Jersey*, ACM, October 1969, pp. 13–18, http://www.multicians.org/pgn-motherhood.html.
[34] Neumann P.G., *Computer-Related Risks*, ACM Press, New York, and Addison–Wesley, Reading, MA, 1995.
[35] Neumann P.G., "Practical architectures for survivable systems and networks", Technical report, Final Report, Phase Two, Project 1688, SRI International, Menlo Park, California, June 2000. http://www.csl.sri.com/neumann/survivability.html.
[36] Neumann P.G., "Principled assuredly trustworthy composable architectures", Technical report, Computer Science Laboratory, SRI International, Menlo Park, California, December 2004. http://www.csl.sri.com/neumann/chats4.html, .pdf and .ps.
[37] Neumann P.G., "Holistic systems", *ACM Software Engineering Notes* **31** (6) (November 2006).
[38] Neumann P.G., Boyer R.S., Feiertag R.J., Levitt K.N., Robinson L. , "A provably secure operating system: The system, its applications, and proofs", Technical report, Computer

Science Laboratory, SRI International, Menlo Park, California, May 1980, 2nd edition, Report CSL-116.

[39] Neumann P.G., Feiertag R.J., "PSOS revisited", in: *Proc. of the 19th Annual Computer Security Applications Conference (ACSAC 2003), Classic Papers section*, *Las Vegas, Nevada*, IEEE Computer Society, December 2003, pp. 208–216, http://www.acsac.org/ and http://www.csl.sri.com/neumann/psos03.pdf.

[40] Organick E.I., *The Multics System: An Examination of Its Structure*, MIT Press, Cambridge, MA, 1972.

[41] Parnas D.L., "On the criteria to be used in decomposing systems into modules", *Commun. ACM* **15** (12) (December 1972).

[42] Petroski H., *To Engineer Is Human: The Role of Failure in Successful Design*, St. Martin's Press, New York, 1985.

[43] Reussner R.H., et al. (Eds.), *Architecting Systems with Trustworthy Components*, *International Seminar, Dagstuhl, Germany*, *Lecture Notes in Comput. Sci.*, vol. 3938, Springer-Verlag, Berlin, December 2004.

[44] Rivest R.L., "The threeballot voting system", Technical report, MIT, Cambridge, MA, October 2006.

[45] Robinson L., Levitt K.N., "Proof techniques for hierarchically structured programs", *Commun. ACM* **20** (4) (April 1977) 271–283.

[46] Robinson L., Levitt K.N., Silverberg B.A., *The HDM Handbook*, Computer Science Laboratory, SRI International, Menlo Park, California, June 1979 (three volumes).

[47] Rochlis J.A., Eichin M.W., "With microscope and tweezers: The Worm from MIT's perspective", *Commun. ACM* **32** (6) (June 1989) 689–698.

[48] Rosen E., "Vulnerabilities of network control protocols", *ACM SIGSOFT Software Engineering Notes* **6** (1) (January 1981) 6–8.

[49] Rushby J., "Modular certification", Technical report, Computer Science Laboratory, SRI International, Menlo Park, California, June 2002.

[50] Rushby J.M., DeLong R., "Compositional assurance for security: A MILS integration protection profile", Technical report, Computer Science Laboratory, SRI International, Menlo Park, California, December 2006.

[51] Saltzer J.H., Schroeder M.D., "The protection of information in computer systems", *Proc. IEEE* **63** (9) (September 1975) 1278–1308.

[52] Saydjari O.S., Beckman J.M., Leaman J.R., "LOCKing computers securely", in: *10th National Computer Security Conference, Baltimore, Maryland*, 21–24 September 1987, pp. 129–141;
Reprinted in Turn R. (Ed.), *Advances in Computer System Security*, vol. 3, Artech House, Dedham, Massachusetts, 1988.

[53] Smith M.A., "Portals: Toward an application framework for interoperability", *Commun. ACM* **47** (10) (October 2004) 93–97.

[54] Spafford E.H., "The Internet Worm: Crisis and aftermath", *Commun. ACM* **32** (6) (June 1989) 678–687.

[55] Zuck L., Pnueli A., Fang Y., Goldberg B., "VOC: A translation validator for optimizing compilers", in: *Electronic Notes in Theoretical Computer Science*, 2002.