The Emperor's Old Armor

Bob Blakley
International Business Machines Corporation

Abstract

The traditional model of computer security was formulated in the 1970's, when computers were expensive, solitary, heavy, and rare. It rests on three fundamental foundations: management of security *policy* describing the set of actions each user is entitled to perform, *integrity* of the physical system, its software, and especially its security-enforcing mechanisms, and *secrecy* of cryptographic keys and sensitive data.

The modern computing environment, with its rapidly accelerating complexity, connectivity, and miniaturization, is undermining all three of these foundations. Nevertheless, the newest "secure" computer systems continue to be built on them. This paper argues that the traditional model of computer security is no longer viable, and that new definitions of the security problem are needed before the industry can begin to work toward effective security in the new environment.

1 Introduction

The traditional computer security model is built around a *reference monitor*, supported by hardware protection mechanisms, which enforces administratively defined security policies. The reference monitor's software is assumed to be of high reliability and integrity. The reference monitor is supplemented by strong cryptography for those unfortunate occasions when our data must venture outside the cozy confines of its safe haven.

This model's analogies are mostly military: the image is that of an *information fortress*, with walls, guards, interior compartments, and a defending army. When you approach the information fortress's outer wall ("security perimeter"), you present your "password" to the guardian of the gate. The fortress's defensive garrison ("access control" facilities) protect your "confidential data" until you want to send it out of the "security perimeter", perhaps through a "firewall", at which point you use a code (but only in your home country – because cryptography is a "munition"!) The system's strong walls and trustworthy gate guards ("integrity features of the Trusted Computing Base") protect it against the introduction of "Trojan Horses" and "logic bombs".

The information fortress model was designed for (and in) a world in which computers were expensive, solitary, heavy, and rare. But that world is long gone. As frequent press reports indicate (the cover of the February 19, 1996 volume of *InformationWeek*, for example, proclaimed "Internet Security: Your Worst Nightmare") information fortresses are not protecting today's information much more effectively than Europe's magnificent physical fortresses are protecting today's national borders.

Given this state of affairs, it seems natural to fear that the security community's current efforts will fail. After all, we all want our systems to be secure (right?) But perhaps we should worry instead that we might *succeed*.

If success is defined in terms of attributes suitable to secure, but expensive, solitary, heavy, and rare computers, it seems probable that - in the increasingly unlikely event of success - we will buy security at the expense of important advantages of cheap, connected, miniature, and ubiquitous computers. In particular, the current security worldview, if vigorously enforced, seems certain to:

- limit connectivity and connected functionality
- Drive software costs sharply upward through substantial additional assurance expense
- Overwhelm administrators with policy, key, and audit log management

Do we really face a choice between useful computing and safe computing? The information fortress paradigm seems to hold out little hope of providing a secure heterogeneous, open, distributed, object-based, (fill in your favorite computing buzzword here), worldwide network – and today we have no alternative paradigm.

2 Symptoms

Thomas Kuhn's book *The Structure of Scientific Revolutions* [Kuh70] provides a useful guidebook for disciplines whose fundamental paradigms are in trouble. In Kuhn's view, the principal symptom of a scientific crisis is the persistent failure of the puzzles of normal science to come out as they should:

> "The state of Ptolemaic astronomy was a scandal before Copernicus' announcement. Galileo's contributions to the study of motion depended closely upon difficulties discovered in Aristotle's theory by scholastic critics. Newton's new theory of light and color originated in the discovery that none of the existing pre-paradigm theories would account for the length of the spectrum, and the wave theory that replaced Newton's was announced in the midst of growing concern about anomalies in the relation of diffraction and polarization effects to Newtonian theory. Thermodynamics was born from the collision of two existing nineteenth-century physical theories, and quantum mechanics from a variety of difficulties surrounding black-body radiation, specific heats, and the photoelectric effect. Furthermore, in all these cases except that of Newton the awareness of the anomaly had lasted so long and penetrated so deep that one can appropriately describe the fields affected by it as in a state of growing crisis. Because it demands large-scale paradigm destruction and major shifts in the problems and techniques of normal science, the emergence of new theories is generally preceded by a period of pronounced professional insecurity. As one might expect, that insecurity is generated by the persistent failure of the puzzles of normal science to come out as they should. Failure of existing rules is the prelude to a search for new ones."

Computer security should perhaps not yet be considered a science, but Kuhn's framework is still useful as a guide to discussing its present state of affairs.

That state is dismal. The same exposures keep recurring; we make no practically useful progress on the hard problems of integrity, assurance, policy, and interoperability; and we are less and less able to adapt the fortress model to new technologies as they arise. Williams, Shafer, and Landoll [WSL95] declared the crisis:

> "There is a crisis emerging in information technology. Reliance on this technology is increasing much more quickly than our ability to deal with the also increasing threats to information security."

Computers are rapidly getting smaller, cheaper, and more richly connected. More and more data resides on machines incapable of meaningful physical security (for example, laptop computers and personal digital assistants) and designed – by economic necessity – with no strong logical security. Even the relatively few remaining information fortresses have thrown open their gates to Ethernet, ISDN, and fiber connections. At the other end of those connections lies the worldwide Internet, on which, as Steve Bellovin has observed, [Bel92] "There Be Dragons".

The Internet exists essentially independent of national and international authorities, has no effective inherent security mechanisms, and houses a terrifying number of attackers of various stripes. Bellovin's monitoring tools detected more than 450 attacks against AT&T's network gateway in just 2 months; these attacks originated from more than 90 different sites. Since Bellovin's paper was published, CERT has observed a steady and rapid growth in the number of Internet security incidents; statistics published in the March 1996 issue of *IEEE Computer* indicate an approximate yearly doubling in the number of incidents reported to CERT [CP96]. This information is reported with the caveat that "each incident may involve one site or hundreds (or even thousands) of sites, and some incidents have ongoing activity for long periods of time...." An even more alarming set of statistics appears in the same *IEEE Computer* column:

> "In a recent series of tests by cooperating organizations, using hacker tools freely available on the Internet and penetrating each other's sites, the results were striking:
>
> - 88 percent of the attempted penetrations were successful
> - 96 percent of all system penetrations went undetected
> - In 95 percent of the instances where penetration was detected, nothing was done"

The advice given to users trying to deal with these issues is not always very reassuring. A recent *PC-Week* article, after describing security defects in Netscape Navigator's Java and JavaScript implementations and in Microsoft Internet Information Server, concluded as follows [Sul96]:

> "None of these problems makes it impossible to maintain a reasonable level of security on an Internet-connected network. However, the nature of the Internet demands at the very least daily scans of Usenet posts, E-mail lists, and Web sites for information on security breaches and the availability of patches.[1]"

Technologies more disruptive than the Internet loom on the horizon. Examples include:

1. Object-orientation blurs the distinction between data and code, robbing us of one of our most powerful integrity tools (hardware-enforced memory protection). At the same time object orientation encourages us to reuse code written by others – in some cases without benefit of access to the source text of the code we reuse. Ken Thompson's nightmare [Tho84] is now perfected:

   > "You can't trust code that you did not totally create yourself. (Especially code from companies that employ people like me.) No amount of source-level verification or scrutiny will protect you from using untrusted code."

   In an object-oriented world we seem likely to ensure that "no amount of source-level verification" will even be possible!

2. Intelligent Agent architectures invite us to execute other peoples' code on our systems and to write our own code and send it out to make its way in the world without benefit of our oversight. These agents are not distinguishable from programs we used to call viruses.

The software industry is in general not keeping up with the escalating threat; although the design of Java shows evidence of some commendable attention to security, most modern software is designed without any thought given to security up-front. The Internet, OMG CORBA, the Worldwide Web, and most Personal Computer operating systems are examples of major components of the worldwide software infrastructure into which security is currently being retrofitted.

The effectiveness of the security we are building is open to question. Two "hot" security technologies (Public-Key Cryptography and Firewalls) are good examples:

2.1 Public-Key Cryptography

Commonly claimed advantages of public-key technology include:

1. Offline servers

   The idea here is that since the only long-term cryptographic keys which must be exchanged in a public-key system are public, we might be able to get away with keeping the servers which generate the public/private key pairs and hand them out to users offline. Furthermore, since trustworthy public keys are available which can be used to protect exchanged session keys, we should also be able to make do without trusted servers in authentication dialogs.

   This all works fine as long as we don't care about auditing (see the next paragraph) or timeliness (see the next bulleted item).

   If we want to audit logons, then some trusted online server must be involved in the logon process. If we want to audit client-server authentication dialogs, it is convenient to have a trusted,

[1] There is a strong temptation to mimic Dave Barry here and add "I am not making this up!" The cited column's recommended solutions to the two Netscape flaws are classics (emphasis added): **Javascript** "Stop using Navigator 2.0 and upgrade to Version 2.01. *Watch for suspicious activity when connecting to unfamiliar Web sites.*" **Java** "Apply Netscape patch. Disable Java support until the patch is installed. *When possible, use the Java Development Kit only on isolated machines.*"

online, third-party server involved in the authentication dialog, because the absence of such an entity requires every server in the system to be able (and trusted) to audit client authentications.

2. Simple key distribution

   The simple version of public-key key distribution is "client chooses session key, encrypts under target's public key, and sends to target". There are subtle problems with this:

   - The client must authenticate that the target's public key is legitimate. Certificates and Certification Authorities were invented to solve this problem.
   - The client must confirm that the Certification Authority hasn't changed its mind about the legitimacy of the target's public key. Certificate Revocation Lists were invented to solve this problem, but not everyone believes CRLs are a good idea. Rivest and Lampson [RL96] are willing to give up the presumed advantages of offline servers, in part to avoid having to use CRLs:

     > "We assume that principals who issue certificates can provide on-line Internet service, or can arrange to provide such via a designated server.... Having such on-line capability permits considerable simplifications – for example, we eliminate 'certificate revocation lists' in favor of on-line 'reconfirmation'."

   Whether a public-key system which requires management of a Certification Authority, Certificates, and CRLs is simpler than a secret-key system which requires management of a Security Server is open to question.

3. Simple administration

   Here the claim is that the lack of on-line repositories full of secrets and the potential for ad-hoc key distribution make public-key systems significantly easier to understand and manage than secret-key systems. It is not clear that experience supports this claim; Don Davis [Dav96] observes that some of the apparent simplicity of public-key system administration is accomplished by sleight of hand: the complexity is still there, but it has been transferred from the system operators to the end users:

   > "it is not widely appreciated that these advantages rely excessively on end-users' security discipline. In fact, the reason public-key security doesn't need a trusted key-management infrastructure is that the burden of key-management falls to public-key clients. With public-key cryptography, clients must constantly be careful to validate rigorously every public key they use, and they must husband the secrecy of their long-lived private keys. It turns out that these tasks are harder than they seem.
   >
   > "End-users are unwilling or unable to manage keys diligently. Perhaps surprisingly, it's impossible to automate asymmetric key management completely; certain security details remain for human intervention, such as Root-key validation, passphrase choice, and clients' physical security. Even where automation is possible, as with revocation-list checks, scaling problems and performance costs make short-cuts likely. If users or developers skip these details, there is no way to detect their omission or to audit the consequences. I have coined the term compliance defect for this situation: a rule of operation that is difficult to follow and that cannot be enforced. Compliance defects undermine the security of public-key cryptography. When users fail to manage their private keys securely, or when they fail to validate each others' public keys rigorously, then authenticity and privacy guarantees weaken, and everyone's security deteriorates."

2.2 Firewalls

Byte magazine [Ker96] recently reported on evidence that firewalls may not be a very effective means of preventing Internet break-ins:

> "The notion of a firewall as an impregnable defense against intruders is going up in smoke. Firewalls were in place in two highly publicized security breaches.... These aren't isolated cases. According to the Computer Security Institute, 30 percent of the Internet sites that reported breaches in their security had a firewall in operation."

There seems to be no documented evidence that firewalls prevent or deter attacks. Until careful studies are done, we might be wise to keep our expectations modest; many firewall deployments are undermined by one or more fallacies:

1. *We've got the place surrounded*

   Firewalls can be effective only if all traffic must go through them to get from the outside of the protected network to the inside and vice versa. But networks without unprotected connections (*e.g.* modems) between the inside and the outside are rare, and the probability that a network has no such illicit connections decreases as its size increases.

2. *Nobody in here but us chickens*

   Firewalls cannot protect a network against bad guys who are already inside. As the size of a network increases, so does the chance that it contains bad-guy insiders.

3. *Sticks and stones can break my bones, but words can never hurt me*

   Firewalls are best at protecting systems against connections to "bad systems". They are much less effective at screening out bad data from "good systems". The bad data problem arises as soon as an organization allows insiders to connect to and download information from the World Wide Web. The recent emergence of word-processor macro viruses shows that bad data can be destructive. Some amount of screening of data is possible using application-level *proxies*, but these proxies are themselves problematic [Ker96]:

   > "Proxy servers present management headaches, according to Kevin Kitagawa, Internet security product line manager for Sun's Internet Commerce Group. 'Proxy servers are wonderful for most common Internet protocols or services,' he says. 'The problem is, for every new protocol or service that comes out, you have to add another application to the proxy server, like screening audio and so on.' The proxy server cannot handle protocols that lack a specific proxy for them. Proxy architectures can also degrade performance and transparency."

   When data is actually code, as with Java applets, the problem gets worse. The Java "sandbox" approach to dealing with this problem is, as [DFW96] and much related work shows, hard for implementors to get right. It is also hard for users to live with, since it requires that they never use internet-resident applets with any data they consider important enough to protect.

3 Skeletons in the locked closet

The Fundamental Principles of the information fortress model are these:

1. *Policy*, enforced by the system, protects resources from unauthorized manipulation
2. *Integrity* of the physical system and its code guarantees that policy is enforced
3. *Secrecy* of crypto keys and sensitive data underlies policy enforcement mechanisms

The security community's dirty little secret is that all three of these principles, which form the pillars of modern software security architecture, rest on infirm foundations:

3.1 Policy

Policy has two, related, fundamental problems: complexity and scale.

Policy scales poorly in every dimension. The complexity of policy which must be stated in order to manage a system securely increases if any of the following increase:

- number of subjects
- number of job functions
- number of objects
- number of operations
- number of semantic classes of data (sensitivity labels, categories, etc...)

The last two of these are the worst. Administering access control is tolerable in a system whose only operations are `create file, read file, write file, delete file`. In a system with relational operators, the access control problem is already much harder -- to do a good job of administering inference control policies, an administrator must have enough detailed knowledge about the structure of a database and the information it contains to know which sequences of queries are inadmissible. As the number and semantic complexity of operations increases, the administrator's job quickly spirals out of intellectual control.

3.2 System integrity and the reference monitor

System integrity assures that the security policy of a system cannot be bypassed. The US National Computer Security Center defines *integrity* as follows [Nat88]:

> "sound, unimpaired, or perfect condition"

This sets the bar pretty high, even by computer security standards. But perfection really is the standard, because any hole in the wall of the fortress will let the enemy in. [Nat91a] describes which pieces of the system must be perfect:

> "Systems that are used to process or handle classified or other sensitive information must be designed to guarantee correct and accurate interpretation of the security policies and must not distort the intent of those policies. Assurance must be provided that correct implementation and operation of the policy exists throughout the system's life-cycle. Application subsystems used to process or handle classified or other sensitive information must be designed, implemented, controlled, and operated in a manner which provides assurance that the goals of both application-specific security policies and system-wide security policies are met without circumvention."

It seems quite unlikely that the software industry will come very close to this standard, for a variety of reasons:

3.2.1 System integrity is hard

Essentially, in order to get a system with excellent system integrity, you must insure that it is designed and built by geniuses. Geniuses are in short supply, as the 1991 US National Research Council report *Computers at Risk* [Nat91b] observes:

> "There is a shortage of well-qualified people to work on production-quality software. There is a more serious shortage of those qualified to build critical software, and a dramatic shortage of people qualified to build secure software...Setting requirements for, specifying, and building critical software require specialized knowledge not possessed by typical software engineers. Over the years other engineering disciplines have developed specialized techniques – hazard analysis – for analyzing critical artifacts. Such techniques are not covered in most software engineering curricula, nor are they covered by most on-the-job training. Furthermore, working on critical software requires specialized knowledge of what can go wrong in the application domain. Working on secure software requires yet more skills. Most notably, one must be trained to understand the potential for attack, for software in general and for the specific application domain in particular."

An illustration of the challenge integrity poses to the average working programmer appeared recently in *IEEE Transactions on Software Engineering*. Presumably a prerequisite to demonstrating that a system always does what it is supposed to do, is specifying what it is supposed to do. Kate Finney studied a population of Computer Science students to find out how well they were able to read specifications. The results were not encouraging [Fin96]:

> "The experiment that was carried out involved 62 students, undergraduate and postgraduate, in reading a very small portion (less than 20 lines) of a specification in Z[2]. All were attending computing courses and most had been through a basic grounding in discrete mathematics in addition to separate tuition in the use of Z... Each student was asked three questions to test their ability to read and understand the specification.
>
> "in general the students found it difficult to understand any of the very simple Z specifications.
>
> "The point to note is that 19 students, nearly a third of the group, could not answer a single question and found the specification incomprehensible."

A correct specification is of course only the beginning of producing a system with good integrity. Even if ordinary working programmers could be trained to understand and use precise specifications, and even if specification tools adequate to the description of secure systems were available,[3] there would still be a long way to go.

[2] Z is a formal language for specifying the behavior of software systems; its formal foundations (set theory and classical first-order predicate logic) are "plain vanilla" by computer science standards. A number of good introductions exist, including for example [Dil94]. The standard reference is [Spi92].

[3] There is some progress; some recently designed protocols are provably secure in a precisely defined sense under explicitly stated assumptions. See *e.g.* [BR94] and [BR95].

A correct specification would have to be transformed into correct code. Systematic approaches to this exist, but they are seldom used. Notable examples include the specification-refinement methodology of Dijkstra, Gries, and others [Gri81], and the Cleanroom methodology [Dye92] (there are many others). Both methods have long pedigrees, but neither has made significant inroads into commercial programming practice. In the absence of reliable software engineering methodologies, software error rates remain depressingly high.

Correct code would need to be compiled correctly; for the dangers here *cf.* Thompson's Turing Award lecture, cited above. Finally, correctly compiled code would have to be executed on correctly functioning hardware; Intel's recent floating-point division bug reminds us that this cannot always be taken for granted.

3.2.2 System integrity is very expensive

Assurance (the process of demonstrating that a system has good integrity) is particularly costly. Most of today's software is designed for a mass market, in which many customers either do not have or do not acknowledge a serious security requirement. Cost-justifying the effort required to assure system integrity for mass-market software appears to be infeasible.[4] Even where assurance has been done diligently, the sheer size of modern software artifacts raises doubts about its effectiveness. Microsoft unquestionably put a lot of effort into assurance of the Windows NT security architecture and implementation; NT is an NCSC C2 evaluated system – but it is also more than 5 *million* lines of code [McC96]. What does the C2 evaluation imply, in practical terms, about the likely number of security-critical errors remaining in those 5+ million lines?[5]

[4]The temptation to condemn vendors as lazy or irresponsible for underspending on assurance is powerful; however, the security community has not made a very good case that the market will support the cost of assurance – or even that money for assurance is well spent. `sendmail` is a fairly small program. A quick check of the CERT `ftp` archives (`ftp://info.cert.org/pub/cert_advisories`) reveals that sendmail is the subject of at least six CERT advisories – three in 1995 alone – of which the first was issued in January, 1990. Does anyone believe, after all this attention and after application of the many patches referenced in CERT's advisories, that `sendmail` is now secure?

[5]Microsoft-bashers considering indulging in a feeling of superiority here should reconsider: other modern, popular, commercial operating systems are also huge and complicated; any of them could have been chosen for this example – except that many of them aren't even C2 evaluated!

3.2.3 System Integrity requires tradeoffs

System integrity is often bought at the expense of other desirable system quality attributes. Modularization, with strong inter-module boundaries, is a technique often used to improve system integrity. Unfortunately, inter-module boundary traversals tend to be expensive on general-purpose hardware; therefore, systems with strong integrity are often slow.

3.3 Secrecy

The fortress model depends heavily on secrecy. In networked environments, the use of cryptography to guarantee confidentiality and integrity of data has led to an explosion of cryptographic keys, with the result that key generation, management, and distribution is the central problem of distributed system security design. It is by no means a solved problem; for example, faults in key handling code lie at the heart of recently publicized flaws in SSL and Kerberos (version 4).

Even stand-alone systems depend critically on secrecy, for authentication; attacks on users' secret passwords are a long-standing and persistent problem.

The security community has recognized the problems associated with secrecy and has shrunk the secrecy perimeter to exclude everything except cryptographic keys; this has been formalized as Kerchoff's principle: *security is in the keys*, which is intended to mean that if the keys remain confidential, the system is secure. But decades of experience with the problems of passwords and crypto key management suggest that a more accurate formulation might be *insecurity is in the keys.*

The simple problem with secrets is that people are not good at keeping them. Though simple, this problem is fundamental: many attacks on "secure" systems succeed simply because they are performed by insiders who misuse their legitimate access to authentication secrets, cryptographic keys, or confidential information. Outsiders also penetrate "secure" systems by exploiting secrecy failures; this is called *social engineering*, and it too is a serious problem. Ira Winkler writes [Win96]

> "even the best security mechanisms can be bypassed through Social Engineering. Social Engineering uses very low cost and low technology means to overcome impediments posed by information security measures."

He goes on to discuss the details of a particular social engineering attack. He concludes

"The attack yielded sensitive company information and numerous user passwords, from many areas within the company, giving the attackers the ability to cripple the company despite extremely good technical information security measures. The results would have been similar with almost any other company.

...

"Even the best technical mechanisms could not have prevented the attack."

As Ruth Nelson observes, there are also complicated problems [Nel94] with secrecy:

"Another interesting question is what piece of information contains or communicates a secret. The relationship between information and secrecy is complicated, as the following examples suggest:

1. If we cut a secret in half, is it still a secret? Suppose that a secret recipe calls for 6 cups of sugar. Is 6 the secret? Cups? Sugar? ...
2. If we move the secret out of context, is it still a secret? In the example above, it is clear that "6" is not a secret in general. In the context of the secret recipe, it may be.
3. If we collect enough non-secret information and process it correctly, we may have a secret....
4. Some observers may already know something about a secret or have a good guess at it; in that case, a large secret can be communicated with very little information flow..."

Public-key cryptography tries to address some of the problems caused by excessive reliance on secrecy. But is it the devil's bargain? The false hope that public keys could be printed in the newspaper and forgotten has faded like a mirage, and in its place we have hundreds of pages of legalese outlining under what circumstances key pairs (and corresponding certificates) of various grades may be issued, what it's safe to use them for, and what obligations they impose upon their issuers and owners (for an example, see [Ver96]). Indeed, public-key key pairs seem more and more like nuclear waste; their private halves are hazardous if anyone comes in contact with them,[6] and their public halves need to be kept around in elaborately secure crypts for longer than their owners are likely to live.[7]

This metaphor is in deadly earnest. Before we as a society create huge numbers of these key pairs, we had better understand the management and disposal issues we will face down the road. Public-key certificates are essentially reified trust, just as cash is reified value. Mankind has no experience managing stockpiles of trust – especially stockpiles of misplaced trust. Ghosts of broken promises, echoes of failed relationships, the assurances of frauds – all these will be in the box, waiting for some modern Pandora to discover a private key, erase a CRL entry, or break an algorithm[8] and let them out.

4 Manifesto

The central proposition of this paper is:

No viable secure system design can be based on the principles of Policy, Integrity, and Secrecy, because in the modern world Integrity and Secrecy are not achievable and Policy is not manageable.

This is why computer security is starting to fail - and why it will continue to fail until it is re-built on new foundations.

5 Why us; Why Now?

Three trends have precipitated the crisis by undermining the foundations of the fortress:

1. *Miniaturization* makes physical security infeasible, and makes assurance expenses burdensome because they work against the economies of scale which cost-justify miniaturization. Miniaturization also makes theft easy. Finally, very small devices give the impression of being not very valuable; sociologically this works against securing them.

2. *Connectivity* exposes systems to a much broader and more diverse population of users than

[6]Because anyone other than their owners can use them to commit fraud.

[7]Because they may be needed to verify signatures on documents with very long lifespans.

[8]While we're on the subject, when we create key pairs with 10-year lifespans, or use a private key to sign a 30-year mortgage, will we think about what percentage of the planet's wealth we're willing to bet on the proposition that our cryptographers are smarter than everyone alive today, *and* everyone waiting to be born?

ever before. The software which provides connectivity is itself an exposure in today's environment; it typically enforces few or no policies and is complex and poorly assured. Finally, connectivity compounds the problems of complexity, scale and policy composition in heterogeneous systems.

3. *The Mass Market* imposes severe economic constraints on software development. Mass-market software must simultaneously compete on price and get quickly to market. This has the effect of putting pressure simultaneously on schedules and costs. The well-known schedule-cost-quality triangle guarantees that design and assurance quality will suffer as a result.

6 Reactions to the Unfolding Crisis

Kuhn's description of the development of a field in crisis describes us well:

> "When ... an anomaly comes to seem more than just another puzzle of normal science, the transition to crisis and to extraordinary science has begun. The anomaly itself now comes to be more generally recognized as such by the profession. More and more attention is devoted to it by more and more of the field's most eminent men. If it still continues to resist, as it usually does not, many of them may come to view its resolution as the subject matter of their discipline. For them the field will no longer look quite the same as it had earlier.... An ... important source of change is the divergent nature of the numerous partial solutions that concerted attention to the problem has made available. The early attacks upon the resistant problem will have followed the paradigm rules quite closely. But with continuing resistance, more and more of the attacks upon it will have involved some minor or not so minor articulation of the paradigm, no two of them quite alike, each partially successful, but none sufficiently so to be accepted as paradigm by the group. Through this proliferation of divergent articulations (more and more frequently they will come to be described as ad hoc adjustments), the rules of normal science become increasingly blurred. Though there still is a paradigm, few practitioners prove to be entirely agreed about what it is. Even formerly standard solutions of solved problems are called in question.
>
> When acute, this situation is sometimes recognized by the scientists involved.... Einstein... wrote... 'It was as if the ground had been pulled out from under one, with no firm foundation to be seen anywhere, upon which one could have built.' And Wolfgang Pauli, in the months before Heisenberg's paper on matrix mechanics pointed the way to a new quantum theory, wrote to a friend, 'At the moment physics is again terribly confused. In any case, it is too difficult for me, and I wish I had been a movie comedian or something of the sort and had never heard of physics.'"

We are undeniably experiencing "a proliferation of divergent articulations" of the dominant paradigm. Articulations proliferate in all areas: standards (IEEE 802.10, GSS-API, SSL, X.509, Kerberos) and standards organizations (ISO, ECMA, IETF, X/Open), industry consortia (OMG, OSF, OURS, I4), technologies (RSA, DSS), product approaches (virus scanners, firewalls, smartcards, authentication servers, SSO products, hardware copy-protection devices). This all comes to a head in the problem of integration of secure systems – a direct example of "the divergent nature of the numerous partial solutions" being "an important source of change".

Pauli's lament is also familiar; more and more security practitioners seem to be asking themselves, and asking one another, whether the computer security problem as we have currently stated it, is not simply too hard to be solved. Bace and Schaefer [BS95], for example, put it this way:

> "
>
> - Ain't gonna be no secure-enough operating systems to meet the needs of every[wo]man (graphics, cheap, fast, modern, object-oriented, windows'n'MIDI, etc.)
> - Ain't gonna be no immediate cure for usurpation of privilege by borrowed software or downloaded programs
> - Ain't gonna be no immediate cures for violations of license agreements or use of pirated software and illicit cloning of software
> - Ain't gonna be no cure for incorrect software or hardware and consequences of running it"

The New Security Paradigms Workshop is a kind of formal recognition by practitioners of the art that the Old Security Paradigm is nearing (or perhaps beyond) the end of its useful life.

7 What is the Way Forward?

Kuhn observes that resolution of a paradigm crisis requires the existence of a workable alternative paradigm:

> "Let us then assume that crises are a necessary precondition for the emergence of novel theories and ask next how scientists respond to their existence. Part of the answer, as obvious as it is important, can be discovered by noting first what scientists never do when confronted by even severe and prolonged anomalies. Though they may begin to lose faith and then to consider alternatives, they do not renounce the paradigm that has led them into crisis. They do not, that is, treat anomalies as counterinstances, though in the vocabulary of philosophy of science that is what they are.... Once it has achieved the status of paradigm, a scientific theory is declared invalid only if an alternate candidate is available to take its place.... The decision to reject one paradigm is always simultaneously the decision to accept another, and the judgment leading to that decision involves the comparison of both paradigms with nature and with each other."

If we accept this judgment, it's clear that the first step towards resolution of the crisis is to start building candidates for the position of alternative paradigm.

8 New Fundamentals

How does one build a new paradigm? Kuhn comments on this also:

> "The transition from a paradigm in crisis to a new one from which a new tradition of normal science can emerge is far from a cumulative process, one achieved by an articulation or extension of the old paradigm. Rather it is a reconstruction of the field from new fundamentals, a reconstruction that changes some of the field's most elementary theoretical generalizations as well as many of its paradigm methods and applications. During the transition period there will be a large but never complete overlap between the problems that can be solved by the old and by the new paradigm. But there will also be a decisive difference in the modes of solution. When the transition is complete, the profession will have changed its view of the field, its methods, and its goals. One perceptive historian, viewing a classic case of a science's reorientation by paradigm change, recently described it as 'picking up the other end of the stick'"

Where do we look for inspiration? Kuhn again:

> "It is, I think, particularly in periods of acknowledged crisis that scientists have turned to philosophical analysis as a device for unlocking the riddles of their field."

If the information fortress arises from an essentially military view of the world, it seems natural to look first to military philosophers.

Clausewitz [von93] warned against perfection as a practical standard, and explained why activities which seem simple in theory, or on a small scale, don't work in real-world conflicts:

> "If one has never personally experienced war, one cannot understand in what the difficulties constantly mentioned really consist, nor why a commander should need any brilliance and exceptional ability. Everything looks simple; the knowledge required does not look remarkable, the strategic options are so obvious that by comparison the simplest problem of higher mathematics has an impressive scientific dignity....
>
> Everything in war is very simple, but the simplest thing is difficult. The difficulties accumulate and end by producing a kind of friction that is inconcievable unless one has experienced war.... Countless minor incidents – the kind you can never really foresee – combine to lower the general level of performance, so that one always falls far short of the intended goal.... every fault and exaggeration of the theory is instantly exposed in war."

In theory, designing a secure system looks simple; just see to it that users' passwords stay secret, write some code to enforce the required policy, and make sure the code gets run every time it needs to. The knowledge required does not look so very remarkable; the strategic options indeed seem obvious. But as soon as one starts to *build* the system, the faults and exaggerations of the theory multiply beyond imagination, and one in the end always falls far short of the intended goal. The *friction* of countless minor incidents opposes all efforts to perfect the system.

Even if the friction could be overcome, Sun Tzu [Tzu91] taught his emperor almost 2500 years ago that a perfect fortress is no defense against a wise enemy:

> "What is of supreme importance is to attack the enemy's strategy. Next best is to disrupt his alliances. The next best is to attack his army. The worst policy is to attack walled cities. Attack cities only when there is no alternative."

Centuries of history underscore his point. In the real world just as in our own discipline, fortresses are as often betrayed, or bypassed, or starved out, or lost to the Trojan horse, as they are carried by storm. The deadliest enemies of *our* fortresses are not cryptanalysts and "crackers" - they are social engineers, insiders, and the authors of viruses, worms, and Trojan horses. And no wall will keep them out.

Our new fundamentals should focus on attacking the enemy's strategy, rather than on building fortresses which he will simply avoid.

As evidence that this is not a hopeless task, here are a few examples[9] which suggest that the fortress assumptions (policy, integrity, secrecy) are not the only ones on which secure systems could be based:

8.1 Inherent vs. Imposed Properties

A back-of-the-envelope calculation suggests that $1 Billion US, in $100 bills, occupies perhaps 15 cubic yards. At prices current as this is written, $1 Billion US, in gold, weighs about 80 tons.

$1 Billion US, in electronic cash, on the other hand, is 32 bits plus some application-dependent headers. This is madness - surely a prescription for fraud on a breathtaking scale.

This sort of thing happens because programmers naturally think about how they can make the world "better", where better often means "faster and with less human involvement", without paying much attention to why things are the way they are in the world today.

[9]The word *examples* should be taken seriously here. This section is intended only to illustrate that the definition of security currently in vogue is not the only one *possible*. It should not be construed as a design for next year's secure systems.

The size and weight of cash is inconvenient. It was *designed* to be inconvenient - precisely so that there would be inherent limits to the scale on which fraud, smuggling, and theft are feasible. All of our value-bearing instruments, in fact, are built with these sorts of intrinsic limitations. A check's amount field is small - in part to limit the amount which can conveniently be represented in it. This is one of the reasons business checks are often printed on bigger stock than personal checks - businesses legitimately engage in bigger transactions. The temptation to make electronic cash better (than physical cash) by removing the inconvenient relationship between value and size is natural - and it should be resisted.

The software approach to building systems with limits is ordinarily to first build systems without limits and then later add limiting mechanisms. This is particularly true in the case of security; we build systems under the assumption that everyone is authorized to do everything, and then we build in *authentication* and *access control* mechanisms to limit the actions of particular users. This means that in most cases, security is a property which is imposed on the system rather than a property which is inherent in the system.

If we want to make electronic cash secure, a good start would be to give it *physicality* by making its size more proportional to its value. A sensible approach to electronic representation of cash amounts might be to take the dollar value and square it to arrive at the desired number of representation bits. Note that by making value inherent in the representation of electronic cash, we can make all implementations more secure, without imposing an assurance burden on implementation code - if your machine doesn't have 1 Billion terabits of storage, you can't steal a Billion dollars, no matter how flaky the owner's e-cash implementation is.

A variant of this approach could be used to insure payment for copyrighted information downloaded over the Internet. Copyrighted data could be transformed (for example, using a secret-sharing scheme) to make the length of its representation linearly proportional to its assigned price. Users accessing the document could then simply be charged for connect time; the properties of secret-sharing schemes guarantee that customers would have to download an entire document in order to use any part of it, and the length of the transformed representation would insure that downloaders who retrieved intelligible versions of documents have spent enough in connect-time charges to pay the copyright fee. No access control is required to implement this scheme, and no distribution of *document keys* to users is required either. Authentication is required only to accurately establish which user's account should be charged for connect time. Ruth Nelson's notion [Nel95] of *unhelpfulness* as a security policy seems related to this approach.

8.2 Economic Models

A common assumption today is that privacy requires secrecy. However this is manifestly not the case. An example will illustrate:

Imagine a system which retains medical records. Its record structure looks like this:

```
owner_bank_account
owner_amount // $100,000 ?
owner_medical_information
```

A request message looks like this:

```
requester_bank_account
requested_record
```

A request to access a medical record succeeds if and only if a transfer from `requester_bank_account` to `owner_bank_account` succeeds. This has the nice property that no policy expressed in terms of subjects is required. The system works because:

- If I access my own records, I transfer money from my own account into my own account. No harm done.
- If my doctor accesses my records, he pays me $100,000. But this is OK with him, because he'll just put it on my bill (and I'll pay, because I now have the money!)
- If Kevin Mitnick accesses my records, then my privacy has been violated. On the other hand, my new $100,000 bank balance goes some way to soothing the sting.

This system is an example of a fairly effective privacy-protection system which does not depend upon secrecy; instead it depends upon economics. (Unfortunately, it does still depend to some extent upon system integrity - even Homer sometimes nods.) It also illustrates a candidate rule for designing systems which depend minimally upon policy administration:

Make the users ask forgiveness, not permission

The basis for this rule is reversible sanctions; the sanction which penalizes bad behavior is always applied before any behavior. Later, a user who believes his action was benign can ask to have the sanction reversed. In the system described, the users ask forgiveness when they ask for their money back after accessing a protected resource. If the owner of the resource decides that the access was legitimate, she refunds the money. Otherwise, she keeps it. Punishment is swift and inevitable, which discourages attack. As Sun Tzu observes:

> "To be certain to hold what you defend is to defend a place the enemy does not attack."

8.3 Rigging the game

The previous example illustrated an economic model based on individual self-interest. It is possible to design systems which provide incentives for cooperation. Axelrod investigated such systems in [Axe84]; more recently Rosenschein and Zlotkin [RZ94] investigated *social engineering for machines*:

> "We want to understand the kinds of negotiation protocols, and punitive and incentive mechanisms, that would cause individual designers to build machines that act in particular ways. Since we assume that the agents' designers are basically interested in their own goals, we want to find interaction techniques that are "stable", that make it worthwhile for the agent designer not to have his machine deviate from the target behavior."

Note again the emphasis on security and stability as inherent, rather than imposed, properties of the system.

8.4 Immunity

Nature has been protecting systems against attack by foreign code and data for millions of years. The systems are organisms, and the foreign code and data are viruses, bacteria, and other toxins. Recent work by Stephanie Forrest and others [FHSL96], [DFH96] applies lessons learned from natural immune systems to computer security with very encouraging results.

Independently, Jeff Kephart [Kep94] has proposed epidemiologic and immune-system models for protecting networked computers against viruses.

8.5 Co-evolution of programs and data

Another assumption designers often make is that general purpose stored-program computers must have the property that all copies of a program look the same. But this is not true; it might be possible to substantially reduce the incidence of computer viruses by *co-evolving* program code and data. One way to do this might be:

- pass an application's code and all the files a user creates using that application to a secret-sharing function
- Divide the output into as many files as were provided as input (*i.e.* one file corresponding to the application's code and one each corresponding to each of the user's data files)
- Store these files in place of the originals
- When the application is invoked, re-constitute the application code and data files by recovering the shared secret

This results in a version of the application which is personalized to an individual user and the data he creates using that application. A nice property of this kind of system is that neither an application's code nor user data can be modified without corresponding changes to all other related files. Obviously, explicit import and export operations would be required to "introduce" new files into the personalized application and to prepare files for use by people other than their owners.

8.6 Safety (or at least integrity) in Numbers

The fortress approaches integrity maintenance by making one perfect copy of high-integrity data and burying it in a fallout shelter under 24-hour armed guard. Nature guarantees the integrity of some data (DNA, for example) by making millions of indifferent copies, distributing them as widely as possible, and ruthlessly exterminating those which aren't "good enough". Note that even the definition of integrity is relaxed here – from perfection to fitness for a particular purpose.

9 Agenda for the Revolution

Perhaps the reader has been convinced that the Emperor really isn't wearing any armor. If so, she will have realized by now that a lot of work lies ahead. Even if the suggestions in the previous section make some sense, the present paper plainly doesn't lay a new paradigm out in clear Copernican circles.

If today's security epicycles now seem too ugly to live with, the next steps look like these:

1. Enumerate the principles of the new security worldview.

   Here are a few nominations:

   - Assume low integrity.
   - You can't keep a secret.
   - Security should be inherent, not imposed.
   - Policy is evidence that security is imposed.
   - Identity is a side-effect of policy (don't depend on it; don't authenticate it).
   - Trust is evidence that security is imposed (trust nothing and no one).
   - Ease of use should be proportional to the probability that use is harmless.
   - Make the user ask forgiveness, not permission.
   - Plan for emergence.
   - Secrecy is not privacy.
   - Control is not protection.
   - "Confidentiality, integrity, availability" is not security.
   - Good enough is good enough. Perfect is too good.
   - Evolve!

2. Identify foundational primitives required to build the new-model world

   This paper has hinted at a few useful primitives, built by analogy with biological and economic systems.

   Other useful analogies undoubtedly exist, but we should be careful not to settle for analogy if what is really required is an unprecedented model.

3. Map out an infrastructure development programme for the new-model world

4. Set out the research agenda for the new-model world

References

[Axe84] Robert Axelrod. *The Evolution of Cooperation.* Basic Books, New York, 1984.

[Bel92] S. Bellovin. There be dragons. In *Proc. USENIX Security Symposium III*, pages 1–16. USENIX, September 1992.

[BR94] M. Bellare and P. Rogaway. Entity authentication and key distribution. In D. Stinson, editor, *Proc. IEEE Crypto 93*. Springer-Verlag, 1994. LNCS no. 773.

[BR95] M. Bellare and P. Rogaway. Optimal asymmetric encryption – how to encrypt with rsa. In A. DeSantis, editor, *Proc. Eurocrypt 94*. Springer-Verlag, 1995. LNCS no. 950.

[BS95] R. Bace and M. Schaefer. 'tsupdood? repackaged problems for you and mmi. In *Proc. New Security Foundations Workshop 1995*, pages 2–10. IEEE Computer Society Press, 1995.

[CP96] D. Cooper and C. Pfleeger. Security and privacy tc. *IEEE Computer*, pages 118–9, March 1996.

[Dav96] D. Davis. Compliance defects in public-key cryptography. In *Proc. 6th USENIX Security Symposium*, pages 171–8. USENIX, 1996.

[DFH96] P. D'haeseleer, S. Forrest, and P. Helman. An immunological approach to change detection: Algorithms, analysis, and implications. In *Proc. 1996 IEEE Symposium on Security and Privacy*, pages 110–9. IEEE, 1996.

[DFW96] D. Dean, E. Felten, and D. Wallach. Java security: from hotjava to netscape and beyond. In *Proc. 1996 IEEE Symposium on Security and Privacy*. IEEE, 1996.

[Dil94] A. Diller. *Z: An Introduction to Formal Methods*. John Wiley and Sons, second edition, 1994.

[Dye92] M. Dyer. *The Cleanroom Approach to Quality Software Development*. John Wiley and Sons, 1992.

[FHSL96] S. Forrest, S. Hofmeyr, A. Somayaji, and T. Longstaff. A sense of self for unix processes. In *Proc. 1996 IEEE Symposium on Security and Privacy*, pages 120–8. IEEE, 1996.

[Fin96] K. Finney. Mathematical notation in formal specification: Too difficult for the masses? *IEEE Trans. Software Engineering*, 22(2):158–9, February 1996.

[Gri81] D. Gries. *The Science of Programming*. Springer-Verlag, 1981.

[Kep94] J. Kephart. A biologically inspired immune system for computers. In *Artificial Life IV*, pages 130–9. MIT Press, July 1994.

[Ker96] D. Kerr. Barbarians at the firewall. *Byte*, 21(9):80NA3–8, September 1996.

[Kuh70] T. Kuhn. *The Structure of Scientific Revolutions*. University of Chicago Press, 1970.

[McC96] S. McConnell. Daily build and smoke test. *IEEE Software*, 13(4):144–3, July 1996.

[Nat88] National Computer Security Center. *Glossary of Computer Security Terms*. NCSC, 1988. Document NCSC-TG-004.

[Nat91a] National Computer Security Center. *Integrity-Oriented Control Objectives: Proposed Revisions to the Trusted Computer System Evaluation Criteria (TCSEC) DOD 5200.28-STD*. NCSC, 1991. Document C Technical Report 111-91.

[Nat91b] National Research Council. *Computers at Risk: Safe Computing In the Information Age*. National Research Council, 1991.

[Nel94] R. Nelson. What is a secret - and - what does that have to do with computer security. In *Proc. New Security Paradigms Workshop*, pages 74–81. IEEE Computer Society Press, 1994.

[Nel95] R. Nelson. Unhelpfulness as a security policy, or, it's about time. In *Proc. New Security Paradigms Workshop 1995*, pages 29–32. IEEE Computer Society, 1995.

[RL96] R. Rivest and B. Lampson. SDSI: A simple distributed security infrastructure. This appears on Ron Rivest's web page, 1996.

[RZ94] J. Rosenschein and G. Zlotkin. *Rules of Encounter: Designing Conventions for Automated Negotiation among Computers*. MIT Press, 1994.

[Spi92] J. Spivey. *The Z Notation: A Reference Manual*. Prentice Hall, second edition, 1992.

[Sul96] E. Sullivan. Internet software problems drive home security issue. *PCWeek*, 13(10):1, March 1996.

[Tho84] K. Thompson. Reflections on trusting trust. *Communications of the ACM*, August 1984. Turing Award Lecture.

[Tzu91] Sun Tzu. *The Art of War*. Shambhala Press, 1991. tr. T. Cleary.

[Ver96] Verisign, Inc. *Verisign Certification Practice Statement*. Verisign, Inc., 1996. version 1.1.

[von93] C. von Clausewitz. *On War*. Alfred A. Knopf, 1993.

[Win96] I. Winkler. The non-technical threat to computing systems. *Computing Systems*, 9(1):3–14, 1996.

[WSL95] J. Williams, M. Schaefer, and D. Landoll. Pretty good assurance. In *Proc. New Security Paradigms Workshop, 1995*, pages 82–89. IEEE Computer Society Press, 1995.