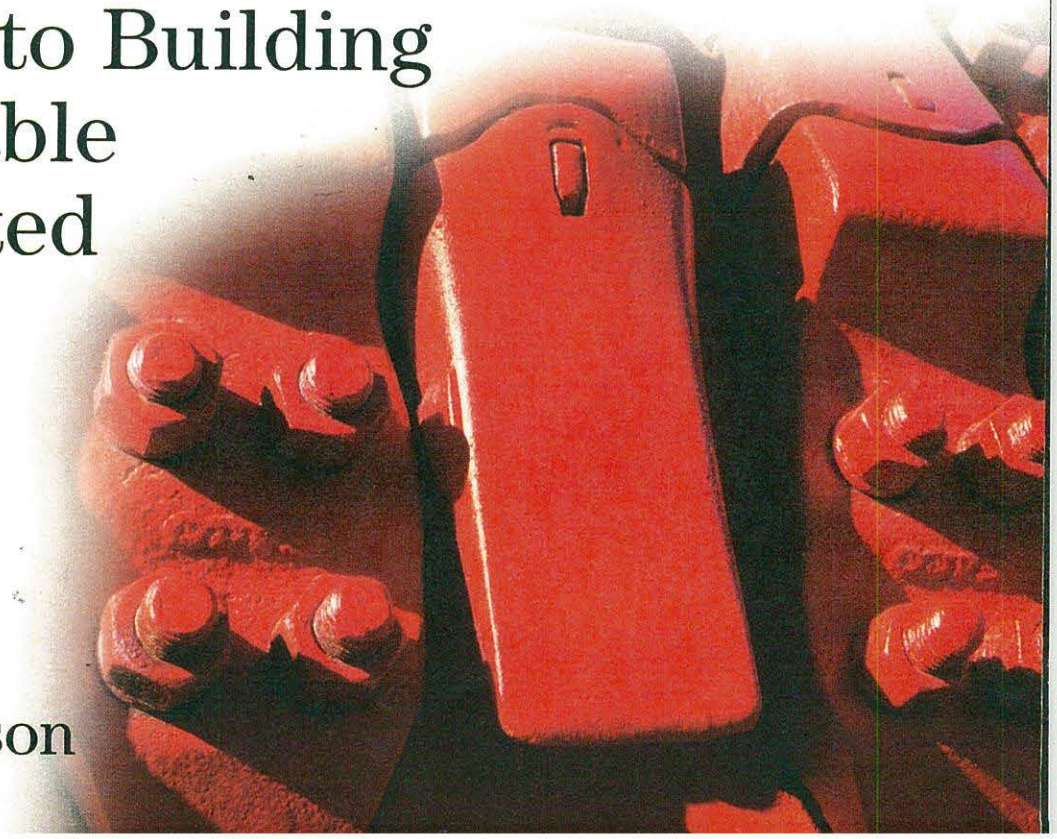
WILEY

Security Engineering

A Guide to Building Dependable Distributed Systems



Ross Anderson

Security First Innovations, LLC, Exhibit 2009
Page 2009 - 1
IPR2024-00214, Google LLC v. Security First Innovations, LLC

Security Engineering: A Guide to Building Dependable Distributed Systems

Ross J. Anderson

Wiley Computer Publishing

John Wiley & Sons, Inc.

NEW YORK • CHICHESTER • WEINHEIM • BRISBANE • SINGAPORE • TORONTO

Publisher: Robert Ipsen
Editor: Carol A. Long
Managing Editor: Micheline Frederick
Text Design & Composition: Interactive Composition Corporation

Designations used by companies to distinguish their products are often claimed as trademarks. In all instances where John Wiley & Sons, Inc., is aware of a claim, the product names appear in initial capital or ALL CAPITAL LETTERS. Readers, however, should contact the appropriate companies for more complete information regarding trademarks and registration.

This book is printed on acid-free paper. ♾

Copyright © 2001 by Ross J. Anderson. All rights reserved.

Published by John Wiley & Sons, Inc.

Published simultaneously in Canada.

No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, except as permitted under Sections 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 750-4744. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 605 Third Avenue, New York, NY 10158-0012, (212) 850-6011, fax (212) 850-6008, E-Mail: PERMREQ @ WILEY.COM.

This publication is designed to provide accurate and authoritative information in regard to the subject matter covered. It is sold with the understanding that the publisher is not engaged in professional services. If professional advice or other expert assistance is required, the services of a competent professional person should be sought.

Library of Congress Cataloging-in-Publication Data

Anderson, Ross, 1956-
Security engineering : a guide to building dependable distributed systems/Ross J. Anderson.
p. cm.
"Wiley Computer Publishing."
Includes bibliographical references and index.
ISBN 0-471-38922-6 (pbk. : alk. paper)
1. Computer security. 2. Electronic data processing—Distributed processing. I. Title.

QA76.9.A25 A54 2001
005.8—dc21 00-068486

Printed in the United States of America.

10 9 8 7 6 5 4 3 2

Research Problems

I mentioned the lack of published empirical research. Although a little has been done, there's a lot more to do. For example, what are the best ways of enforcing user compliance with a password policy? There are some extreme solutions—such as issuing each user with a printed list of random passwords, each of which can be used once only—and these certainly work. But what can be done in applications where such drastic measures aren't justified?

Another problem, which straddles the borderline with security protocol design, is whether we can design interactive password systems that are better. There are various visual schemes and memorization schemes in the literature, and some early products: one system presents users with nine faces, only one of which is of a person they know; they have to pick the right face several times in a row to log on [223]. Other schemes present a table of numbers, and let the user do a secret computation using mental arithmetic. Designing such schemes is fairly easy; evaluating them is harder, as it involves elements of cryptology, psychology, and systems engineering.

An increasingly common mechanism is to ask for several pieces of security information rather than one. A call center might ask not just for your mother's maiden name, a password, and the amount of your last purchase, but also your dog's nickname and your favorite color. The underlying idea is that although an attacker might find out anything you know, it's much harder for him to find out everything you know. Again, such schemes need careful evaluation of their usability and effectiveness using the tools of applied psychology.

Further Reading

There isn't as much literature on passwords as one would like, despite the subject's importance. The papers by Bob Morris and Ken Thompson [561], Fred Grampp and Bob Morris [350], and Dan Klein [460], are the classics. The DoD guidelines are very influential [242].

Security First Innovations, LLC, Exhibit 2009
Page 2009 - 4
IPR2024-00214, Google LLC v. Security First Innovations, LLC

CHAPTER

4

Access Control

Going all the way back to early time-sharing systems, we systems people regarded the users, and any code they wrote, as the mortal enemies of us and each other. We were like the police force in a violent slum.

—ROGER NEEDHAM

Microsoft could have incorporated effective security measures as standard, but good sense prevailed. Security systems have a nasty habit of backfiring, and there is no doubt they would cause enormous problems.

—RICK MAYBURY

4.1 Introduction

Access control is the traditional center of gravity of computer security. It is where security engineering meets computer science. Its function is to control which principals (persons, processes, machines, . . .) have access to which resources in the system—which files they can read, which programs they can execute, how they share data with other principals, and so on.

NOTE

This chapter necessarily assumes more computer science background than previous chapters, but I try to keep it to a minimum.

Security First Innovations, LLC, Exhibit 2009
Page 2009 - 5
IPR2024-00214, Google LLC v. Security First Innovations, LLC

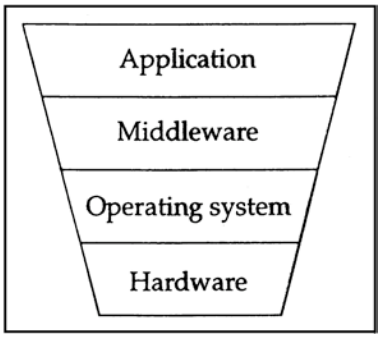


Figure 4.1 Access controls at different levels in a system.

Access control works at a number of levels, as shown in Figure 4.1, and described in the following:

1. *The access control mechanisms, which the user sees at the application level, may express a very rich and complex security policy.* A modern online business could assign staff to one of dozens of different roles, each of which could initiate some subset of several hundred possible transactions in the system. Some of these (such as credit card transactions with customers) might require online authorization from a third party while others (such as refunds) might require dual control.
2. *The applications may be written on top of middleware, such as a database management system or bookkeeping package, which enforces a number of protection properties.* For example, bookkeeping software may ensure that a transaction that debits one ledger for a certain amount must credit another ledger for the same amount.
3. *The middleware will use facilities provided by the underlying operating system.* As this constructs resources such as files and communications ports from lower-level components, it acquires the responsibility for providing ways to control access to them.
4. *Finally, the operating system access controls will usually rely on hardware features provided by the processor or by associated memory management hardware.* These control which memory addresses a given process can access.

As we work up from the hardware through the operating system and middleware to the application layer, the controls become progressively more complex and less reliable. Most actual computer frauds involve staff accidentally discovering features of the application code that they can exploit in an opportunistic way, or just abusing features of the application that they were trusted not to. But in this chapter, we will focus on the fundamentals: access control at the hardware and operating system level. (Application-level controls aren't different in principle, but I leave detailed discussion to Part 2 of this book.)

As with the other building blocks discussed so far, access control makes sense only in the context of a protection goal, typically expressed as a security policy. This puts us at a slight disadvantage when discussing PCs running single-user operating systems such as DOS and Win95/98, which have no overt security policy: any process can modify any data. People do have implicit protection goals, though; you don't expect a

Security First Innovations, LLC, Exhibit 2009
Page 2009 - 6
IPR2024-00214, Google LLC v. Security First Innovations, LLC

this). Administrators are all-powerful (so you can't create truly tamper-resistant audit trails without using write-once storage devices), but the damage they can do can be limited by suitable organization. The data structure used to manage all this, and hide the ACL details from the user interface, is called the *Registry*.

Problems with designing an NT architecture in very large organizations include naming issues (which we'll explore later), the way domains scale as the number of principals increases (badly), and the restriction that a user in another domain can't be an administrator (which can cause complex interactions between local and global groups).

One peculiarity of NT is that `everyone` is a principal, not a default or an absence of control, so `remove everyone` means just prevent a file being generally accessible. A resource can be locked quickly by setting `everyone` to have `no access`. This brings us naturally to the subject of capabilities.

4.2.5 Capabilities

The next way to manage the access control matrix is to store it by rows. These are called *capabilities*. In the example in Figure 4.2, Bob's capabilities would be as shown in Figure 4.5.

The strengths and weaknesses of capabilities are more or less the opposite of ACLs. Runtime security checking is more efficient, and we can do delegation without much difficulty: Bob could create a certificate saying "Here is my capability, and I hereby delegate to David the right to read file 4 from 9 A.M. to 1 P.M.; signed Bob." On the other hand, changing a file's status can suddenly become more tricky, as it can be difficult to find out which users have access. This can be tiresome when investigating an incident or preparing evidence of a crime.

There were a number of experimental implementations in the 1970s, which were rather like file passwords; users would get hard-to-guess bitstrings for the various read, write, and other capabilities to which they were entitled. It was found that such an arrangement could give very comprehensive protection [804]. It was not untypical to find that almost all of an operating system could run in user mode, rather than as supervisor, so operating system bugs were not security critical. (In fact, many operating system bugs caused security violations, which made debugging the operating system much easier.)

The IBM AS/400 series systems employed capability-based protection, and enjoyed some commercial success. Now capabilities are making a comeback in the form of *public key certificates*. We'll discuss the mechanisms of public key cryptography in Chapter 5, and give more concrete details of certificate-based systems, such as SSL/TLS, in Section 19.5. For now, think of a public key certificate as a credential signed by some authority, which declares that the holder of a certain cryptographic key is a certain person, a member of some group, or the holder of some privilege.

User	Operating System	Accounts Program	Accounting Data	Audit Trail
Bob	rx	r	r	r

Figure 4.5 A capability.

Security First Innovations, LLC, Exhibit 2009
Page 2009 - 7
IPR2024-00214, Google LLC v. Security First Innovations, LLC

As an example of where certificate-based capabilities can be useful, consider a hospital. If we implemented a rule stating "a nurse will have access to all the patients who are on her ward, or who have been there in the last 90 days," naively, each access control decision in the patient record system would require several references to administrative systems, to find out which nurses and which patients were on which ward, when. This means that a failure of the administrative systems can now affect patient safety much more directly than was previously the case, which is a clearly bad thing. Matters can be much simplified by giving nurses certificates that entitle them to access the files associated with their current ward. Such a system is starting to be fielded at our university hospital.

One point to bear in mind is that as public key certificates are often considered to be "crypto" rather than "access control," their implications for access control policies and architectures are not always thought through. The lessons that could have been learned from the capability systems of the 1970s are generally having to be rediscovered (the hard way). In general, the boundary between crypto and access control is a fault line where things can easily go wrong. The experts often come from different backgrounds, and the products from different suppliers.

4.2.6 Added Features in Windows 2000

A number of systems, from mainframe access control products to research systems, have combined ACLs and capabilities in an attempt to get the best of both worlds. But the most important application of capabilities is in Win2K.

Win2K adds capabilities in two ways that can override or complement the ACLs of NT4. First, users or groups can be either whitelisted or blacklisted by means of profiles. (Some limited blacklisting was also possible in NT4.) Security policy is set by groups rather than for the system as a whole. Groups are intended to be the primary method for centralized configuration management and control (group policy overrides individual profiles). Group policy can be associated with sites, domains, or organizational units, so it can start to tackle some of the real complexity problems with naming. Policies can be created using standard tools or by custom-coding (Microsoft has announced that group policy data will be exposed in a standard schema). Groups are defined in the *Active Directory*, an object-oriented database which organizes users, groups, machines, and organizational units within a domain in a hierarchical namespace, indexing them so they can searched for on any attribute. There are also finer-grained access control lists on individual resources.

As already mentioned, Win2K uses Kerberos as its main means of authenticating users across networks.[1] This is encapsulated behind the *Security Support Provider Interface* (SSPI), which enables administrators to plug in other authentication services.

[1]In fact, it's a proprietary variant, with changes to the ticket format, which prevent Win2K clients from working with existing Unix Kerberos infrastructures. The documentation for the changes is released on condition that it not be used to make compatible implementations. Microsoft's goal is to get everyone to install Win2K Kerberos servers. This has caused an outcry in the open systems community [76].

Security First Innovations, LLC, Exhibit 2009
Page 2009 - 8
IPR2024-00214, Google LLC v. Security First Innovations, LLC

This brings us to the second way in which capabilities insinuate their way into Win2K: in many applications, people are likely to use the public key protocol SSL/TLS, which is widely used on the Web, and which is based on public key certificates. The management of these certificates can provide another, capability-oriented, layer of access control outside the purview of the Active Directory. (I discuss SSL/TLS in Section 19.5.)

There are various backward-compatibility issues. For example, high-security configurations of Win2K with full cryptographic authentication can't interwork with NT4 systems. This is because an active directory can exist alongside the registry of NT4, but the registry can't read it. So the deployment of Win2K's high-security features in large organizations is likely to be delayed until all the important applications have migrated.

Win2K provides a richer and more flexible set of access control tools than any system previously sold in mass markets. It does still have design limitations. Implementing roles whose requirements differ from those of groups could be tricky in some applications; SSL certificates are the obvious way to do this, but would require an external management infrastructure. Second, Windows is still (in most of its incarnations) a single-user operating system, in the sense that only one person can operate a PC at a time. Thus, if I want to run an unprivileged, sacrificial user on my PC for accessing untrustworthy Web sites that might contain malicious code, I have to log off and log on again, or use other techniques that are so inconvenient that few users will bother. So users still do not get the benefit from the operating system's protection properties that they might wish when browsing the Web.

4.2.7 Granularity

A practical problem with all current flavors of access control system is granularity. As the operating system works with files, this will usually be the smallest object with which its access control mechanisms can deal. So it will be application-level mechanisms that, for example, ensure that a bank customer at a cash machine can see his or her own balance but not anybody else's.

But it goes deeper than that. Many applications are built using database tools that give rise to some problems that are much the same whether running DB2 on MVS or Oracle on Unix. All the application data is bundled together in one file, and the operating system must either grant or deny a user access to the lot. So, if you developed your branch accounting system under a database product, then you'll probably have to manage one access mechanism at the operating system level and another at the database or application level. Many real problems result. For example, the administration of the operating system and the database system may be performed by different departments, which do not talk to each other; and often user pressure drives IT departments to put in crude hacks that make the various access control systems seem to work as one, but that open up serious holes.

Another granularity problem is *single sign-on*. Despite the best efforts of computer managers, most large companies accumulate systems of many different architectures, so users get more and more logons to different systems; consequently, the cost of administering them escalates. Many organizations want to give each employee a single logon to all the machines on the network. A crude solution is to endow their PCs with a menu of hosts to which a logon is allowed, and hide the necessary userids and passwords in scripts. More sophisticated solutions may involve a single security server

Security First Innovations, LLC, Exhibit 2009
Page 2009 - 9
IPR2024-00214, Google LLC v. Security First Innovations, LLC

through which all logons must pass, or a smartcard to do multiple authentication protocols for different systems. Such solutions are hard to engineer properly. Whichever route one takes, the security of the best system can easily be reduced to that of the worst.

4.2.8 Sandboxing and Proof-Carrying Code

Another way of implementing access control is a software *sandbox*. Here users want to run some code that they have downloaded from the Web as an applet. Their concern is that the applet might do something nasty, such as taking a list of all their files and mailing it off to a software marketing company.

The designers of Java tackle this problem by providing a "sandbox" for such code—a restricted environment in which it has no access to the local hard disk (or at most only temporary access to a restricted directory), and is only allowed to communicate with the host it came from. These security objectives are met by having the code executed by an interpreter—the Java Virtual Machine (JVM)—which has only limited access rights [346]. Java is also used on smartcards, but (in current implementations at least) the JVM is, in effect, a compiler external to the card, which raises the issue of how the code it outputs can be gotten to the card in a trustworthy manner.

An alternative is proof-carrying code. Here, code to be executed must carry with it a proof that it doesn't do anything that contravenes the local security policy. This way, rather than using an interpreter with the resulting speed penalty, one merely has to trust a short program that checks the proofs supplied by downloaded programs before allowing them to be executed. The huge overhead of a JVM is not necessary [585].

Both of these are less general alternatives to an architecture that supports proper supervisor-level confinement.

4.2.9 Object Request Brokers

There has been much interest of late in object-oriented software development, as it has the potential to cut the cost of software maintenance. An *object* consists of code and data bundled together, accessible only through specified externally visible *methods*. This also gives the potential for much more powerful and flexible access control. Much research is underway with the goal of producing a uniform security interface that is independent of the underlying operating system and hardware.

The idea is to base security functions on the *object request broker*, or ORB, a software component that mediates communications between objects. Many research efforts focus on the *Common Object Request Broker Architecture* (CORBA), which is an attempt at an industry standard for object-oriented systems. The most important aspect of this is that an ORB is a means of controlling calls that are made across protection domains. This approach appears promising but is still under development. (A book on CORBA security is [112].)

Security First Innovations, LLC, Exhibit 2009
Page 2009 - 10
IPR2024-00214, Google LLC v. Security First Innovations, LLC

government and defense contractors, there is a protocol for establishing transient keys with forward and backward security; to exclude middleperson attacks, users have a *crypto ignition key*, a portable electronic device that they can plug into the phone to identify not just their names, but their security clearance levels. In general, books on the topic tend to talk about identification as the main goal of authentication and key management protocols; but in real life, it's usually authorization that matters. This is more complex, as it starts to introduce assumptions about the application into the protocol design. (In fact, the NSA security manual emphasizes the importance of always knowing whether there is an uncleared person in the room. The STU-III design is a natural way of extending this to electronic communications.)

One serious weakness of relying on public key certificates is the difficulty of getting users to understand all their implications and to manage them properly, especially where they are not an exact reimplementation of a familiar manual control system [224]. Many other things can go wrong with certification at the level of systems engineering as well, and we'll look at these in the next chapter.

5.7.5 The Strength of Asymmetric Cryptographic Primitives

To provide the same level of protection as a symmetric block cipher, asymmetric cryptographic primitives are believed to require at least twice the block length. Elliptic curve systems appear to achieve this; a 128-bit elliptic scheme could be about as hard to break as a 64-bit block cipher with a 64-bit key. The commoner schemes, based on factoring and discrete log, are less robust because there are shortcut attack algorithms such as the number field sieve, which exploit the fact that some integers are *smooth*, that is, they have a large number of small factors. At the time of writing, the number field sieve has been used to attack keys up to 512 bits, a task comparable in difficulty to performing keysearch on 56-bit DES keys. The current consensus is that private keys for RSA and for standard discrete log systems should be at least 1024 bits long, while 2048 bits gives some useful safety margin against mathematicians making significant improvements in algorithms.

There has been some publicity recently about *quantum computers*. These are devices that perform a large number of computations simultaneously using superposed quantum states. Peter Shor has shown that if a sufficiently large quantum computer can be built, then both factoring and discrete logarithm computations will become easy. So far only very small quantum computers can be built, and many people are sceptical about whether the technology can be made to work well enough to threaten real systems. In the event that it can, asymmetric cryptography may have to be abandoned. So it is fortunate that many of the things that are currently done with asymmetric mechanisms can also be done with symmetric ones; thus many authentication protocols can be redesigned to use variants on Kerberos.

5.8 Summary

Many ciphers fail because they're used improperly, so we need a clear model of what a cipher does. The random oracle model is useful here: we assume that each new value returned by the encryption engine is random in the sense of being statistically independent of all the outputs seen before.

Security First Innovations, LLC, Exhibit 2009
Page 2009 - 11
IPR2024-00214, Google LLC v. Security First Innovations, LLC

Block ciphers for symmetric key applications can be constructed by the careful combination of substitutions and permutations; for asymmetric applications such as public key encryption and digital signature one uses number theory. In both cases, there is quite a large body of mathematics to guide us. Other kinds of ciphers—stream ciphers and hash functions—can be constructed from block ciphers by using them in suitable modes of operation. These have different error propagation, pattern concealment, and integrity protection properties.

The basic properties the security engineer needs to understand are not too difficult to grasp, though there are some subtle things that can go wrong. In particular, it is surprisingly hard to build systems that are robust even when components fail (or are encouraged to), and where the cryptographic mechanisms are well integrated with other measures such as access control and physical security. We'll return to this repeatedly in later chapters.

Research Problems

There are many active threads in cryptography research. Many of them are where crypto meets a particular branch of mathematics (number theory, algebraic geometry, complexity theory, combinatorics, graph theory, and information theory). The empirical end of the business is concerned with designing primitives for encryption, signature, and composite operations, and that perform reasonably well on available platforms. The two meet in the study of subjects ranging from linear and differential cryptanalysis to attacks on public key protocols. Research is more driven by the existing body of knowledge than by applications, though there are exceptions: copyright protection concerns have been a stimulus, and so has the recent competition to find an Advanced Encryption Standard.

The best way to get a flavor of what's going on is to read the last few years' proceedings of research conferences, such as Crypto, Eurocrypt, Asiacrypt and Fast Software Encryption, all published by Springer-Verlag in the *Lecture Notes on Computer Science* (LNCS) series.

Further Reading

The classic papers by Diffie and Hellman [248] and by Rivest, Shamir, and Adleman [649] are the closest to required reading on this subject. The most popular modern introduction is Bruce Schneier's *Applied Cryptography* [681], which covers a lot of ground at a level a nonmathematician can understand, and which has C source code for a number of algorithms. The *Handbook of Applied Cryptography*, by Alfred Menezes, Paul von Oorschot and Scott Vanstone [544], is the closest to a standard reference book on the mathematical detail.

More specialized texts include a book by Eli Biham and Adi Shamir [102], which is the standard reference on differential cryptanalysis; the best explanation of linear cryptanalysis may be in a textbook by Doug Stinson [738]; the modern theory of block ciphers can be found developing in the papers of the *Fast Software Encryption* conference series during the 1990s (the proceedings are published by Springer-Verlag in the LNCS series). The original book on modes of operation is Carl Meyer and Steve Mat-

Security First Innovations, LLC, Exhibit 2009
Page 2009 - 12
IPR2024-00214, Google LLC v. Security First Innovations, LLC

yas [548]. Neal Koblitz has a good basic introduction to the mathematics behind public key cryptography [463]; the number field sieve is described in [497]; while quantum factoring is described in [698].

There's a shortage of good books on the random oracle model and on theoretical cryptology in general; all the published texts I've seen are very technical and heavy going. Probably the most well-regarded source is a book being written by Oded Goldreich: the online fragments of this can be found at [342]. If you need something with an ISBN, try his lecture notes on 'Modern Cryptography, Probabilistic Proofs and Pseudorandomness' [343], which are pitched at the level of a postgraduate mathematics student. A less thorough but more readable introduction to randomness and algorithms is in [360]. Current research at the theoretical end of cryptology is found at the FOCS, STOC, Crypto, Eurocrypt, and Asiacrypt conferences.

The history of cryptology is fascinating, and so many old problems keep on recurring in modern guises that the security engineer should be familiar with it. The standard work is by David Kahn [428]; there are also compilations of historical articles from *Cryptologia* [229, 227, 228], as well as several books on the history of cryptology during World War II [188, 429, 523, 800]. The NSA Museum at Fort George Meade, Maryland, is also worth a visit, as is the one at Bletchley Park in England.

Finally, no chapter that introduces public key encryption would be complete without a mention that, under the name of 'non-secret encryption,' it was first discovered by James Ellis in about 1969. However, as Ellis worked for GCHQ—Britain's Government Communications Headquarters, the equivalent of the NSA—his work remained classified. The RSA algorithm was then invented by Clifford Cocks, and also kept secret. This story is told in [267]. One effect of the secrecy was that their work was not used: although it was motivated by the expense of Army key distribution, Britain's Ministry of Defence did not start building electronic key distribution systems for its main networks until 1992. It should also be noted that the classified community did not pre-invent digital signatures; they remain the achievement of Whit Diffie and Martin Hellman.

Security First Innovations, LLC, Exhibit 2009
Page 2009 - 13
IPR2024-00214, Google LLC v. Security First Innovations, LLC

case of Sun's operating system Solaris, for example, we found in 1996 that the audit formats were not documented, and tools to read them were not available. The audit facility seemed to have been installed to satisfy the formal checklist requirements of government systems buyers, rather than to perform any useful function. There was the hope that improving this would help system administrators detect attacks, whether after the fact or even when they were still in progress.

18.5.1 Types of Intrusion Detection

The simplest intrusion detection methods involve sounding an alarm when a threshold is passed. Three or more failed logons, a credit card expenditure of more than twice the moving average of the last three months, or a mobile phone call lasting more than six hours, might all flag the account in question for attention. More sophisticated systems generally fall into two categories.

The first, *misuse detection systems*, use a model of the likely behavior of an intruder. An example would be a banking system that alarms if a user draws the maximum permitted amount from a cash machine on three successive days. Another would be a Unix intrusion detection system that looked for a user's account being taken over by someone who used the system in a much more sophisticated way; thus an account whose user previously used only simple commands would alarm if the log showed use of a compiler. An alarm might also be triggered by specific actions such as an attempt to download the password file. In general, most misuse detection systems, like antivirus scanners, look for a *signature*, a known characteristic of some particular attack. One of the most general misuse detection signatures is interest in a *honey trap*—something enticing left to attract attention. I mentioned, for example, that some hospitals maintain dummy records with celebrities' names to entrap staff who don't respect medical confidentiality.

The second type of intrusion detection strategy is *anomaly detection*. Such systems attempt the much harder job of looking for anomalous patterns of behavior in the absence of a clear model of the attacker's modus operandi. The hope is to detect attacks that have not been previously recognized and catalogued. Systems of this type often use artificial intelligence techniques—neural networks are particularly fashionable.

The dividing line between misuse and anomaly detection is somewhat blurred. A particularly good borderline case is given by Benford's law, which describes the distribution of digits in random numbers. One might expect that numbers beginning with the digits 1, 2, . . . 9 would be equally common. But, in fact, numbers that come from random natural sources, so that their distribution is independent of the number system in which they're expressed, have a logarithmic distribution: about 30% of decimal numbers start with 1. (In fact, all binary numbers start with 1, if initial zeroes are suppressed.) Crooked clerks who think up numbers to cook the books, or even use random number generators without knowing Benford's law, are often caught using it [529].

18.5.2 General Limitations of Intrusion Detection

Some intrusions are really obvious. If what you're worried about is a script kiddie vandalizing your corporate Web site, then the obvious thing to do is to have a machine in

Security First Innovations, LLC, Exhibit 2009
Page 2009 - 14
IPR2024-00214, Google LLC v. Security First Innovations, LLC

your operations room that fetches the page once a second, displays it, and rings a really loud alarm when it changes. (Make sure you do this via an outside proxy; and don't forget that it's not just your own systems at risk. The kiddie could replace your advertisers' pictures with porn, for example, in which case you'd want to pull the links to them pretty fast.)

In general however, intrusion detection is a difficult problem. Fred Cohen proved that detecting viruses (in the sense of deciding whether a program is going to do something bad) is as hard as the halting problem, meaning we can't ever expect a complete solution [192].

Another fundamental limitation comes from the fact that there are basically two different types of security failure: those that cause an error (which I defined in Section 6.2 to be an incorrect state) and those that don't. An example of the former is a theft from a bank that leaves traces on the audit trail. An example of the latter is an undetected confidentiality failure caused by a radio microphone placed by a foreign intelligence service in your room. The former can be detected (at least in principle, and forgetting for now about the halting problem) by suitable processing of the data available to you. But the latter can't be. It's a good idea to design systems so that as many failures as possible fall into the former category, but it's not always practicable [182].

There's also the matter of definitions. Some intrusion detection systems are configured to block any instances of suspicious behavior, and, in extreme cases, to take down the affected systems. Apart from opening the door to service denial attacks, this turns the intrusion detection system into an access control mechanism. As we've already seen, access control is in general a hard problem, that incorporates all sorts of issues of security policy which people often disagree on or simply get wrong. (The common misconceptions that you can do access control with intrusion detection mechanisms and that all intrusion detection can be done with neural networks together would imply that some neural network bolted on to a LAN could be trained to enforce something like Bell-LaPadula. This seems fatuous.)

I prefer to define an intrusion detection system as one that monitors the logs and draws the attention of authority to suspicious occurrences. This is closer to the way mobile phone operators work. It's also critical in financial investigations; see [658] for a discussion, by a special agent with the U.S. Internal Revenue Service, of what he looks for when trying to trace hidden assets and income streams. A lot hangs on educated suspicion, based on long experience. For example, a $25 utility bill may lead to a $250,000 second house hidden behind a nominee. Building an effective system means having the people, and the machines, each do the part of the job they're best at; and this means getting the machine to do the preliminary filtering.

Then there's the cost of false alarms. For example, I used to go to San Francisco every May, and I got used to the fact that after I'd used my U.K. debit card in an ATM five days in a row, it would stop working. Not only does this upset the customer, but villains quickly learn to exploit it (as do the customers—I just started making sure I got enough dollars out in the first five days to last me the whole trip). As in so many security engineering problems, the trade-off between the fraud rate and the insult rate is the critical one; And, as I noted in Chapter 13, "Biometrics," Section 13.8, we can't expect to improve this trade-off simply by looking at lots of different indicators. In general, we must expect that an opponent will always get past the threshold if he or she is patient enough, and either does the attack very slowly or does a large number of small attacks.

Security First Innovations, LLC, Exhibit 2009
Page 2009 - 15
IPR2024-00214, Google LLC v. Security First Innovations, LLC

A particularly intractable problem with commercial intrusion detection systems is *redlining.* When insurance companies used claim statistics on postcodes to decide the level of premiums to charge, it was found that many poor and minority areas suffered high premiums or were excluded altogether from coverage. In a number of jurisdictions, this is now illegal. But the problem is much broader. For example, Washington is pushing airlines to bring in systems to profile passengers for terrorism risk, so they can be subjected to more stringent security checks. The American-Arab Anti-Discrimination Committee has reported many incidents where innocent passengers have been harassed by airlines that have implemented some of these recommendations [516].

In general, if you build an intrusion detection system based on data-mining techniques, you are at serious risk of discriminating. If you use neural network techniques, you'll have no way of explaining to a court what the rules underlying your decisions are, so defending yourself could be hard. Opaque rules can also contravene European data protection law, which entities citizens to know the algorithms used to process their personal data.

In general, most fielded intrusion detection systems use a number of different techniques [661]. They tend to draw heavily on knowledge of the application, and to be developed by slow evolution.

18.5.3 Specific Problems Detecting Network Attacks

Turning now to the specific problem of detecting network intrusion, the problem is much harder than, say, detecting mobile phone cloning, for a number of reasons. For starters, the available products still don't work very well, with success rates of perhaps 60–80% in laboratory tests and a high false alarm rate. For example, at the time of writing, the U.S. Air Force has so far not detected an intrusion using the systems it has deployed on local networks—although once one is detected by other means, the traces can be found on the logs.

The reasons for the poor performance include the following, in no particular order.

- *The Internet is a very "noisy" environment, not just at the level of content but also at the packet level.* A large amount of random crud arrives at any substantial site, and enough of it can be interpreted as hostile to generate a significant false alarm rate. A survey by Bellovin [89] reports that many bad packets result from software bugs; others are the fault of out-of-date or corrupt DNS data; and some are local packets that escaped, travelled the world, and returned.
- *There are too few attacks.* If there are ten real attacks per million sessions—which is almost certainly an overestimate—then even if the system has a false alarm rate as low as 0.1%, the ratio of false to real alarms will be 100. I talked about similar problems with burglar alarms in Chapter 10; it's also a well known issue for medics running screening programs for diseases such as HIV where the test error exceeds the organism's prevalence in the population. In general, where the signal is so far below the noise, an alarm system is likely to so fatigue the guards that even the genuine alarms get missed.

Security First Innovations, LLC, Exhibit 2009
Page 2009 - 16
IPR2024-00214, Google LLC v. Security First Innovations, LLC