Information Security: Principles and Practice, Second Edition
by Mark Stamp

Chapter 8

Authorization

It is easier to exclude harmful passions than to rule them, and to deny them admittance than to control them after they have been admitted.
— Seneca

You can always trust the information given to you by people who are crazy; they have an access to truth not available through regular channels.
— Sheila Ballantyne

8.1 Introduction

Authorization is the part of access control concerned with restrictions on the actions of authenticated users. In our terminology, authorization is one aspect of access control and authentication is another. Unfortunately, some authors use the term "access control" as a synonym for authorization.

In the previous chapter we discussed authentication, where the issue is one of establishing identity. In its most basic form, authorization deals with the situation where we've already authenticated Alice and we want to enforce restrictions on what she is allowed to do. Note that while authentication is binary (either a user is authenticated or not), authorization can be a much more fine grained process.

In this chapter, we'll extend the traditional notion of authorization to include a few non-traditional topics. We'll discuss CAPTCHAs, which are designed to restrict access to humans (as opposed to computers), and we'll consider firewalls, which can be viewed as a form of access control for networks. We'll follow up the section on firewalls with a discussion of intrusion detection systems, which come into play when firewalls fail to keep the bad guys out.

8.2 A Brief History of Authorization

History is ... bunk.
— Henry Ford

Back in the computing dark ages,[1] authorization was often considered the heart of information security. Today, that seems like a rather quaint notion. In any case, it is worth briefly considering the historical context from which modern information security has arisen.

While cryptography has a long and storied history, other aspects of modern information security are relative newcomers. Here, we take a brief look at the history of system certification, which, in some sense, represents the modern history of authorization. The goal of such certification regimes is to give users some degree of confidence that the systems they use actually provide a specified level of security. While this is a laudable goal, in practice, system certification is often laughable. Consequently, certification has never really become a significant piece of the security puzzle—as a rule, only those products that absolutely must be certified are. And why would any product need to be certified? Governments, which created the certification regimes, require certification for certain products that they purchase. So, as a practical matter, certification is generally only an issue if you are trying to sell your product to the government.[2]

8.2.1 The Orange Book

The Trusted Computing System Evaluation Criteria (TCSEC), or "orange book" [309] (so called because of the color of its cover) was published in 1983. The orange book was one of a series of related books developed under the auspices of the National Security Agency. Each book had a different colored cover and collectively they are known as the "rainbow series." The orange book primarily deals with system evaluation and certification and, to some extent, multilevel security—a topic discussed later in this chapter.

Today, the orange book is of little, if any, practical relevance. Moreover, in your opinionated author's opinion, the orange book served to stunt the growth of information security by focusing vast amounts of time and resources on some of the most esoteric and impractical aspects of security.[3]

Of course, not everyone is as enlightened as your humble author, and, in some circles, there is still something of a religious fervor for the orange book

[1] That is, before the Apple Macintosh was invented.

[2] It's tempting to argue that certification is an obvious failure simply because there is no evidence that the government is any more secure than anybody else, in spite of its use of certified security products. However, your certifiable author will, for once, refrain from making such a smug and unsubstantiated (but oddly satisfying) claim.

[3] Other than that, the orange book was a smashing success.

and its view of the security universe. In fact, the faithful tend to believe that if only the orange book way of thinking had prevailed, we'd all be much more secure today.

So, is it worth knowing something about the orange book? Well, it's always good to have some historical perspective on any subject. Also, as previously mentioned, there are some people who still take it seriously (although fewer and fewer each day), and you may need to deal with such a person at some point. In addition, it is possible that you may need to worry about system certification.

The stated purpose of the orange book is to provide criteria for assessing the effectiveness of the security provided by "automatic data processing system products." The overriding goals, as given in [309], are:

a. To provide users with a yardstick with which to assess the degree of trust that can be placed in computer systems for the secure processing of classified or other sensitive information.

b. To provide guidance to manufacturers as to what to build into their new, widely available trusted commercial products in order to satisfy trust requirements for sensitive applications.

c. To provide a basis for specifying security requirements in acquisition specifications.

In short, the orange book intended to provide a way to assess the security of existing products and to provide guidance on how to build more secure products. The practical effect was that the orange book provided the basis for a certification regime that could be used to provide a security rating to a security product. In typical governmental fashion, the certification was to be determined by navigating through a complex and ill-defined maze of rules and requirements.

The orange book proposes four broad divisions, labeled as D through A, with D being the lowest and A the highest. Most of the divisions are split into classes. For example, under the C division, we have classes C1 and C2. The four divisions and their corresponding classes are as follows.

D. Minimal protection — This division contains only one class which is reserved for those systems that can't meet the requirements for any higher class. That is, these are the losers that couldn't make it into any "real" class.

C. Discretionary protection — There are two classes here, both of which provide some level of "discretionary" protection. That is, they don't necessarily force security on users, but instead they provide some means of detecting security breaches—specifically, there must be an audit capability. The two classes in this division are the following.

C1. Discretionary security protection — In this class, a system must provide "credible controls capable of enforcing access limitations on an individual basis."[4]

C2. Controlled access protection — Systems in this class "enforce a more finely grained discretionary access control than (C1) systems."[5]

B. Mandatory protection — This a big step up from C. The idea of the C division is that users can break the security, but they might get caught. However, for division B, the protection is mandatory, in the sense that users cannot break the security even if they try. The B classes are the following.

B1. Labeled security protection — Mandatory access control is based on specified labels. That is, all data carries some sort of label, which determines which users can do what with the data. Also, the access control is enforced in a way so that users cannot violate it (i.e., the access control is mandatory).

B2. Structured protection — This adds covert channel protection (discussed later in this chapter) and a few other technical issues on top of B1.

B3. Security domains — On top of B2 requirements, this class adds that the code that enforces security must "be tamperproof, and be small enough to be subjected to analysis and tests." We'll have much more to say about software issues in later chapters. For now, it is worth mentioning that making software tamperproof is, at best, difficult and expensive, and is seldom attempted in any serious way.

A. Verified protection — This is the same as B3, except that so-called formal methods must be used to, in effect, prove that the system does what is claimed. In this division there is a class A1 and a brief discussion of what might lie beyond A1.

The A division was certainly very optimistic for a document published in the 1980s, since the formal proofs that it envisions are not yet feasible for systems of moderate or greater complexity. As a practical matter, satisfying the C level requirements should be, in principle, almost trivial, but even today, achieving any of the B (or higher) classes would be a challenging task, except, perhaps, for certain straightforward applications (e.g., digital signature software).

[4]Hmm...

[5]Yes, of course, it's all so clear now...

There is a second part to the orange book that covers the "rationale and criteria," that is, it gives the reasoning behind the various requirements outlined above, and it attempts to provide specific guidance on how to meet the requirements. The rationale section includes a brief discussion of such topics as a reference monitor and a trusted computing base—topics that we will mention in our final chapter. There is also a brief discussion of the Bell-LaPadula security model, which we cover later in this chapter.

The criteria (i.e., the guidelines) section is certainly much more specific than the general discussion of the classes, but it is not clear that the guidelines are really all that useful or sensible. For example, under the title of "testing for division C" we have the following guidance, where "team" refers to the security testing team [309].

> The team shall independently design and implement at least five system-specific tests in an attempt to circumvent the security mechanisms of the system. The elapsed time devoted to testing shall be at least one month and need not exceed three months. There shall be no fewer than twenty hands-on hours spent carrying out system developer-defined tests and test team-defined tests.

While this is specific, it's not difficult to imagine a scenario where one team could accomplish more in a few hours of automated testing than another team could accomplish in three months of manual testing.[6]

8.2.2 The Common Criteria

Formally, the orange book has been superseded by the cleverly named Common Criteria [65], which is an international government-sponsored standard for certifying security products. The Common Criteria is similar to the orange book in the sense that, as much as is humanly possible, it is ignored in practice. However, if you want to sell your security product to the government, it may be necessary to obtain some specified level of Common Criteria certification. Even the lower-level Common Criteria certifications can be costly to obtain (on the order of six figures, in U.S. dollars), and the higher-level certifications are prohibitively expensive due to many fanciful requirements.

A Common Criteria certification yields a so-called Evaluation Assurance Level (EAL) with a numerical rating from 1 to 7, that is, EAL1 through EAL7, where the higher the number, the better. Note that a product with a higher EAL is not necessarily more secure than a product with a lower (or no) EAL. For example, suppose that product A is certified EAL4, while product B

[6] As an aside, your easily annoyed author finds it highly ironic and somewhat disturbing that the same people who gave us the dubious orange book now want to set educational standards in information security; see [216].

carries an EAL5 rating. All this means is that product A was evaluated for EAL4 (and passed), while product B was actually evaluated for EAL5 (and passed). It is possible that product A could actually have achieved EAL5 or higher, but the developers simply felt it was not worth the cost and effort to try for a higher EAL. The different EALs are listed below [106].

- EAL1 — Functionally Tested
- EAL2 — Structurally Tested
- EAL3 — Methodically Tested and Checked
- EAL4 — Methodically Designed, Tested, and Reviewed
- EAL5 — Semiformally Designed and Tested
- EAL6 — Semiformally Verified Design and Tested
- EAL7 — Formally Verified Design and Tested

To obtain an EAL7 rating, formal proofs of security must be provided, and security experts carefully analyze the product. In contrast, at the lowest EALs, the documentation is all that is analyzed. Of course, at an intermediate level, something between these two extremes is required.

Certainly the most commonly sought-after Common Criteria certification is EAL4, since it is generally the minimum required to sell to the government. Interestingly, your hard-working author could find a grand total of precisely two products certified at the highest Common Criteria level, EAL7. This is not an impressive number considering that this certification regime has been around for more than a decade and it is an international standard.

And who are these security "experts" who perform Common Criteria evaluations? The security experts work for government-accredited Common Criteria Testing Laboratories—in the U.S. the accrediting agency is NIST.

We won't go into the details of Common Criteria certification here.[7] In any case, the Common Criteria will never evoke the same sort of passions (pro or con) as the orange book. Whereas the orange book is, in a sense, a philosophical statement claiming to provide the answers about how to do security, the Common Criteria is little more than a mind-numbing bureaucratic hurdle that must be overcome if you want to sell your product to the government. It is also worth noting that whereas the orange book is only about 115 pages long, due to inflation, the Common Criteria documentation exceeds 1000 pages. Consequently, few mortals will ever read the Common

[7]During your tireless author's two years at a small startup company, he spent an inordinate amount of time studying the Common Criteria documentation—his company was hoping to sell its product to the U.S. government. Because of this experience, mere mention of the Common Criteria causes your usually hypoallergenic author to break out in hives.

Criteria, which is another reason why it will never evoke more than a yawn from the masses.

Next, we consider the classic view of authorization. Then we look at multilevel security (and related topics) before considering a few cutting-edge topics, including firewalls, IDS, and CAPTCHAs.

8.3 Access Control Matrix

The classic view of authorization begins with Lampson's access control matrix [5]. This matrix contains all of the relevant information needed by an operating system to make decisions about which users are allowed to do what with the various system resources.

We'll define a *subject* as a user of a system (not necessarily a human user) and an *object* as a system resource. Two fundamental constructs in the field of authorization are *access control lists*, or ACLs, and *capabilities*, or C-lists. Both ACLs and C-lists are derived from Lampson's *access control matrix*, which has a row for every subject and a column for every object. Sensibly enough, the access allowed by subject S to object O is stored at the intersection of the row indexed by S and the column indexed by O. An example of an access control matrix appears in Table 8.1, where we use UNIX-style notation, that is, `x`, `r`, and `w` stand for execute, read, and write privileges, respectively.

Table 8.1: Access Control Matrix

	OS	Accounting program	Accounting data	Insurance data	Payroll data
Bob	`rx`	`rx`	`r`	—	—
Alice	`rx`	`rx`	`r`	`rw`	`rw`
Sam	`rwx`	`rwx`	`r`	`rw`	`rw`
Accounting program	`rx`	`rx`	`rw`	`rw`	`r`

Notice that in Table 8.1, the accounting program is treated as both an object and a subject. This is a useful fiction, since we can enforce the restriction that the accounting data is only modified by the accounting program. As discussed in [14], the intent here is to make corruption of the accounting data more difficult, since any changes to the accounting data must be done by software that, presumably, includes standard accounting checks and balances. However, this does not prevent all possible attacks, since the system administrator, Sam, could replace the accounting program with a faulty (or fraudulent) version and thereby break the protection. But this trick does

allow Alice and Bob to access the accounting data without allowing them to corrupt it—either intentionally or unintentionally.

8.3.1 ACLs and Capabilities

Since all subjects and all objects appear in the access control matrix, it contains all of the relevant information on which authorization decisions can be based. However, there is a practical issue in managing a large access control matrix. A system could have hundreds of subjects (or more) and tens of thousands of objects (or more), in which case an access control matrix with millions of entries (or more) would need to be consulted before any operation by any subject on any object. Dealing with such a large matrix could impose a significant burden on the system.

To obtain acceptable performance for authorization operations, the access control matrix can be partitioned into more manageable pieces. There are two obvious ways to split the access control matrix. First, we could split the matrix into its columns and store each column with its corresponding object. Then, whenever an object is accessed, its column of the access control matrix would be consulted to see whether the operation is allowed. These columns are known as access control lists, or ACLs. For example, the ACL corresponding to insurance data in Table 8.1 is

$$(\text{Bob}, —), (\text{Alice}, \texttt{rw}), (\text{Sam}, \texttt{rw}), (\text{accounting program}, \texttt{rw}).$$

Alternatively, we could store the access control matrix by row, where each row is stored with its corresponding subject. Then, whenever a subject tries to perform an operation, we can consult its row of the access control matrix to see if the operation is allowed. This approach is know as capabilities, or C-lists. For example, Alice's C-list in Table 8.1 is

$$(\text{OS}, \texttt{rx}), (\text{accounting program}, \texttt{rx}), (\text{accounting data}, \texttt{r}), (\text{insurance data}, \texttt{rw}), (\text{payroll data}, \texttt{rw}).$$

It might seem that ACLs and C-lists are equivalent, since they simply provide different ways of storing the same information. However, there are some subtle differences between the two approaches. Consider the comparison of ACLs and capabilities illustrated in Figure 8.1.

Note that the arrows in Figure 8.1 point in opposite directions, that is, for ACLs, the arrows point from the resources to the users, while for capabilities, the arrows point from the users to the resources. This seemingly trivial difference has real significance. In particular, with capabilities, the association between users and files is built into the system, while for an ACL-based system, a separate method for associating users to files is required. This illustrates one of the inherent advantages of capabilities. In fact, capabilities have several security advantages over ACLs and, for this reason, C-lists are

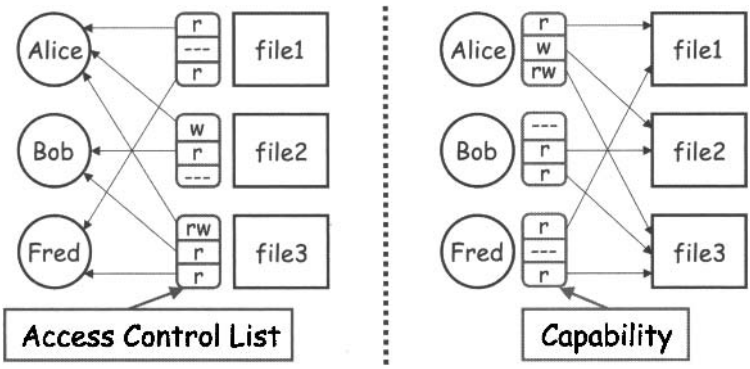

Figure 8.1: ACLs versus Capabilities

much beloved within the academic research community [206]. In the next section, we discuss one potential security advantage of capabilities over ACLs. Then we move on to the topic of multilevel security.

8.3.2 Confused Deputy

The *confused deputy* is a classic security problem that arises in many contexts [139]. For our illustration of this problem, we consider a system with two resources, a compiler and a file named BILL that contains critical billing information, and one user, Alice. The compiler can write to any file, while Alice can invoke the compiler and she can provide a filename where debugging information will be written. However, Alice is not allowed to write to the file BILL, since she might corrupt the billing information. The access control matrix for this scenario appears in Table 8.2.

Table 8.2: Access Control Matrix for Confused Deputy Example

	Compiler	BILL
Alice	x	—
Compiler	rx	rw

Now suppose that Alice invokes the compiler, and she provides BILL as the debug filename. Alice does not have the privilege to access the file BILL, so this command should fail. However, the compiler, which is acting on Alice's behalf, does have the privilege to overwrite BILL. If the compiler acts with its privilege, then a side effect of Alice's command will be the trashing of the BILL file, as illustrated in Figure 8.2.

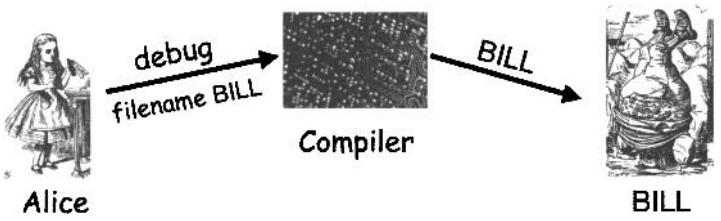

Figure 8.2: Confused Deputy

Why is this problem known as the confused deputy? The compiler is acting on Alice's behalf, so it is her deputy. The compiler is confused since it is acting based on its own privileges when it should be acting based on Alice's privileges.

With ACLs, it's more difficult (but not impossible) to avoid the confused deputy. In contrast, with capabilities it's relatively easy to prevent this problem, since capabilities are easily delegated, while ACLs are not. In a capabilities-based system, when Alice invokes the compiler, she can simply give her C-list to the compiler. The compiler then consults Alice's C-list when checking privileges before attempting to create the debug file. Since Alice does not have the privilege to overwrite `BILL`, the situation in Figure 8.2 can be avoided.

A comparison of the relative advantages of ACLs and capabilities is instructive. ACLs are preferable when users manage their own files and when protection is data oriented. With ACLs, it's also easy to change rights to a particular resource. On the other hand, with capabilities it's easy to delegate (and sub-delegate and sub-sub-delegate, and so on), and it's easier to add or delete users. Due to the ability to delegate, it's easy to avoid the confused deputy when using capabilities. However, capabilities are more complex to implement and they have somewhat higher overhead—although it may not be obvious, many of the difficult issues inherent in distributed systems arise in the context of capabilities. For these reasons, ACLs are used in practice far more often than capabilities.

8.4 Multilevel Security Models

In this section we briefly discuss security modeling in the context of multilevel security. Security models are often presented at great length in information security textbooks, but here we'll only mention two of the best-known models, and we only present an overview of these models. For a more thorough introduction to MLS and related security models, see [283] or Gollmann's book [125].

In general, security models are descriptive, not proscriptive. That is, these models tell us what needs to be protected, but they don't answer the real question, that is, how to provide such protection. This is not a flaw in the models, as they are designed to set a framework for protection, but it is an inherent limitation on the practical utility of security modeling.

Multilevel security, or MLS, is familiar to all fans of spy novels, where classified information often figures prominently. In MLS, the subjects are the users (generally, human) and the objects are the data to be protected (for example, documents). Furthermore, *classifications* apply to objects while *clearances* apply to subjects.

The U.S. Department of Defense, or DoD, employs four levels of classifications and clearances, which can be ordered as

$$\text{TOP SECRET} > \text{SECRET} > \text{CONFIDENTIAL} > \text{UNCLASSIFIED}. \tag{8.1}$$

For example, a subject with a SECRET clearance is allowed access to objects classified SECRET or lower but not to objects classified TOP SECRET. Apparently to make them more visible, security levels are generally rendered in upper case.

Let O be an object and S a subject. Then O has a classification and S has a clearance. The security *level* of O is denoted $L(O)$, and the security level of S is similarly denoted $L(S)$. In the DoD system, the four levels shown above in (8.1) are used for both clearances and classifications. Also, for a person to obtain a SECRET clearance, a more-or-less routine background check is required, while a TOP SECRET clearance requires an extensive background check, a polygraph exam, a psychological profile, etc.

There are many practical problems related to the classification of information. For example, the proper classification is not always clear, and two experienced users might have widely differing views. Also, the level of granularity at which to apply classifications can be an issue. It's entirely possible to construct a document where each paragraph, when taken individually, is UNCLASSIFIED, yet the overall document is TOP SECRET. This problem is even worse when source code must be classified, which is sometimes the case within the DoD. The flip side of granularity is aggregation—an adversary might be able to glean TOP SECRET information from a careful analysis of UNCLASSIFIED documents.

Multilevel security is needed when subjects and objects at different levels use the same system resources. The purpose of an MLS system is to enforce a form of access control by restricting subjects so that they only access objects for which they have the necessary clearance.

Military and government have long had an interest in MLS. The U.S. government, in particular, has funded a great deal of research into MLS and, as a consequence, the strengths and weaknesses of MLS are relatively well understood.

Today, there are many potential uses for MLS outside of its traditional classified government setting. For example, most businesses have information that is restricted to, say, senior management, and other information that is available to all management, while still other proprietary information is available to everyone within the company and, finally, some information is available to everyone, including the general public. If this information is stored on a single system, the company must deal with MLS issues, even if they don't realize it. Note that these categories correspond directly to the TOP SECRET, SECRET, CONFIDENTIAL, and UNCLASSIFIED classifications discussed above.

There is also interest in MLS in such applications as network firewalls. The goal in such a case is to keep an intruder, Trudy, at a low level to limit the damage that she can inflict after she breaches the firewall. Another MLS application that we'll examine in more detail below deals with private medical information.

Again, our emphasis here is on MLS models, which explain what needs to be done but do not tell us how to implement such protection. In other words, we should view these models as high-level descriptions, not as security algorithms or protocols. There are many MLS models—we'll only discuss the most elementary. Other models can be more realistic, but they are also more complex and harder to analyze and verify.

Ideally, we would like to prove results about security models. Then any system that satisfies the assumptions of the model automatically inherits all of the results that have been proved about the model. However, we will not delve so deeply into security models in this book.

8.4.1 Bell-LaPadula

The first security model that we'll consider is Bell-LaPadula, or BLP, which, believe it or not, was named after its inventors, Bell and LaPadula. The purpose of BLP is to capture the minimal requirements, with respect to confidentiality, that any MLS system must satisfy. BLP consists of the following two statements:

> **Simple Security Condition**: Subject S can read object O if and only if $L(O) \leq L(S)$.
>
> ***-Property** (Star Property): Subject S can write object O if and only if $L(S) \leq L(O)$.

The simple security condition merely states that Alice, for example, cannot read a document for which she lacks the appropriate clearance. This condition is clearly required of any MLS system.

The star property is somewhat less obvious. This property is designed to prevent, say, TOP SECRET information from being written to, say, a SECRET document. This would break MLS security since a user with a SECRET clearance could then read TOP SECRET information. The writing could occur intentionally or, for example, as the result of a computer virus. In his groundbreaking work on viruses, Cohen mentions that viruses could be used to break MLS security [60], and such attacks remain a very real threat to MLS systems today.

The simple security condition can be summarized as "no read up," while the star property implies "no write down." Consequently, BLP is sometimes succinctly stated as "no read up, no write down." It's difficult to imagine a security model that's any simpler.

Although simplicity in security is a good thing, BLP may be too simple. At least that is the conclusion of McLean, who states that BLP is "so trivial that it is hard to imagine a realistic security model for which it does not hold" [198]. In an attempt to poke holes in BLP, McLean defined a "system Z" in which an administrator is allowed to temporarily reclassify objects, at which point they can be "written down" without violating BLP. System Z clearly violates the spirit of BLP, but, since it is not expressly forbidden, it is apparently allowed.

In response to McLean's criticisms, Bell and LaPadula fortified BLP with a *tranquility property*. Actually, there are two versions of this property. The strong tranquility property states that security labels can never change. This removes McLean's system Z from the BLP realm, but it's also impractical in the real world, since security labels must sometimes change. For example, the DoD regularly declassifies documents, which would be impossible under strict adherence to the strong tranquility property. For another example, it is often desirable to enforce *least privilege*. If a user has, say, a TOP SECRET clearance but is only browsing UNCLASSIFIED Web pages, it is desirable to only give the user an UNCLASSIFIED clearance, so as to avoid accidentally divulging classified information. If the user later needs a higher clearance, his active clearance can be upgraded. This is known as the *high water mark principle*, and we'll see it again when we discuss Biba's model, below.

Bell and Lapadula also offered a *weak tranquility property* in which a security label can change, provided such a change does not violate an "established security policy." Weak tranquility can defeat system Z, and it can allow for least privilege, but the property is so vague as to be nearly meaningless for analytic purposes.

The debate concerning BLP and system Z is discussed thoroughly in [34], where the author points out that BLP proponents and McLean are each making fundamentally different assumptions about modeling. This debate gives rise to some interesting issues concerning the nature—and limits—of modeling.

The bottom line regarding BLP is that it's very simple, and as a result it's one of the few models for which it's possible to prove things about systems. Unfortunately, BLP may be too simple to be of any practical benefit.

BLP has inspired many other security models, most of which strive to be more realistic. The price that these systems pay for more reality is more complexity. This makes most other models more difficult to analyze and more difficult to apply, that is, it's more difficult to show that a real-world system satisfies the requirements of the model.

8.4.2 Biba's Model

In this section, we'll look briefly at Biba's model. Whereas BLP deals with confidentiality, Biba's model deals with integrity. In fact, Biba's model is essentially an integrity version of BLP.

If we trust the integrity of object O_1 but not that of object O_2, then if object O is composed of O_1 and O_2, we cannot trust the integrity of object O. In other words, the integrity level of O is the minimum of the integrity of any object contained in O. Another way to say this is that for integrity, a low water mark principle holds. In contrast, for confidentiality, a high water mark principle applies.

To state Biba's model formally, let $I(O)$ denote the integrity of object O and $I(S)$ the integrity of subject S. Biba's model is defined by the two statements:

> **Write Access Rule**: Subject S can write object O if and only if $I(O) \leq I(S)$.
>
> **Biba's Model**: A subject S can read the object O if and only if $I(S) \leq I(O)$.

The write access rule states that we don't trust anything that S writes any more than we trust S. Biba's model states that we can't trust S any more than the lowest integrity object that S has read. In essence, we are concerned that S will be "contaminated" by lower integrity objects, so S is forbidden from viewing such objects.

Biba's model is actually very restrictive, since it prevents S from ever viewing an object at a lower integrity level. It's possible—and, in many cases, perhaps desirable—to replace Biba's model with the following:

> **Low Water Mark Policy**: If subject S reads object O, then $I(S) = \min\big(I(S), I(O)\big)$.

Under the low water mark principle, subject S can read anything, under the condition that the integrity of subject S is downgraded after accessing an object at a lower level.

Figure 8.3 illustrates the difference between BLP and Biba's model. Of course the fundamental difference is that BLP is for confidentiality, which implies a high water mark principle, while Biba is for integrity, which implies a low water mark principle.

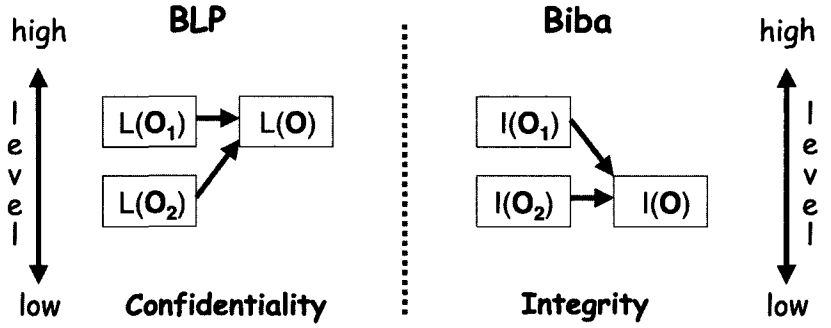

Figure 8.3: BLP versus Biba

8.5 Compartments

Multilevel security systems enforce access control (or information flow) "up and down," where the security levels are ordered in a hierarchy, such as (8.1). Usually, a simple hierarchy of security labels is not flexible enough to deal with a realistic situation. In practice, it is usually necessary to also use *compartments* to further restrict information flow "across" security levels.

We use the notation

SECURITY LEVEL {COMPARTMENT}

to denote a security level and its associated compartment or compartments. For example, suppose that we have compartments CAT and DOG within the TOP SECRET level. Then we would denote the resulting compartments as TOP SECRET {CAT} and TOP SECRET {DOG}. Note that there is also a TOP SECRET {CAT,DOG} compartment. While each of these compartments is TOP SECRET, a subject with a TOP SECRET clearance can only access a compartment if he or she is specifically allowed to do so. As a result, compartments have the effect of restricting information flow across security levels.

Compartments serve to enforce the *need to know* principle, that is, subjects are only allowed access to the information that they must know for their work. If a subject does not have a legitimate need to know everything at, say,

the TOP SECRET level, then compartments can be used to limit the TOP SECRET information that the subject can access.

Why create compartments instead of simply creating a new classification level? It may be the case that, for example, TOP SECRET {CAT} and TOP SECRET {DOG} are not comparable, that is, neither

$$\text{TOP SECRET } \{\text{CAT}\} \leq \text{TOP SECRET } \{\text{DOG}\}$$

nor

$$\text{TOP SECRET } \{\text{CAT}\} \geq \text{TOP SECRET } \{\text{DOG}\}$$

holds. Using a strict MLS hierarchy, one of these two conditions must hold true.

Consider the compartments in Figure 8.4, where the arrows represent "$\geq$" relationships. In this example, a subject with a TOP SECRET {CAT} clearance does not have access to information in the TOP SECRET {DOG} compartment. In addition, a subject with a TOP SECRET {CAT} clearance has access to the SECRET {CAT} compartment but not to the compartment SECRET {CAT,DOG}, even though the subject has a TOP SECRET clearance. Again, compartments provide a means to enforce the need to know principle.

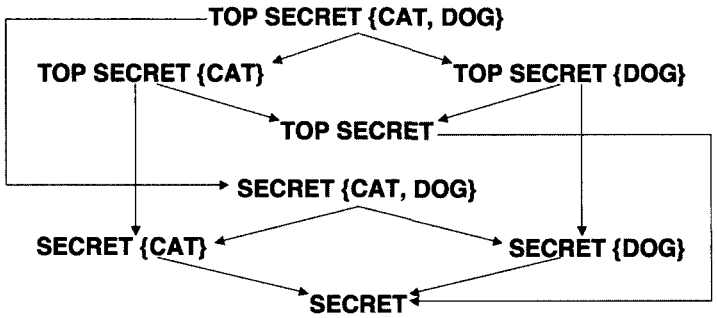

Figure 8.4: Compartments Example

Multilevel security can be used without compartments and vice versa, but the two are usually used together. An interesting example described in [14] concerns the protection of personal medical records by the British Medical Association, or BMA. The law that required protection of medical records mandated a multilevel security system—apparently because lawmakers were familiar with MLS. Certain medical conditions, such as AIDS, were considered to be the equivalent of TOP SECRET, while other less sensitive information, such as drug prescriptions, was considered SECRET. But if a subject had been prescribed AIDS drugs, anyone with a SECRET clearance could easily deduce TOP SECRET information. As a result, all information tended

to be classified at the highest level, and consequently all users required the highest level of clearance, which defeated the purpose of the system. Eventually, the BMA system was changed to a system using only compartments, which effectively solved the problem. Then, for example, AIDS prescription information could be compartmented from general prescription information, thereby enforcing the desired need to know principle.

In the next two sections we'll discuss covert channels and inference control. Both of these topics are related to MLS, but covert channels, in particular, arise in many different contexts.

8.6 Covert Channel

A *covert channel* is a communication path not intended as such by the system's designers. Covert channels exist in many situations, but they are particularly prevalent in networks. Covert channels are virtually impossible to eliminate, so the emphasis is instead on limiting the capacity of such channels.

MLS systems are designed to restrict legitimate channels of communication. But a covert channel provides another way for information to flow. It is not difficult to give an example where resources shared by subjects at different security levels can be used to pass information, and thereby violate the security of an MLS system.

For example, suppose Alice has a TOP SECRET clearance while Bob only has a CONFIDENTIAL clearance. If the file space is shared by all users, then Alice and Bob can agree that if Alice wants to send a 1 to Bob, she will create a file named, say, `FileXYzW`, and if she wants to send a 0 she will not create such a file. Bob can check to see whether file `FileXYzW` exists, and if it does, he knows Alice has sent him a 1, while if it does not, Alice has sent him a 0. In this way, a single bit of information has been passed through a covert channel, that is, through a means that was not intended for communication by the designers of the system. Note that Bob cannot look inside the file `FileXYzW` since he does not have the required clearance, but we are assuming that he can query the file system to see if such a file exists.

A single bit leaking from Alice to Bob is not a concern, but Alice could leak any amount of information by synchronizing with Bob. That is, Alice and Bob could agree that Bob will check for the file `FileXYzW` once each minute. As before, if the file does not exist, Alice has sent 0, and if it does exists, Alice has sent a 1. In this way Alice can (slowly) leak TOP SECRET information to Bob. This process is illustrated in Figure 8.5.

Covert channels are everywhere. For example, the print queue could be used to signal information in much the same way as in the example above. Networks are a rich source of covert channels, and several hacking tools exist that exploit these covert channels—we'll mention one later in this section.

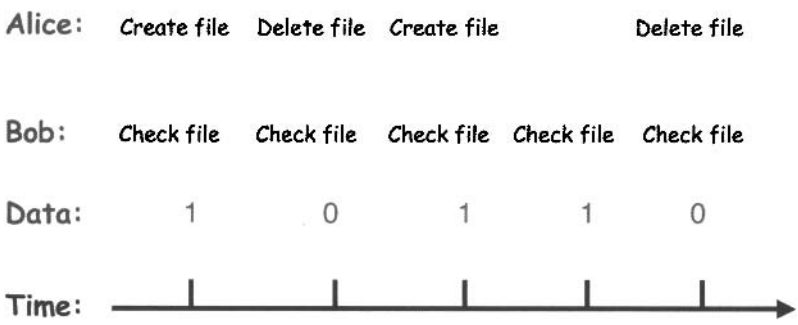

Figure 8.5: Covert Channel Example

Three things are required for a covert channel to exist. First, the sender and receiver must have access to a shared resource. Second, the sender must be able to vary some property of the shared resource that the receiver can observe. Finally, the sender and receiver must be able to synchronize their communication. From this description, it's apparent that potential covert channels really are everywhere. Of course, we can eliminate all covert channels—we just need to eliminate all shared resources and all communication. Obviously such a system would generally be of little use.

The conclusion here is that it's virtually impossible to eliminate all covert channels in any useful system. The DoD apparently agrees, since their guidelines merely call for reducing covert channel capacity to no more than one bit per second [131]. The implication is that DoD has given up trying to eliminate covert channels.

Is a limit of one bit per second sufficient to prevent damage from covert channels? Consider a TOP SECRET file that is 100 MB in size. Suppose the plaintext version of this file is stored in a TOP SECRET file system, while an encrypted version of the file—encrypted with, say, AES using a 256-bit key—is stored in an UNCLASSIFIED location. Following the DoD guidelines, suppose that we have reduced the covert channel capacity of this system to 1 bit per second. Then it would take more than 25 years to leak the entire 100 MB TOP SECRET document through a covert channel. However, it would take less than 5 minutes to leak the 256-bit AES key through the same covert channel. The conclusion is that reducing covert channel capacity might be useful, but it will not be sufficient in all cases.

Next, we consider a real-world example of a covert channel. The Transmission Control Protocol (TCP) is widely used on the Internet. The TCP header, which appears in the Appendix in Figure A-3, includes a "reserved" field which is reserved for future use, that is, it is not used for anything. This field can easily be used to pass information covertly.

It's also easy to hide information in the TCP sequence number or ACK field and thereby create a more subtle covert channel. Figure 8.6 illustrates the method used by the tool `Covert_TCP` to pass information in the sequence number. The sender hides the information in the sequence number X and the packet—with its source address forged to be the address of the intended recipient—is sent to an innocent server. When the server acknowledges the packet, it unwittingly completes the covert channel by passing the information contained in X to the intended recipient. Such stealthy covert channels are often employed in network attacks [270].

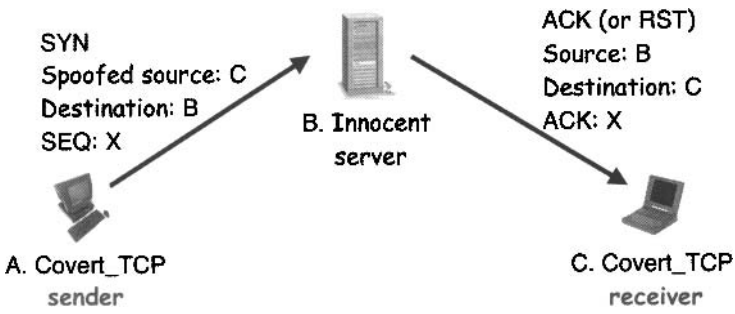

Figure 8.6: Covert Channel Using TCP Sequence Number

8.7 Inference Control

Consider a database that includes information on college faculty in California. Suppose we query the database and ask for the average salary of female computer science professors at San Jose State University (SJSU) and we find the answer is $100,000. We then query the database and ask for the number of female computer science professors at SJSU, and the answer is one. Then we could go to the SJSU computer science department website and determine the identity of this person.[8] In this example, specific information has leaked from responses to general questions. The goal of inference control is to prevent such leaks from happening, or at least minimize the leakage.

A database containing medical records would be of considerable interest to researchers. For example, by searching for statistical correlations, it may be possible to determine causes or risk factors for certain diseases. But patients want to keep their medical information private. How can we allow access to the statistically significant data while protecting privacy?

[8]In this case, no harm was done, since state employee salaries are public information in California.

An obvious first step is to remove names and addresses from the medical records. But this is not sufficient to ensure privacy as the college professor example above clearly demonstrates. What more can be done to provide stronger inference control while leaving the data accessible for legitimate research uses?

Several techniques used in inference control are discussed in [14]. One such technique is *query set size control*, in which no response is returned if the size of the set it too small. This approach would make it more difficult to determine the college professor's salary in the example above. However, if medical research is focused on a rare disease, query set size control could also prevent or distort important research.

Another technique is known as the *N-respondent, k% dominance rule*, whereby data is not released if $k\%$ or more of the result is contributed by N or fewer subjects. For example, we might query the census database and ask for the average net worth of individuals in Bill Gates' neighborhood. With any reasonable setting for N and k no results would be returned. In fact, this technique is actually applied to information collected by the United States Census Bureau.

Another approach to inference control is randomization, that is, a small amount of random noise is added to the data. This is problematic in situations such as research into rare medical conditions, where the noise might swamp legitimate data.

Many other methods of inference control have been proposed, but none are completely satisfactory. It appears that strong inference control may be impossible to achieve in practice, yet it seems obvious that employing some inference control, even if it's weak, is better than no inference control at all. Inference control will make Trudy's job more difficult, and it will almost certainly reduce the amount of information that leaks, thereby limiting the damage.

Does this same logic hold for crypto? That is, is it better to use weak encryption or no encryption at all? Surprisingly, for crypto, the answer is that, in most cases, you'd be better off not encrypting rather than using a weak cipher. Today, most information is not encrypted, and encryption tends to indicate important data. If there is a lot of data being sent and most of it is plaintext (e.g., email sent over the Internet), then Trudy faces an enormous challenge in attempting to filter interesting messages from this mass of uninteresting data. However, if your data is encrypted, it would be much easier to filter, since encrypted data looks random, whereas unencrypted data tends to be highly structured.[9] That is, if your encryption is weak, you may have just solved Trudy's difficult filtering problem for her, while providing no significant protection from a cryptanalytic attack [14].

[9]For one way around this problem, see [287].

8.8 CAPTCHA

The Turing test was proposed by computing pioneer (and breaker of the Enigma) Alan Turing in 1950. The test has a human ask questions to a human and a computer. The questioner, who can't see either the human or the computer, can only submit questions by typing on a keyboard, and responses are received on a computer screen. The questioner does not know which is the computer and which is the human, and the goal is to distinguish the human from the computer, based solely on the questions and answers. If the human questioner can't solve this puzzle with a probability better than guessing, the computer passes the Turing test. This test is the gold standard in artificial intelligence, and no computer has yet passed the Turing test, but occasionally some claim to be getting close.

A "completely automated public Turing test to tell computers and humans apart," or *CAPTCHA*,[10] is a test that a human can pass, but a computer can't pass with a probability better than guessing [319]. This could be considered as a sort of inverse Turing test. The assumptions here are that the test is generated by a computer program and graded by a computer program, yet no computer can pass the test, even if that computer has access to the source code used to generate the test. In other words, a "CAPTCHA is a program that can generate and grade tests that it itself cannot pass, much like some professors" [319].

At first blush, it seems paradoxical that a computer can create and score a test that it cannot pass. However, this becomes less of a paradox when we look more closely the details of the process.

Since CAPTCHAs are designed to prevent non-humans from accessing resources, a CAPTCHA can be viewed as a form of access control. According to folklore, the original motivation for CAPTCHAs was an online poll that asked users to vote for the best computer science graduate school. In this version of reality, it quickly become obvious that automated responses from MIT and Carnegie-Mellon were skewing the results [320] and researchers developed the idea of a CAPTCHA to prevent automated "bots" from stuffing the ballot box. Today, CAPTCHAs are used in a wide variety of applications. For example, free email services use CAPTCHAs to prevent spammers from automatically signing up for large numbers of email accounts.

The requirements for a CAPTCHA include that it must be easy for most humans to pass and it must be difficult or impossible for a machines to pass, even if the machine has access to the CAPTCHA software. From the attacker's perspective, the only unknown is some randomness that is used to generate the specific CAPTCHA. It is also desirable to have different types

[10]CAPTCHAs are also known as "human interactive proofs," or HIPs. While CAPTCHA may well rank as the worst acronym in the history of the universe, HIP is, well, just not hip.

of CAPTCHAs in case some person cannot pass one particular type. For example, many websites allow users to choose an audio CAPTCHA as an alternative to the usual visual CAPTCHA.

An example of a CAPTCHA from [320] appears in Figure 8.7. In this case, a human might be asked to find three words that appear in the image. This is a relatively easy problem for humans and today it is also a fairly easy problem for computers to solve—much stronger CAPTCHAs exist.

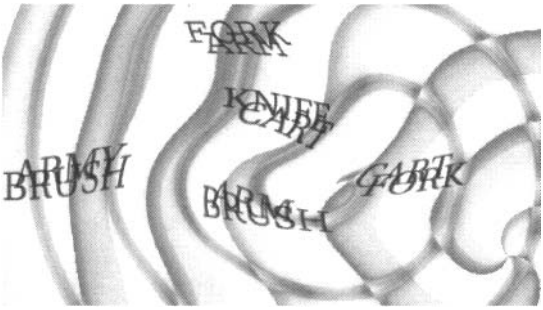

Figure 8.7: CAPTCHA (Courtesy of Luis von Ahn [320])

Perhaps surprisingly, in [56] it is shown that computers are actually better than humans at solving all of the fundamental visual CAPTCHA problems, with one exception—the so-called segmentation problem, i.e., the problem of separating the letters from each other. Consequently, strong CAPTCHAs tend to look more like Figure 8.8 than Figure 8.7.

Figure 8.8: A Strong CAPTCHA [47]

For a word-based visual CAPTCHA, we assume that Trudy knows the set of possible words that could appear and she knows the general format of the image, as well as the types of distortions that can be applied. From Trudy's perspective, the only unknown is a random number that is used to select the word or words and to distort the resulting image.

There are several types of visual CAPTCHAs of which Figures 8.7 and 8.8 are representative examples. There are also audio CAPTCHAs in which the audio is distorted in some way. The human ear is very good at removing such distortion, while automated methods are not so good. Currently, there are no text-based CAPTCHAs.

The computing problems that must be solved to break CAPTCHAs can be viewed as difficult problems from the domain of artificial intelligence, or AI.

For example, automatic recognition of distorted text is an AI problem, and the same is true of problems related to distorted audio. If attackers are able to break such CAPTCHAs, they have, in effect, solved a hard AI problem. As a result, attacker's efforts are being put to good use.

Of course, the attackers may not play by the rules—so-called CAPTCHA farming is possible, where humans are paid to solve CAPTCHAs. For example, it has been widely reported that the lure of free pornography has been successfully used to get humans to solve vast numbers of CAPTCHAs at minimal cost to the attacker [172].

8.9 Firewalls

Suppose you want to meet with the chairperson of your local computer science department. First, you will probably need to contact the computer science department secretary. If the secretary deems that a meeting is warranted, she will schedule it; otherwise, she will not. In this way, the secretary filters out many requests that would otherwise occupy the chair's time.

A *firewall* acts a lot like a secretary for your network. The firewall examines requests for access to your network, and it decides whether they pass a reasonableness test. If so, they are allowed through, and, if not, they are refused.

If you want to meet the chair of the computer science department, the secretary does a certain level of filtering; however, if you want to meet the President of the United States,[11] his secretary will perform a much different level of filtering. This is somewhat analogous to firewalls, where some simple firewalls only filter out obviously bogus requests and other types of firewalls make a much greater effort to filter anything suspicious.

A network firewall, as illustrated in Figure 8.9, is placed between the internal network, which might be considered relatively safe,[12] and the external network (the Internet), which is known to be unsafe. The job of the firewall is to determine what to let into and out of the internal network. In this way, a firewall provides access control for the network.

As with most of information security, for firewalls there is no standard terminology. But whatever you choose to call them, there are essentially three types of firewalls—marketing hype from firewall vendors not withstanding. Each type of firewall filters packets by examining the data up to a particular layer of the network protocol stack. If you are not familiar with networking (and even if you are), now would be a good time to review the networking material in the Appendix.

[11] POTUS, that is.

[12] This is almost certainly not a valid assumption. It's estimated that about 80% of all significant computer attacks are due to insiders [49].

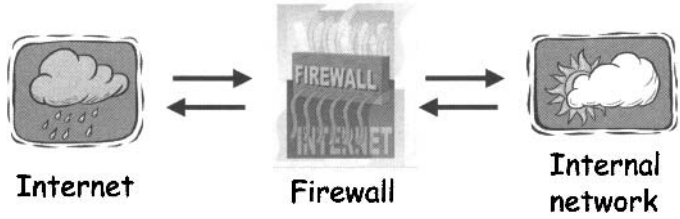

Figure 8.9: Firewall

We'll adopt the following terminology for the classification of firewalls.

- A *packet filter* is a firewall that operates at the network layer.
- A *stateful packet filter* is a firewall that lives at the transport layer.
- An *application proxy* is, as the name suggests, a firewall that operates at the application layer where it functions as a proxy.

8.9.1 Packet Filter

A packet filter firewall examines packets up to the network layer, as indicated in Figure 8.10. As a result, this type of firewall can only filter packets based on the information that is available at the network layer. The information at this layer includes the source IP address, the destination IP address, the source port, the destination port, and the TCP flag bits (SYN, ACK, RST, etc.).[13] Such a firewall can filter packets based on ingress or egress, that is, it can have different filtering rules for incoming and outgoing packets.

The primary advantage of a packet filter is efficiency. Since packets only need to be processed up to the network layer and only header information is examined, the entire operation is inherently efficient. However, there are several disadvantages to the simple approach employed by a packet filter. First, the firewall has no concept of state, so each packet is treated independently of all others. In particular, a packet filter can't examine a TCP connection. We'll see in a moment that this is a serious limitation. In addition, a packet filter firewall is blind to application data, which is where viruses and other malware resides.

Packet filters are configured using access control lists, or ACLs. In this context, "ACL" has a completely different meaning than in Section 8.3.1. An example of a packet filter ACL appears in Table 8.3. Note that the purpose of the ACL in Table 8.3 is to restrict incoming packets to Web responses,

[13]Yes, we're cheating. TCP is part of the transport layer, so the TCP flag bits are not visible if we follow a strict definition of network layer. Nevertheless, it's OK to cheat sometimes, especially in a security class.

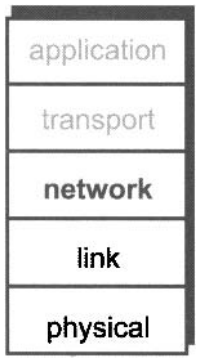

Figure 8.10: Packet Filter

Table 8.3: Example ACL

Action	Source IP	Dest IP	Source Port	Dest Port	Protocol	Flag Bits
Allow	Inside	Outside	Any	80	HTTP	Any
Allow	Outside	Inside	80	> 1023	HTTP	ACK
Deny	All	All	All	All	All	All

which should have source port 80. The ACL allows all outbound Web traffic, which should be destined to port 80. All other traffic is forbidden.

How might Trudy take advantage of the inherent limitations of a packet filter firewall? Before we can answer this question, we need a couple of fun facts. Usually, a firewall (of any type) drops packets sent to most incoming ports. That is, the firewall filters out and drops packets that are trying to access services that should not be accessed. Because of this, the attacker, Trudy, wants to know which ports are open through the firewall. These open ports are where Trudy will concentrate her attack. So, the first step in any attack on a firewall is usually a *port scan*, where Trudy tries to determine which ports are open through the firewall.

Now suppose Trudy wants to attack a network that is protected by a packet filter. How can Trudy conduct a port scan of the firewall? She could, for example, send a packet that has the ACK bit set, without the prior two steps of the TCP three-way handshake. Such a packet violates the TCP protocol, since the initial packet in any connection must have the SYN bit set. Since the packet filter has no concept of state, it will assume that this packet is part of an established connection and let it through—provided that

it is sent to an open port. Then when this forged packet reaches a host on the internal network, the host will realize that there is a problem (since the packet is not part of an established connection) and respond with a RST packet, which is supposed to tell the sender to terminate the connection. While this process may seem harmless, it allows Trudy to scan for open ports through the firewall. That is, Trudy can send an initial packet with the ACK flag set to a particular port p. If no response is received, then the firewall is not forwarding packets sent to port p. However, if a RST packet is received, then the packet was allowed through port p into the internal network. This technique, which is known as a TCP ACK scan, is illustrated in Figure 8.11.

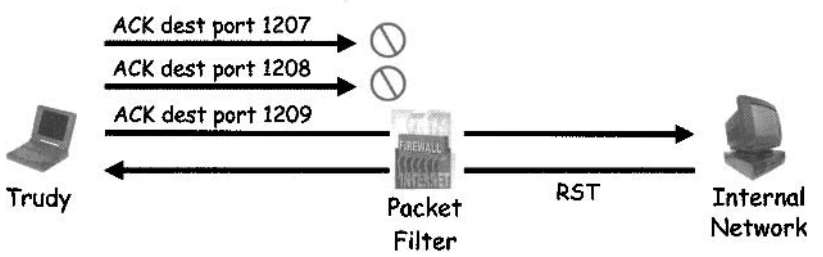

Figure 8.11: TCP ACK Scan

From the ACK scan in Figure 8.11, Trudy has learned that port 1209 is open through the firewall. To prevent this attack, the firewall would need to remember existing TCP connections, so that it will know that the ACK scan packets are not part of any legitimate connection. Next, we'll discuss stateful packet filters, which keep track of connections and are therefore able to prevent this ACK scan attack.

8.9.2 Stateful Packet Filter

As the name implies, a stateful packet filter adds state to a packet filter firewall. This means that the firewall keeps track of TCP connections, and it can remember UDP "connections" as well. Conceptually, a stateful packet filter operates at the transport layer, since it is maintaining information about connections. This is illustrated in Figure 8.12.

The primary advantage of a stateful packet filter is that, in addition to all of the features of a packet filter, it also keeps track of ongoing connection. This prevents many attacks, such as the TCP ACK scan discussed in the previous section. The disadvantages of a stateful packet filter are that it cannot examine application data, and, all else being equal, it's slower than a packet filtering firewall since more processing is required.

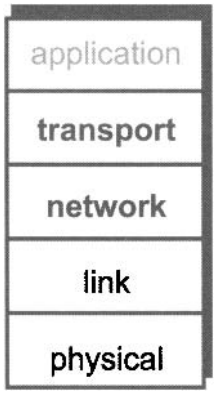

Figure 8.12: Stateful Packet Filter

8.9.3 Application Proxy

A proxy is something that acts on your behalf. An application proxy firewall processes incoming packets all the way up to the application layer, as indicated in Figure 8.13. The firewall, acting on your behalf, is then able to verify that the packet appears to be legitimate (as with a stateful packet filter) and, in addition, that the actual data inside the packet is safe.

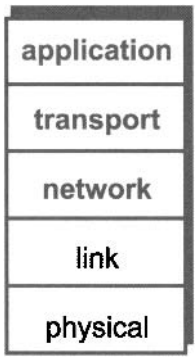

Figure 8.13: Application Proxy

The primary advantage of an application proxy is that it has a complete view of connections and application data. Consequently, it can have as comprehensive of a view as the host itself could have. As a result, the application proxy is able to filter bad data at the application layer (such as viruses) while also filtering bad packets at the transport layer. The disadvantage of an ap-

plication proxy is speed or, more precisely, the potential lack thereof. Since the firewall is processing packets to the application layer, examining the resulting data, maintaining state, etc., it is doing a great deal more work than packet filtering firewalls.

One interesting feature of an application proxy is that the incoming packet is destroyed and a new packet is created in its place when the data passes through the firewall. Although this might seem like a minor and insignificant point, it's actually a security feature. To see why creating a new packet is beneficial, we'll consider the tool known as `Firewalk`, which is designed to scan for open ports through a firewall. While the purpose of `Firewalk` is the same as the TCP ACK scan discussed above, the implementation is completely different.

The time to live, or TTL, field in an IP packet header contains the number of hops that the packet will travel before it is terminated. When a packet is terminated due to the TTL field, an ICMP "time exceeded" error message is sent back to the source.[14]

Suppose Trudy knows the IP address of the firewall, the IP address of one system on the inside network, and the number of hops to the firewall. Then she can send a packet to the IP address of the known host inside the firewall, with the TTL field set to one more than the number of hops to the firewall. Suppose Trudy sets the destination port of such a packet to p. If the firewall does not let data through on port p, there will be no response. If, on the other hand, the firewall does let data through on port p, Trudy will receive a time exceeded error message from the first router inside the firewall that receives the packet. Trudy can then repeat this process for different ports p to determine open ports through the firewall. This port scan is illustrated in Figure 8.14. `Firewalk` will succeed if the firewall is a packet filter or a stateful packet filter. However, `Firewalk` won't succeed if the firewall is an application proxy (see Problem 29).

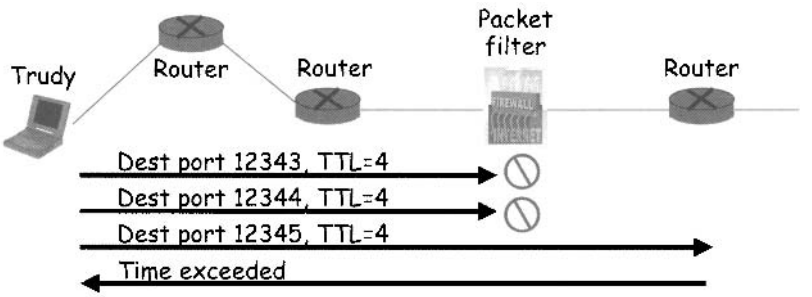

Figure 8.14: Firewalk

[14] And what happens to terminated packets? Of course, they die and go to packet heaven.

The net effect of an application proxy is that it forces Trudy to talk to the proxy and convince it to forward her messages. Since the proxy is likely to be well configured and carefully managed—compared with a typical host—this may prove difficult.

8.9.4 Personal Firewall

A personal firewall is used to protect a single host or a small network, such as a home network. Any of the three methods discussed above (packet filter, stateful packet filter, or application proxy) could be used, but generally such firewalls are relatively simple for the sake of efficiency and ease of configuration.

8.9.5 Defense in Depth

Finally, we consider a network configuration that includes several layers of protection. Figure 8.15 gives a schematic for a network that includes a packet filter firewall, an application proxy, and personal firewalls, as well as a demilitarized zone, or DMZ.

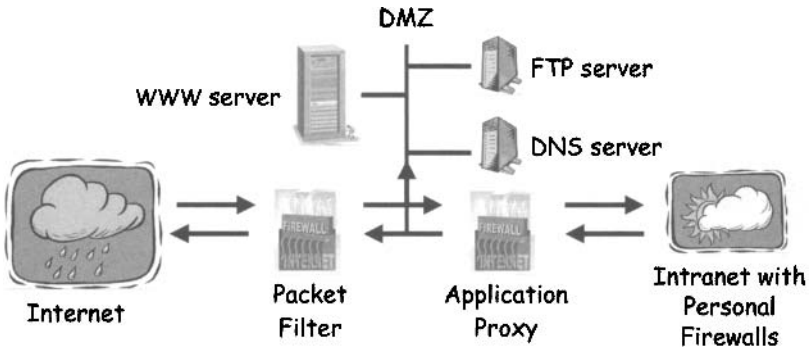

Figure 8.15: Defense in Depth

The packet filter in Figure 8.15 is used to prevent common attacks on the systems in the DMZ. The systems in the DMZ are those that must be exposed to the outside world. These systems receive most of the outside traffic, so a simple packet filter is used for the sake of efficiency. The systems in the DMZ must be carefully maintained by the administrator since they are the most exposed to attack. However, if an attack succeeds on a system in the DMZ, the consequences for the company are annoying, but they will probably not be life threatening, since the internal network is largely unaffected.

In Figure 8.15, an application proxy firewall sits between the internal network and the DMZ. This provides the strongest possible firewall protection

for the internal network. The amount of traffic into the internal network is likely to be relatively small, so an application proxy in this position will not create a bottleneck. As a final layer of protection, personal firewalls could be deployed on the individual hosts inside the corporate network.

The architecture in Figure 8.15 is an example of *defense in depth*, which is a good security strategy in general—if one layer of the defense is breached, there are more layers that the attacker must overcome. If Trudy is skilled enough to break through one level, then she may have the necessary skills to penetrate other levels. But it's likely to take her some time to do so and the longer it takes, the more time an administrator has to detect Trudy's attack in progress.

Regardless of the strength of the firewall (or firewalls), some attacks by outsiders will succeed. In addition, attacks by insiders are a serious threat and firewalls are of limited value against such attacks. In any case, when an attack succeeds, we would like to detect it as soon as possible. In the next section we'll discuss this intrusion detection problem.

8.10 Intrusion Detection Systems

The primary focus of computer security tends to be *intrusion prevention*, where the goal is to keep the Trudys of the world out of your system or network. Authentication can be viewed as a means to prevent intrusions, and firewalls are certainly a form of intrusion prevention, as are most types of virus protection. Intrusion prevention is the information security analog of locking the doors on your car.

But even if you lock the doors on your car, it might still get stolen. In information security, no matter how much effort you put into intrusion prevention, occasionally the bad guys will be successful and an intrusion will occur.

What should we do when intrusion prevention fails? *Intrusion detection systems*, or IDSs, are a relatively recent development in information security. The purpose of such a system is to detect attacks before, during, and after they occurr.

The basic approach employed by IDSs is to look for "unusual" activity. In the past, an administrator would scan through log files looking for signs of unusual activity—automated intrusion detection is a natural outgrowth of manual log file analysis.

It is also worth noting that intrusion detection is currently an active research topic. As with any relatively new technology, there are many claims in the field that have yet to be substantiated. At this point, it's far from clear how successful or useful some of these techniques will prove, particularly in the face of increasingly sophisticated attacks.

Before discussing the main threads in IDS, we mention in passing that *intrusion response* is a related topic of practical importance. That is, once an intrusion is detected, we want to respond to it. In some cases we obtain specific information and a reasonable response is fairly obvious. For example, we might detect a password guessing attack aimed at a specific account, in which case we could respond by locking the account. However, it's not always so straightforward. We'll see below that in some cases IDSs provide little specific information on the nature of an attack. In such cases, determining the proper response is not easy, since we may not be sure of the specifics of the attack. In any case, we won't deal further with intrusion response here.

Who are the intruders that an IDS is trying to detect? An intruder could be a hacker who got through your network defenses and is now launching an attack on the internal network. Or, even more insidious, the intrusion could be due to an evil insider, such as a disgruntled employee.

What sorts of attacks might an intruder launch? An intruder with limited skills (i.e., a "script kiddie") would likely attempt a well-known attack or a slight variation on such an attack. A more skilled attacker might be capable of launching a significant variation on a well-known attack, or a little-known attack or an entirely new attack. Often, the attacker will simply use the breached system as a base from which to launch attacks on other systems.

Broadly speaking, there are two approaches to intrusion detection.

- *Signature-based IDSs* detect attacks based on specific known signatures or patterns. This is analogous to signature-based virus detection, which we'll discuss in Chapter 11.

- *Anomaly-based IDSs* attempt to define a baseline of normal behavior and provide a warning whenever the system strays too far from this baseline.

We'll have more to say about signature-based and anomaly-based intrusion detection below.

There are also two basic architectures for IDSs.

- *Host-based IDSs* apply their detection method or methods to activity that occurs on hosts. These systems have the potential to detect attacks that are visible at hosts (e.g., buffer overflows or escalation of privilege). However, host-based systems have little or no view of network activities.

- *Network-based IDSs* apply their detection methods to network traffic. These systems are designed to detect attacks such as denial of service, port scans, probes involving malformed packets, etc. Such systems have some obvious overlap with firewalls. Network-based systems have little or no direct view of host-based attacks.

Of course, various combinations of these categories of IDSs are possible. For example a host-based system could use both signature-based and anomaly-based techniques, or a signature-based system might employ aspects of both host-based and network-based detection.

8.10.1 Signature-Based IDS

Failed login attempts may be indicative of a password cracking attack, so an IDS might consider "N failed login attempts in M seconds" an indication, or *signature*, of an attack. Then anytime that N or more failed login attempts occur within M seconds, the IDS would issue a warning that a password cracking attack is suspected to be in progress.

If Trudy happens to know that Alice's IDS issues a warning whenever N or more failed logins occur within M seconds, then Trudy can safely guess $N-1$ passwords every M seconds. In this case, the signature detection would slow Trudy's password guessing attack, but it would not completely prevent the attack. Another concern with such a scheme is that N and M must be set so that the number of false alarms is not excessive.

Many techniques are used to make signature-based detection more robust, where the usual approach is to detect "almost" signatures. For example, if about N login attempts occur in about M seconds, then the system could warn of a possible password cracking attack, perhaps with a degree of confidence based on the number of attempts and the time interval. But it's not always easy to determine reasonable values for "about." Statistical analysis and heuristics are useful, but much care must be taken to minimize the false alarm rate. False alarms will quickly undermine confidence in any security system—like the boy who cried wolf, the security system that screams "attack" when none is present, will soon be ignored.

The advantages of signature-based detection include simplicity, efficiency (provided the number of signatures is not excessive), and an excellent ability to detect known attacks. Another major benefit is that the warning that is issued is specific, since the signature matches a specific attack pattern. With a specific warning, an administrator can quickly determine whether the suspected attack is real or a false alarm and, if it is real, the admin can usually respond appropriately.

The disadvantages of signature detection include the fact that the signature file must be current, the number of signatures may become large thereby reducing efficiency, and most importantly, the system can only detect known attacks. Even slight variations on known attack will likely be missed by signature-based systems.

Anomaly-based IDSs attempt to overcome the shortcomings of signature-based schemes. But no anomaly-based scheme available today could reasonably claim to be a replacement for signature-based detection. That is, an

anomaly-based system can supplement the performance of a signature-based system, but it is not a replacement for signature detection.

8.10.2 Anomaly-Based IDS

Anomaly-based IDSs look for unusual or abnormal behavior. There are several major challenges inherent in such an approach. First, we must determine what constitutes normal behavior for a system, and this must occur when the system is behaving normally. Second, the definition of normal must adapt as system usage changes and evolves, otherwise the number of false alarms will grow. Third, there are difficult statistical thresholding issues involved. For example, we must have a good idea of how far abnormal is away from normal.

Statistics are obviously necessary in the development of an anomaly-based IDS. Recall that the *mean* defines the statistical norm while the *variance* gives us a way to measure the distribution of the data about the mean. The mean and variance together gives us a way to determine abnormal behavior.

How can we measure normal system behavior? Whatever characteristics we decide to measure, we must take the measurements during times of representative behavior. In particular, we must not set the baseline measurements during an attack or else an attack will be considered normal. Measuring abnormal or, more precisely, determining how to separate normal variations in behavior from an attack, is an equally challenging problem. Abnormal must be measured relative to some specific value of normal. We'll consider abnormal as synonymous with attack, although in reality there are other possible causes of abnormal behavior, which further complicates the situation.

Statistical discrimination techniques are used to separate normal from abnormal. Examples of such techniques include Bayesian analysis, linear discriminant analysis (LDA), quadratic discriminant analysis (QDA), neural nets, and hidden Markov models (HMM), among others. In addition, some anomaly detection researchers employ advanced modeling techniques from the fields of artificial intelligence and artificial immune systems. Such approaches are beyond the scope of our discussion here.

Next, we'll consider two simplified examples of anomaly detection. The first example is simple, but not very realistic, whereas the second is slightly less simple and correspondingly more realistic.

Suppose that we monitor the use of the three commands

```
open, read, close.
```

We find that under normal use, Alice uses the series of commands

```
open, read, close, open, open, read, close.
```

For our statistic, we'll consider pairs of consecutive commands and try to devise a measure of normal behavior for Alice. From Alice's series of com-

mands, we observe that, of the six possible ordered pairs or commands, four pairs appear to be normal for Alice, namely,

(`open,read`), (`read,close`), (`close,open`), (`open,open`),

while the other two pairs,

(`read,open`), (`close,read`),

are not normally used by Alice. We can use this observation to identify potentially unusual behavior by "Alice" that might indicate an intruder is posing as Alice. We can then monitor the use of these three commands by Alice. If the ratio of abnormal to normal pairs is "too high," we would warn the administrator that an attack may be in progress.

This simple anomaly detection scheme can be improved. For example, we could include the expected frequency of each normal pair in the calculation, and if the observed pairs differ significantly from the expected distribution, we would warn of a possible attack. We might also try to improve the anomaly detection by using more than two consecutive commands, or by including more commands, or by including other user behavior in the model, or by using a more sophisticated statistical discrimination technique.

For a slightly more plausible anomaly detection scheme, let's focus on file access. Suppose that, over an extended period of time, Alice has accessed four files, F_0, F_1, F_2, F_3, at the rates H_0, H_1, H_2, H_3, respectively, where the observed values of the H_i are given in Table 8.4.

Table 8.4: Alice's Initial File Access Rates

H_0	H_1	H_2	H_3
0.10	0.40	0.40	0.10

Now suppose that, over a recent time interval, Alice has accessed file F_i at the rate A_i, for $i = 0, 1, 2, 3$, as given in Table 8.5. Do Alice's recent file access rates represent normal use? To decide, we need some way to compare her long-term access rates to the current rates. To answer this question, we'll employ the statistic

$$S = (H_0 - A_0)^2 + (H_1 - A_1)^2 + (H_2 - A_2)^2 + (H_3 - A_3)^2, \tag{8.2}$$

where we define $S < 0.1$ as normal. In this example, we have

$$S = (0.1 - 0.1)^2 + (0.4 - 0.4)^2 + (0.4 - 0.3)^2 + (0.1 - 0.2)^2 = 0.02,$$

and we conclude that Alice's recent use is normal—at least according to this one statistic.

Table 8.5: Alice's Recent File Access Rates

A_0	A_1	A_2	A_3
0.10	0.40	0.30	0.20

Alice's file access rates can be expected to vary over time, and we need to account for this in our IDS. We'll do so by updating Alice's long-term history values H_i according to the formula

$$H_i = 0.2 \cdot A_i + 0.8 \cdot H_i \quad \text{for } i = 0, 1, 2, 3. \tag{8.3}$$

That is, we update the historical access rates based on a moving average that combines the previous values with the recently observed rates—the previous values are weighted at 80%, while the current values are weighted 20%. Using the data in Tables 8.4 and 8.5, we find that the updated values of H_0 and H_1 are unchanged, whereas

$$H_2 = 0.2 \cdot 0.3 + 0.8 \cdot 0.4 = 0.38 \quad \text{and} \quad H_3 = 0.2 \cdot 0.2 + 0.8 \cdot 0.1 = 0.12.$$

These updated values appear in Table 8.6.

Table 8.6: Alice's Updated File Access Rates

H_0	H_1	H_2	H_3
0.10	0.40	0.38	0.12

Suppose that over the next time interval Alice's measured access rates are those given in Table 8.7. Then we compute the statistic S using the values in Tables 8.6 and 8.7 and the formula in equation (8.2) to find

$$S = (0.1 - 0.1)^2 + (0.4 - 0.3)^2 + (0.38 - 0.3)^2 + (0.12 - 0.3)^2 = 0.0488.$$

Since $S = 0.0488 < 0.1$ we again conclude that this is normal use for Alice. Again, we update Alice's long-term averages using the formula in (8.3) and

Table 8.7: Alice's More Recent File Access Rates

A_0	A_1	A_2	A_3
0.10	0.30	0.30	0.30

Table 8.8: Alice's Second Updated Access Rates

H_0	H_1	H_2	H_3
0.10	0.38	0.364	0.156

the data in Tables 8.6 and 8.7. In this case, we obtain the results that appear in Table 8.8.

Comparing Alice's long-term file access rates in Table 8.4 with her long-term averages after two updates, as given in Table 8.8, we see that the rates have changed significantly over time. Again, it is necessary that an anomaly-based IDS adapts over time, otherwise we will have a large number of false alarms (and a very annoyed system administrator) as Alice's actual behavior changes. However, this also presents an opportunity for the attacker, Trudy.

Since the H_i values slowly evolve to match Alice's behavior, Trudy can pose as Alice and remain undetected, provided she doesn't stray too far from Alice's usual behavior. But even more worrisome is the fact that Trudy can eventually convince the anomaly detection algorithm that her evil behavior is normal for Alice, provided Trudy has enough patience. For example, suppose that Trudy, posing as Alice, wants to always access file F_3. Then, initially, she can access file F_3 at a slightly higher rate than is normal for Alice. After the next update of the H_i values, Trudy will be able to access file F_3 at an even higher rate without triggering a warning from the anomaly detection software, and so on. By going slowly, Trudy will eventually convince the anomaly detector that it's normal for "Alice" to only access file F_3.

Note that $H_3 = 0.1$ in Table 8.4 and, two iterations later, $H_3 = 0.156$ in Table 8.8. These changes did not trigger a warning by the anomaly detector. Does this change represent a new usage pattern by Alice, or does it indicate an attempt by Trudy to trick the anomaly detector by going slow?

To make this anomaly detection scheme more robust, we should also incorporate the variance. In addition, we would certainly need to measure more than one statistic. If we measured N different statistics, $S_1, S_2, \ldots S_N$, we might combine them according to a formula such as

$$T = (S_1 + S_2 + S_3 + \ldots + S_N)/N$$

and make the determination of normal or abnormal based on the statistic T. This would provide a more comprehensive view of normal behavior and make it more difficult for Trudy, as she would need to approximate more of Alice's normal behavior. A similar—although much more sophisticated—approach is used in a popular IDS known as NIDES [9, 155]. NIDES incorporates both anomaly-based and signature-based IDSs. A good elementary introduction to NIDES, as well as several other IDSs, can be found in [304].

Robust anomaly detection is a difficult problem for a number of reasons. For one, system usage and user behavior constantly evolves and, therefore, so must the anomaly detector. Without allowing for such changes in behavior, false alarms would soon overwhelm the administrator, who would quickly lose confidence in the system. But an evolving anomaly detector means that it's possible for Trudy to slowly convince the anomaly detector that an attack is normal.

Another fundamental issue with anomaly detection is that a warning of abnormal behavior may not provide any useful specific information to the administrator. A vague warning that the system may be under attack could make it difficult to take concrete action. In contrast, a signature-based IDS will provide the administrator with precise information about the nature of the suspected attack.

The primary potential advantage of anomaly detection is that there is a chance of detecting previously unknown attacks. It's also sometimes argued that anomaly detection can be more efficient than signature detection, particularly if the signature file is large. In any case, the current generation of anomaly detectors must be used in combination with a signature-based IDS since they are not sufficiently robust to act as standalone systems.

Anomaly-based intrusion detection is an active research topic, and many security professionals have high hopes for its ultimate success. Anomaly detection is often cited as key future security technology [120]. But it appears that the hackers are not convinced, at least based on the title of a talk presented at a recent Defcon[15] conference: "Why anomaly-based intrusion detection systems are a hacker's best friend" [79].

The bottom line is that anomaly detection is a difficult and tricky problem. It also appears to have parallels with the field of artificial intelligence. Nearly a third of a century has passed since we were promised "robots on your doorstep" [327] and such predictions appear no more plausible today than at the time they were originally made. If anomaly-based intrusion detection proves to be anywhere near as challenging as AI, it may never live up to its claimed potential.

8.11 Summary

In this chapter we reviewed some of the history of authorization, with the focus on certification regimes. Then we covered the basics of traditional authorization, namely, Lampson's access control matrix, ACLs, and capabilities. The confused deputy problem was used to highlight the differences between ACLs and capabilities. We then presented some of the security issues re-

[15]Defcon is the oldest, largest, and best-known hackers convention. It's held in Las Vegas each August, and it's inexpensive, totally chaotic, lots of fun, and hot (literally).

lated to multilevel security (MLS) and compartments, as well as the topics of covert channels and inference control. MLS naturally led us into the rarified air of security modeling, where we briefly considered Bell-LaPadula and Biba's Model.

After covering the basics of security modeling, we pulled our heads out of the clouds, put our feet back on *terra firma*, and proceeded to discuss a few important non-traditional access control topics, including CAPTCHAs and firewalls. We concluded the chapter by stretching the definition of access control to cover intrusion detection systems (IDS). Many of the issues we discussed with respect to IDSs will resurface when we cover virus detection in Chapter 11.

8.12 Problems

1. On page 269 there is an example of orange book guidelines for testing at the so-called C division. Your skeptical author implies that these guidelines are somewhat dubious.

   a. Why might the guidelines that appear on page 269 not be particularly sensible or useful?

   b. Find three more examples of useless guidelines that appear in Part II of the orange book [309]. For each of these, summarize the guideline and give reasons why you feel it is not particularly sensible or useful.

2. The seven Common Criteria EALs are listed in Section 8.2.2. For each of these seven levels, summarize the testing required to achieve that level of certification.

3. In this chapter we discussed access control lists (ACLs) and capabilities (aka C-lists).

   a. Give two advantaged of capabilities over ACLs.

   b. Give two advantages of ACLs over capabilities.

4. In the text, we argued that it's easy to delegate using capabilities.

   a. It is also possible to delegate using ACLs. Explain how this would work.

   b. Suppose Alice delegates to Bill who then delegates to Charlie who, in turn, delegates to Dave. How would this be accomplished using capabilities? How would this be accomplished using ACLs? Which is easier and why?

   c. Which is better for delegation, ACLs or capabilities? Why?

5. Suppose Alice wants to temporarily delegate her C-list (capabilities) to Bob. Alice decides that she will digitally sign her C-list before giving it to Bob.

   a. What are the advantages, if any, of such an approach?

   b. What are the disadvantages, if any, of such an approach?

6. Briefly discuss one real-world application not mentioned in the text where multilevel security (MLS) would be useful.

7. What is the "need to know" principle and how can compartments be used to enforce this principle?

8. Suppose that you work in a classified environment where MLS is employed and you have a TOP SECRET clearance.

   a. Describe a potential covert channel involving the User Datagram Protocol (UDP).

   b. How could you minimize your covert channel in part a, while still allowing network access and communication by users with different clearances?

9. The *high water mark principle* and *low water mark principle* both apply in the realm of multilevel security.

   a. Define the high water mark principle and the low water mark principle in the context of MLS.

   b. Is BLP consistent with a high water mark principle, a low water mark principle, both, or neither? Justify your answer.

   c. Is Biba's Model consistent with a high water mark principle, a low water mark principle, both, or neither? Justify your answer.

10. This problem deals with covert channels.

    a. Describe a covert channel involving the print queue and estimate the realistic capacity of your covert channel.

    b. Describe a subtle covert channel involving the TCP network protocol.

11. We briefly discussed the following methods of inference control: query set size control; N-respondent, $k\%$ dominance rule; and randomization.

    a. Explain each of these three methods of inference control.

    b. Briefly discuss the relative strengths and weaknesses of each of these methods.

12. Inference control is used to reduce the amount of private information that can leak as a result of database queries.

    a. Discuss one practical method of inference control not mentioned in the book.

    b. How could you attack the method of inference control given in your solution to part a?

13. A *botnet* consists of a number of compromised machines that are all controlled by an evil botmaster [39, 146].

    a. Most botnets are controlled using the Internet Relay Chat (IRC) protocol. What is IRC and why is it particularly useful for controlling a botnet?

    b. Why might a covert channel be useful for controlling a botnet?

    c. Design a covert channel that could provide a reasonable means for a botmaster to control a botnet.

14. Read and briefly summarize each of the following sections from the article on covert channels at [131]: 2.2, 3.2, 3.3, 4.1, 4.2, 5.2, 5.3, 5.4.

15. Ross Anderson claims that "Some kinds of security mechanisms may be worse than useless if they can be compromised" [14].

    a. Does this statement hold true for inference control? Why or why not?

    b. Does this hold true for encryption? Why or why not?

    c. Does this hold true for methods that are used to reduce the capacity of covert channels? Why or why not?

16. Combine BLP and Biba's Model into a single MLS security model that covers both confidentiality and integrity.

17. BLP can be stated as "no read up, no write down." What is the analogous statement for Biba's Model?

18. Consider the visual CAPTCHA known as Gimpy [249].

    a. Explain how EZ Gimpy and Hard Gimpy work.

    b. How secure is EZ Gimpy compared to Hard Gimpy?

    c. Discuss the most successful known attack on each type of Gimpy.

19. This problem deals with visual CAPTCHAs.

a. Describe an example of a real-world visual CAPTCHA not discussed in the text and explain how this CAPTCHA works, that is, explain how a program would generate the CAPTCHA and score the result, and what a human would need to do to pass the test.

b. For the CAPTCHA in part a, what information is available to an attacker?

20. Design and implement your own visual CAPTCHA. Outline possible attacks on your CAPTCHA. How secure is your CAPTCHA?

21. This problem deals with audio CAPTCHAs.

a. Describe an example of a real-world audio CAPTCHA and explain how this CAPTCHA works, that is, explain how a program would generate the CAPTCHA and score the result, and what a human would need to do to pass the test.

b. For the CAPTCHA in part a, what information is available to an attacker?

22. Design and implement your own audio CAPTCHA. Outline possible attacks on your CAPTCHA. How secure is your CAPTCHA?

23. In [56] it is shown that computers are better than humans at solving all of the fundamental visual CAPTCHA problems, with the exception of the segmentation problem.

a. What are the fundamental visual CAPTCHA problems?

b. With the exception of the segmentation problem, how can computers solve each of these fundamental problems?

c. Intuitively, why is the segmentation problem more difficult for computers to solve?

24. The reCAPTCHA project is an attempt to make good use of the effort humans put into solving CAPTCHAs [322]. In reCAPTCHA, a user is shown two distorted words, where one of the words is an actual CAPTCHA, but the other is a word—distorted to look like a CAPTCHA—that an optical character recognition (OCR) program was unable to recognize. If the real CAPTCHA is solved correctly, then the reCAPTCHA program assumes that the other word was also solved correctly. Since humans are good at correcting OCR errors, reCAPTCHA can be used, for example, to improve the accuracy of digitized books.

a. It is estimated that about 200,000,000 CAPTCHAs are solved daily. Suppose that each of these is a reCAPTCHA and each requires about 10 seconds to solve. Then, in total, about how

much time would be spent by users solving OCR problems each day? Note that we assume two CAPTCHAs are solved for one reCAPTCHA, so 200,000,000 CAPTCHAs represents 100,000,000 reCAPTCHAs.

b. Suppose that when digitizing a book, on average, about 10 hours of human effort is required to fix OCR problems. Under the assumptions in part a, how long would it take to correct all of the OCR problems created when digitizing all books in the Library of Congress? The Library of Congress has about 32,000,000 books, and we assume that every CAPTCHA in the world is a reCAPTCHA focused on this specific problem.

c. How could Trudy attack a reCAPTCHA system? That is, what could Trudy do to make the results obtained from a reCAPTCHA less reliable?

d. What could the reCAPTCHA developer do to minimize the effect of attacks on the system?

25. It has been widely reported that spammers sometimes pay humans to solve CAPTCHAs [293].

a. Why would spammers want to solve lots of CAPTCHAs?

b. What is the current cost, per CAPTCHA solved (in U.S. dollars), to have humans solve CAPTCHAs?

c. How might you entice humans to solve CAPTCHAs for you without paying them any money?

26. In this chapter, we discussed three types of firewalls: packet filter, stateful packet filter, and application proxy.

a. At which layer of the Internet protocol stack does each of these firewalls operate?

b. What information is available to each of these firewalls?

c. Briefly discuss one practical attack on each of these firewalls.

27. Commercial firewalls do not generally use the terminology packet filter, stateful packet filter, or application proxy. However, any firewall must be one of these three types, or a combination thereof. Find information on a commercial firewall product and explain (using the terminology of this chapter) which type of firewall it really is.

28. If a packet filter firewall does not allow reset (RST) packets out, then the TCP ACK scan described in the text will not succeed.

a. What are some drawbacks to this approach?

b. Could the TCP ACK scan attack be modified to work against such a system?

29. In this chapter it's stated that `Firewalk`, a port scanning tool, will succeed if the firewall is a packet filter or a stateful packet filter, but it will fail if the firewall is an application proxy.

    a. Why is this the case? That is, why does `Firewalk` succeed when the firewall is a packet filter or stateful packet filter, but fail when the firewall is an application proxy?

    b. Can `Firewalk` be modified to work against an application proxy?

30. Suppose that a packet filter firewall resets the TTL field to 255 for each packet that it allows through the firewall. Then the `Firewalk` port scanning tool described in the this chapter will fail.

    a. Why does `Firewalk` fail in this case?

    b. Does this proposed solution create any problems?

    c. Could `Firewalk` be modified to work against such a firewall?

31. An application proxy firewall is able to scan all incoming application data for viruses. It would be more efficient to have each host scan the application data it receives for viruses, since this would effectively distribute the workload among the hosts. Why might it still be preferable to have the application proxy perform this function?

32. Suppose incoming packets are encrypted with a symmetric key that is known only to the sender and the intended recipient. Which types of firewall (packet filter, stateful packet filter, application proxy) will work with such packets and which will not? Justify your answers.

33. Suppose that packets sent between Alice and Bob are encrypted and integrity protected by Alice and Bob with a symmetric key known only to Alice and Bob.

    a. Which fields of the IP header can be encrypted and which cannot?

    b. Which fields of the IP header can be integrity protected and which cannot?

    c. Which of the firewalls—packet filter, stateful packet filter, application proxy—will work in this case, assuming all IP header fields that can be integrity protected are integrity protected, and all IP header fields that can be encrypted are encrypted? Justify your answer.

34. Suppose that packets sent between Alice and Bob are encrypted and integrity protected by Alice's firewall and Bob's firewall with a symmetric key known only to Alice's firewall and Bob's firewall.

    a. Which fields of the IP header can be encrypted and which cannot?

    b. Which fields of the IP header can be integrity protected and which cannot?

    c. Which of the firewalls—packet filter, stateful packet filter, application proxy—will work in this case, assuming all IP header fields that can be integrity protected are integrity protected, and all IP header fields that can be encrypted are encrypted? Justify your answer.

35. Defense in depth using firewalls is illustrated in Figure 8.15. List other security applications where defense in depth is a sensible strategy.

36. Broadly speaking, there are two distinct types of intrusion detection systems, namely, signature-based and anomaly-based.

    a. List the advantages of signature-based intrusion detection, as compared to anomaly-based intrusion detection.

    b. List the advantages of an anomaly-based IDS, in contrast to a signature-based IDS.

    c. Why is effective anomaly-based IDS inherently more challenging than signature-based detection?

37. A particular vendor uses the following approach to intrusion detection.[16] The company maintains a large number of honeypots distributed across the Internet. To a potential attacker, these honeypots look like vulnerable systems. Consequently, the honeypots attract many attacks and, in particular, new attacks tend to show up on the honeypots soon after—sometimes even during—their development. Whenever a new attack is detected at one of the honeypots, the vendor immediately develops a signature and distributes the resulting signature to all systems using its product. The actual derivation of the signature is generally a manual process.

    a. What are the advantages, if any, of this approach as compared to a standard signature-based system?

    b. What are the advantages, if any, of this approach as compared to a standard anomaly-based system?

[16]This problem is based on a true story, just like many Hollywood movies...

c. Using the terminology given in this chapter, the system outlined in this problem would be classified as a signature-based IDS, not an anomaly-based IDS. Why?

d. The definition of signature-based and anomaly-based IDS are not standardized.[17] The vendor of the system outlined in this problem refers to it as an anomaly-based IDS. Why might they insist on calling it an anomaly-based IDS, when your well-nigh infallible author would classify it as a signature-based system?

38. The anomaly-based intrusion detection example presented in this chapter is based on file-use statistics.

a. Many other statistics could be used as part of an anomaly-based IDS. For example, network usage would be a sensible statistic to consider. List five other statistics that could reasonably be used in an anomaly-based IDS.

b. Why might it be a good idea to combine several statistics rather than relying on just a few?

c. Why might it not be a good idea to combine several statistics rather than relying on just a few?

39. Recall that the anomaly-based IDS example presented in this chapter is based on file-use statistics. The expected file use percentages (the H_i values in Table 8.4) are periodically updated using equation (8.3), which can be viewed as a moving average.

a. Why is it necessary to update the expected file use percentages?

b. When we update the expected file use percentages, it creates a potential avenue of attack for Trudy. How and why is this the case?

c. Discuss a different generic approach to constructing and updating an anomaly-based IDS.

40. Suppose that at the time interval following the results in Table 8.8, Alice's file-use statistics are given by $A_0 = 0.05$, $A_1 = 0.25$, $A_2 = 0.25$, and $A_3 = 0.45$.

a. Is this normal for Alice?

[17] Lack of standard terminology is a problem throughout most of the fields in information security (crypto being one of the few exceptions). It's important to be aware of this situation, since differing definitions is a common source of confusion. Of course, this problem is not unique to information security—differing definitions also cause confusion in many other fields of human endeavor. For proof, ask any two randomly selected economists about the current state of the economy.

b. Compute the updated values of H_0 through H_3.

41. Suppose that we begin with the values of H_0 through H_3 that appear in Table 8.4.

    a. What is the minimum number of iterations required until it is possible to have $H_2 > 0.9$ without the IDS triggering a warning at any step?

    b. What is the minimum number of iterations required until it is possible to have $H_3 > 0.9$ without the IDS triggering a warning at any step?

42. Consider the results given in Table 8.6.

    a. For the subsequent time interval, what is the largest possible value for A_3 that will not trigger a warning from the IDS?

    b. Give values for A_0, A_1, and A_2 that are compatible with the solution to part a.

    c. Compute the statistic S, using the solutions from parts a and b, and the H_i values in Table 8.6.