# PART 6: SYSTEM SECURITY

# CHAPTER 20

# INTRUDERS

*They agreed that Graham should set the test for Charles Mabledene. It was neither more nor less than that Dragon should get Stern's code. If he had the 'in' at Utting which he claimed to have this should be possible, only loyalty to Moscow Centre would prevent it. If he got the key to the code he would prove his loyalty to London Central beyond a doubt.*

—*Talking to Strange Men*, Ruth Rendell

## KEY POINTS

- Unauthorized intrusion into a computer system or network is one of the most serious threats to computer security.
- Intrusion detection systems have been developed to provide early warning of an intrusion so that defensive action can be taken to prevent or minimize damage.
- Intrusion detection involves detecting unusual patterns of activity or patterns of activity that are known to correlate with intrusions.
- One important element of intrusion prevention is password management, with the goal of preventing unauthorized users from having access to the passwords of others.

A significant security problem for networked systems is hostile, or at least unwanted, trespass by users or software. User trespass can take the form of unauthorized logon to a machine or, in the case of an authorized user, acquisition of privileges or performance of actions beyond those that have been authorized. Software trespass can take the form of a virus, worm, or Trojan horse.

All these attacks relate to network security because system entry can be achieved by means of a network. However, these attacks are not confined to network-based attacks. A user with access to a local terminal may attempt trespass without using an intermediate network. A virus or Trojan horse may be introduced into a system by means of an optical disc. Only the worm is a uniquely network phenomenon. Thus, system trespass is an area in which the concerns of network security and computer security overlap.

Because the focus of this book is network security, we do not attempt a comprehensive analysis of either the attacks or the security countermeasures related to system trespass. Instead, in this Part we present a broad overview of these concerns.

This chapter covers the subject of intruders. First, we examine the nature of the attack and then look at strategies intended for prevention and, failing that, detection. Next we examine the related topic of password management.

## 20.1 INTRUDERS

One of the two most publicized threats to security is the intruder (the other is viruses), often referred to as a hacker or cracker. In an important early study of intrusion, Anderson [ANDE80] identified three classes of intruders:

- **Masquerader:** An individual who is not authorized to use the computer and who penetrates a system's access controls to exploit a legitimate user's account
- **Misfeasor:** A legitimate user who accesses data, programs, or resources for which such access is not authorized, or who is authorized for such access but misuses his or her privileges
- **Clandestine user:** An individual who seizes supervisory control of the system and uses this control to evade auditing and access controls or to suppress audit collection

The masquerader is likely to be an outsider; the misfeasor generally is an insider; and the clandestine user can be either an outsider or an insider.

Intruder attacks range from the benign to the serious. At the benign end of the scale, there are many people who simply wish to explore internets and see what is out there. At the serious end are individuals who are attempting to read privileged data, perform unauthorized modifications to data, or disrupt the system.

[GRAN04] lists the following examples of intrusion:

- Performing a remote root compromise of an e-mail server
- Defacing a Web server
- Guessing and cracking passwords
- Copying a database containing credit card numbers
- Viewing sensitive data, including payroll records and medical information, without authorization
- Running a packet sniffer on a workstation to capture usernames and passwords
- Using a permission error on an anonymous FTP server to distribute pirated software and music files
- Dialing into an unsecured modem and gaining internal network access
- Posing as an executive, calling the help desk, resetting the executive's e-mail password, and learning the new password
- Using an unattended, logged-in workstation without permission

### Intruder Behavior Patterns

The techniques and behavior patterns of intruders are constantly shifting, to exploit newly discovered weaknesses and to evade detection and countermeasures. Even so, intruders typically follow one of a number of recognizable behavior patterns, and these patterns typically differ from those of ordinary users. In the following, we look at three broad examples of intruder behavior patterns, to give the reader some feel for the challenge facing the security administrator. Table 20.1, based on [RADC04], summarizes the behavior.

*HACKERS* Traditionally, those who hack into computers do so for the thrill of it or for status. The hacking community is a strong meritocracy in which status is determined by level of competence. Thus, attackers often look for targets of opportunity and then share the information with others. A typical example is a break-in at a large financial institution reported in [RADC04]. The intruder took advantage of the fact that the corporate network was running unprotected services, some of which were not even needed. In this case, the key to the break-in was the pcAnywhere application. The manufacturer, Symantec, advertises this program as a remote control solution that enables secure connection to remote devices. But the attacker had an easy time gaining access to pcAnywhere; the administrator used the same three-letter username and password for the program. In this case, there was no intrusion detection system on the 700-node corporate network. The intruder was only discovered when a vice president walked into her office and saw the cursor moving files around on her Windows workstation.

**Table 20.1** Some Examples of Intruder Patterns of Behavior

**(a) Hacker**

1. Select the target using IP lookup tools such as NSLookup, Dig, and others.
2. Map network for accessible services using tools such as NMAP.
3. Identify potentially vulnerable services (in this case, pcAnywhere).
4. Brute force (guess) pcAnywhere password.
5. Install remote administration tool called DameWare.
6. Wait for administrator to log on and capture his password.
7. Use that password to access remainder of network.

**(b) Criminal Enterprise**

1. Act quickly and precisely to make their activities harder to detect.
2. Exploit perimeter through vulnerable ports.
3. Use Trojan horses (hidden software) to leave back doors for reentry.
4. Use sniffers to capture passwords.
5. Do not stick around until noticed.
6. Make few or no mistakes.

**(c) Internal Threat**

1. Create network accounts for themselves and their friends.
2. Access accounts and applications they wouldn't normally use for their daily jobs.
3. E-mail former and prospective employers.
4. Conduct furtive instant-messaging chats.
5. Visit Web sites that cater to disgruntled employees, such as f'dcompany.com.
6. Perform large downloads and file copying.
7. Access the network during off hours.

Benign intruders might be tolerable, although they do consume resources and may slow performance for legitimate users. However, there is no way in advance to know whether an intruder will be benign or malign. Consequently, even for systems with no particularly sensitive resources, there is a motivation to control this problem.

Intrusion detection systems (IDSs) and intrusion prevention systems (IPSs) are designed to counter this type of hacker threat. In addition to using such systems, organizations can consider restricting remote logons to specific IP addresses and/or use virtual private network technology.

One of the results of the growing awareness of the intruder problem has been the establishment of a number of computer emergency response teams (CERTs). These cooperative ventures collect information about system vulnerabilities and disseminate it to systems managers. Hackers also routinely read CERT reports. Thus, it is important for system administrators to quickly insert all software patches to discovered vulnerabilities. Unfortunately, given the complexity of many IT systems, and the rate at which patches are released, this is increasingly difficult to achieve without automated updating. Even then, there are problems caused by incompatibilities resulting from the updated software. Hence the need for multiple layers of defense in managing security threats to IT systems.

*Criminals* Organized groups of hackers have become a widespread and common threat to Internet-based systems. These groups can be in the employ of a corporation or government but often are loosely affiliated gangs of hackers. Typically, these gangs are young, often Eastern European, Russian, or southeast Asian hackers who do business on the Web [ANTE06]. They meet in underground forums with names like DarkMarket.org and theftservices.com to trade tips and data and coordinate attacks. A common target is a credit card file at an e-commerce server. Attackers attempt to gain root access. The card numbers are used by organized crime gangs to purchase expensive items and are then posted to carder sites, where others can access and use the account numbers; this obscures usage patterns and complicates investigation.

Whereas traditional hackers look for targets of opportunity, criminal hackers usually have specific targets, or at least classes of targets in mind. Once a site is penetrated, the attacker acts quickly, scooping up as much valuable information as possible and exiting.

IDSs and IPSs can also be used for these types of attackers, but may be less effective because of the quick in-and-out nature of the attack. For e-commerce sites, database encryption should be used for sensitive customer information, especially credit cards. For hosted e-commerce sites (provided by an outsider service), the e-commerce organization should make use of a dedicated server (not used to support multiple customers) and closely monitor the provider's security services.

*Insider Attacks* Insider attacks are among the most difficult to detect and prevent. Employees already have access and knowledge about the structure and content of corporate databases. Insider attacks can be motivated by revenge or simply a feeling of entitlement. An example of the former is the case of Kenneth Patterson, fired from his position as data communications manager for American Eagle Outfitters. Patterson disabled the company's ability to process credit card purchases during five days of the holiday season of 2002. As for a sense of entitlement, there have always been many employees who felt entitled to take extra office supplies for home use, but this now extends to corporate data. An example is that of a vice president of sales for a stock analysis firm who quit to go to a competitor. Before she left, she copied the customer database to take with her. The offender reported feeling no animus toward her former employee; she simply wanted the data because it would be useful to her.

Although IDS and IPS facilities can be useful in countering insider attacks, other more direct approaches are of higher priority. Examples include the following:

- Enforce least privilege, only allowing access to the resources employees need to do their job.
- Set logs to see what users access and what commands they are entering.
- Protect sensitive resources with strong authentication.
- Upon termination, delete employee's computer and network access.
- Upon termination, make a mirror image of employee's hard drive before reissuing it. That evidence might be needed if your company information turns up at a competitor.

In this section, we look at the techniques used for intrusion. Then we examine ways to detect intrusion.

## Intrusion Techniques

The objective of the intruder is to gain access to a system or to increase the range of privileges accessible on a system. Most initial attacks use system or software vulnerabilities that allow a user to execute code that opens a back door into the system. Alternatively, the intruder attempts to acquire information that should have been protected. In some cases, this information is in the form of a user password. With knowledge of some other user's password, an intruder can log in to a system and exercise all the privileges accorded to the legitimate user.

Typically, a system must maintain a file that associates a password with each authorized user. If such a file is stored with no protection, then it is an easy matter to gain access to it and learn passwords. The password file can be protected in one of two ways:

- **One-way function:** The system stores only the value of a function based on the user's password. When the user presents a password, the system transforms that password and compares it with the stored value. In practice, the system usually performs a one-way transformation (not reversible) in which the password is used to generate a key for the one-way function and in which a fixed-length output is produced.
- **Access control:** Access to the password file is limited to one or a very few accounts.

If one or both of these countermeasures are in place, some effort is needed for a potential intruder to learn passwords. On the basis of a survey of the literature and interviews with a number of password crackers, [ALVA90] reports the following techniques for learning passwords:

1. Try default passwords used with standard accounts that are shipped with the system. Many administrators do not bother to change these defaults.
2. Exhaustively try all short passwords (those of one to three characters).
3. Try words in the system's online dictionary or a list of likely passwords. Examples of the latter are readily available on hacker bulletin boards.
4. Collect information about users, such as their full names, the names of their spouse and children, pictures in their office, and books in their office that are related to hobbies.
5. Try users' phone numbers, Social Security numbers, and room numbers.
6. Try all legitimate license plate numbers for this state.
7. Use a Trojan horse (described in Chapter 21) to bypass restrictions on access.
8. Tap the line between a remote user and the host system.

The first six methods are various ways of guessing a password. If an intruder has to verify the guess by attempting to log in, it is a tedious and easily countered means of attack. For example, a system can simply reject any login after three password attempts, thus requiring the intruder to reconnect to the host to try again. Under these circumstances, it is not practical to try more than a handful of passwords. However, the intruder is unlikely to try such crude methods. For example, if an intruder can gain access with a low level of privileges to an encrypted password file, then the strategy would be to capture that file and then use the encryption mechanism of that particular system at leisure until a valid password that provided greater privileges was discovered.

Guessing attacks are feasible, and indeed highly effective, when a large number of guesses can be attempted automatically and each guess verified, without the guessing process being detectable. Later in this chapter, we have much to say about thwarting guessing attacks.

The seventh method of attack listed earlier, the Trojan horse, can be particularly difficult to counter. An example of a program that bypasses access controls was cited in [ALVA90]. A low-privilege user produced a game program and invited the system operator to use it in his or her spare time. The program did indeed play a game, but in the background it also contained code to copy the password file, which was unencrypted but access protected, into the user's file. Because the game was running under the operator's high-privilege mode, it was able to gain access to the password file.

The eighth attack listed, line tapping, is a matter of physical security.

Other intrusion techniques do not require learning a password. Intruders can get access to a system by exploiting attacks such as buffer overflows on a program that runs with certain privileges. Privilege escalation can be done this way as well.

We turn now to a discussion of the two principal countermeasures: detection and prevention. Detection is concerned with learning of an attack, either before or after its success. Prevention is a challenging security goal and an uphill battle at all times. The difficulty stems from the fact that the defender must attempt to thwart all possible attacks, whereas the attacker is free to try to find the weakest link in the defense chain and attack at that point.

## 20.2 INTRUSION DETECTION

Inevitably, the best intrusion prevention system will fail. A system's second line of defense is intrusion detection, and this has been the focus of much research in recent years. This interest is motivated by a number of considerations, including the following:

1. If an intrusion is detected quickly enough, the intruder can be identified and ejected from the system before any damage is done or any data are compromised. Even if the detection is not sufficiently timely to preempt the intruder, the sooner that the intrusion is detected, the less the amount of damage and the more quickly that recovery can be achieved.
2. An effective intrusion detection system can serve as a deterrent, so acting to prevent intrusions.
3. Intrusion detection enables the collection of information about intrusion techniques that can be used to strengthen the intrusion prevention facility.

Intrusion detection is based on the assumption that the behavior of the intruder differs from that of a legitimate user in ways that can be quantified. Of course, we cannot expect that there will be a crisp, exact distinction between an attack by an intruder and the normal use of resources by an authorized user. Rather, we must expect that there will be some overlap.

Figure 20.1 suggests, in very abstract terms, the nature of the task confronting the designer of an intrusion detection system. Although the typical behavior of an

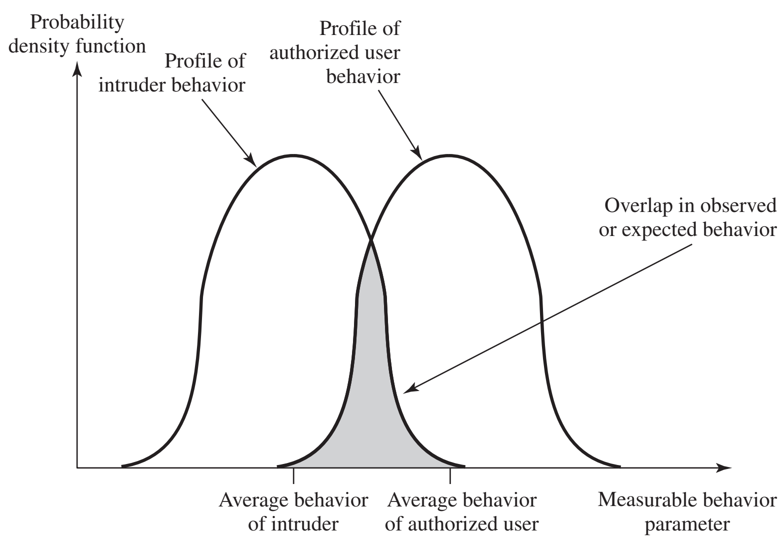



**Figure 20.1** Profiles of Behavior of Intruders and Authorized Users

intruder differs from the typical behavior of an authorized user, there is an overlap in these behaviors. Thus, a loose interpretation of intruder behavior, which will catch more intruders, will also lead to a number of "false positives," or authorized users identified as intruders. On the other hand, an attempt to limit false positives by a tight interpretation of intruder behavior will lead to an increase in false negatives, or intruders not identified as intruders. Thus, there is an element of compromise and art in the practice of intrusion detection.

In Anderson's study [ANDE80], it was postulated that one could, with reasonable confidence, distinguish between a masquerader and a legitimate user. Patterns of legitimate user behavior can be established by observing past history, and significant deviation from such patterns can be detected. Anderson suggests that the task of detecting a misfeasor (legitimate user performing in an unauthorized fashion) is more difficult, in that the distinction between abnormal and normal behavior may be small. Anderson concluded that such violations would be undetectable solely through the search for anomalous behavior. However, misfeasor behavior might nevertheless be detectable by intelligent definition of the class of conditions that suggest unauthorized use. Finally, the detection of the clandestine user was felt to be beyond the scope of purely automated techniques. These observations, which were made in 1980, remain true today.

[PORR92] identifies the following approaches to intrusion detection:

1. **Statistical anomaly detection:** Involves the collection of data relating to the behavior of legitimate users over a period of time. Then statistical tests are applied to observed behavior to determine with a high level of confidence whether that behavior is not legitimate user behavior.
   a. Threshold detection: This approach involves defining thresholds, independent of user, for the frequency of occurrence of various events.
   b. Profile based: A profile of the activity of each user is developed and used to detect changes in the behavior of individual accounts.
2. **Rule-based detection:** Involves an attempt to define a set of rules that can be used to decide that a given behavior is that of an intruder.
   a. Anomaly detection: Rules are developed to detect deviation from previous usage patterns.
   b. Penetration identification: An expert system approach that searches for suspicious behavior.

In a nutshell, statistical approaches attempt to define normal, or expected, behavior, whereas rule-based approaches attempt to define proper behavior.

In terms of the types of attackers listed earlier, statistical anomaly detection is effective against masqueraders, who are unlikely to mimic the behavior patterns of the accounts they appropriate. On the other hand, such techniques may be unable to deal with misfeasors. For such attacks, rule-based approaches may be able to recognize events and sequences that, in context, reveal penetration. In practice, a system may exhibit a combination of both approaches to be effective against a broad range of attacks.

### Audit Records

A fundamental tool for intrusion detection is the audit record. Some record of ongoing activity by users must be maintained as input to an intrusion detection system. Basically, two plans are used:

- **Native audit records:** Virtually all multiuser operating systems include accounting software that collects information on user activity. The advantage of using this information is that no additional collection software is needed. The disadvantage is that the native audit records may not contain the needed information or may not contain it in a convenient form.
- **Detection-specific audit records:** A collection facility can be implemented that generates audit records containing only that information required by the intrusion detection system. One advantage of such an approach is that it could be made vendor independent and ported to a variety of systems. The disadvantage is the extra overhead involved in having, in effect, two accounting packages running on a machine.

A good example of detection-specific audit records is one developed by Dorothy Denning [DENN87]. Each audit record contains the following fields:

- **Subject:** Initiators of actions. A subject is typically a terminal user but might also be a process acting on behalf of users or groups of users. All activity arises through commands issued by subjects. Subjects may be grouped into different access classes, and these classes may overlap.
- **Action:** Operation performed by the subject on or with an object; for example, login, read, perform I/O, execute.
- **Object:** Receptors of actions. Examples include files, programs, messages, records, terminals, printers, and user- or program-created structures. When a subject is the recipient of an action, such as electronic mail, then that subject is considered an object. Objects may be grouped by type. Object granularity may vary by object type and by environment. For example, database actions may be audited for the database as a whole or at the record level.
- **Exception-Condition:** Denotes which, if any, exception condition is raised on return.
- **Resource-Usage:** A list of quantitative elements in which each element gives the amount used of some resource (e.g., number of lines printed or displayed, number of records read or written, processor time, I/O units used, session elapsed time).
- **Time-Stamp:** Unique time-and-date stamp identifying when the action took place.

Most user operations are made up of a number of elementary actions. For example, a file copy involves the execution of the user command, which includes doing access validation and setting up the copy, plus the read from one file, plus the write to another file. Consider the command

```
COPY GAME.EXE TO <Libray>GAME.EXE
```

issued by Smith to copy an executable file GAME from the current directory to the <Library> directory. The following audit records may be generated:

| | | | | | |
|---|---|---|---|---|---|
| Smith | execute | <Library>COPY.EXE | 0 | CPU = 00002 | 11058721678 |

| | | | | | |
|---|---|---|---|---|---|
| Smith | read | <Smith>GAME.EXE | 0 | RECORDS = 0 | 11058721679 |

| | | | | | |
|---|---|---|---|---|---|
| Smith | execute | <Library>COPY.EXE | write-viol | RECORDS = 0 | 11058721680 |

In this case, the copy is aborted because Smith does not have write permission to <Library>.

The decomposition of a user operation into elementary actions has three advantages:

1. Because objects are the protectable entities in a system, the use of elementary actions enables an audit of all behavior affecting an object. Thus, the system can detect attempted subversions of access controls (by noting an abnormality in the number of exception conditions returned) and can detect successful subversions by noting an abnormality in the set of objects accessible to the subject.
2. Single-object, single-action audit records simplify the model and the implementation.
3. Because of the simple, uniform structure of the detection-specific audit records, it may be relatively easy to obtain this information or at least part of it by a straightforward mapping from existing native audit records to the detection-specific audit records.

### Statistical Anomaly Detection

As was mentioned, statistical anomaly detection techniques fall into two broad categories: threshold detection and profile-based systems. Threshold detection involves counting the number of occurrences of a specific event type over an interval of time. If the count surpasses what is considered a reasonable number that one might expect to occur, then intrusion is assumed.

Threshold analysis, by itself, is a crude and ineffective detector of even moderately sophisticated attacks. Both the threshold and the time interval must be determined. Because of the variability across users, such thresholds are likely to generate either a lot of false positives or a lot of false negatives. However, simple threshold detectors may be useful in conjunction with more sophisticated techniques.

Profile-based anomaly detection focuses on characterizing the past behavior of individual users or related groups of users and then detecting significant deviations. A profile may consist of a set of parameters, so that deviation on just a single parameter may not be sufficient in itself to signal an alert.

The foundation of this approach is an analysis of audit records. The audit records provide input to the intrusion detection function in two ways. First, the designer must decide on a number of quantitative metrics that can be used to measure user behavior. An analysis of audit records over a period of time can be used to determine the activity profile of the average user. Thus, the audit records serve to define typical behavior. Second, current audit records are the input used to detect intrusion. That is, the intrusion detection model analyzes incoming audit records to determine deviation from average behavior.

Examples of metrics that are useful for profile-based intrusion detection are the following:

- **Counter:** A nonnegative integer that may be incremented but not decremented until it is reset by management action. Typically, a count of certain event types is kept over a particular period of time. Examples include the number of logins by a single user during an hour, the number of times a given command is executed during a single user session, and the number of password failures during a minute.
- **Gauge:** A nonnegative integer that may be incremented or decremented. Typically, a gauge is used to measure the current value of some entity. Examples include the number of logical connections assigned to a user application and the number of outgoing messages queued for a user process.
- **Interval timer:** The length of time between two related events. An example is the length of time between successive logins to an account.
- **Resource utilization:** Quantity of resources consumed during a specified period. Examples include the number of pages printed during a user session and total time consumed by a program execution.

Given these general metrics, various tests can be performed to determine whether current activity fits within acceptable limits. [DENN87] lists the following approaches that may be taken:

- Mean and standard deviation
- Multivariate
- Markov process
- Time series
- Operational

The simplest statistical test is to measure the **mean and standard deviation** of a parameter over some historical period. This gives a reflection of the average behavior and its variability. The use of mean and standard deviation is applicable to a wide variety of counters, timers, and resource measures. But these measures, by themselves, are typically too crude for intrusion detection purposes.

A **multivariate** model is based on correlations between two or more variables. Intruder behavior may be characterized with greater confidence by considering such correlations (for example, processor time and resource usage, or login frequency and session elapsed time).

A **Markov process** model is used to establish transition probabilities among various states. As an example, this model might be used to look at transitions between certain commands.

A **time series** model focuses on time intervals, looking for sequences of events that happen too rapidly or too slowly. A variety of statistical tests can be applied to characterize abnormal timing.

Finally, an **operational model** is based on a judgment of what is considered abnormal, rather than an automated analysis of past audit records. Typically, fixed limits are defined and intrusion is suspected for an observation that is outside the limits. This type of approach works best where intruder behavior can be deduced from certain types of activities. For example, a large number of login attempts over a short period suggests an attempted intrusion.

As an example of the use of these various metrics and models, Table 20.2 shows various measures considered or tested for the Stanford Research Institute (SRI) intrusion detection system (IDES) [DENN87, JAVI91, LUNT88].

The main advantage of the use of statistical profiles is that a prior knowledge of security flaws is not required. The detector program learns what is "normal" behavior and then looks for deviations. The approach is not based on system-dependent characteristics and vulnerabilities. Thus, it should be readily portable among a variety of systems.

## Rule-Based Intrusion Detection

Rule-based techniques detect intrusion by observing events in the system and applying a set of rules that lead to a decision regarding whether a given pattern of activity is or is not suspicious. In very general terms, we can characterize all approaches as focusing on either anomaly detection or penetration identification, although there is some overlap in these approaches.

**Rule-based anomaly detection** is similar in terms of its approach and strengths to statistical anomaly detection. With the rule-based approach, historical audit records are analyzed to identify usage patterns and to generate automatically rules that describe those patterns. Rules may represent past behavior patterns of users, programs, privileges, time slots, terminals, and so on. Current behavior is then observed, and each transaction is matched against the set of rules to determine if it conforms to any historically observed pattern of behavior.

As with statistical anomaly detection, rule-based anomaly detection does not require knowledge of security vulnerabilities within the system. Rather, the scheme is based on observing past behavior and, in effect, assuming that the future will be like the past. In order for this approach to be effective, a rather large database of rules will be needed. For example, a scheme described in [VACC89] contains anywhere from $10^4$ to $10^6$ rules.

**Rule-based penetration identification** takes a very different approach to intrusion detection. The key feature of such systems is the use of rules for identifying known penetrations or penetrations that would exploit known weaknesses. Rules can also be defined that identify suspicious behavior, even when the behavior is within the bounds of established patterns of usage. Typically, the rules used in these systems are specific to the machine and operating system. The most fruitful approach to developing such rules is to analyze attack tools and scripts collected on the Internet. These rules can be supplemented with rules generated by knowledgeable security personnel. In this latter case, the normal procedure is to interview

**Table 20.2** Measures That May Be Used for Intrusion Detection

| Measure | Model | Type of Intrusion Detected |
|---|---|---|
| **Login and Session Activity** | | |
| Login frequency by day and time | Mean and standard deviation | Intruders may be likely to log in during off-hours. |
| Frequency of login at different locations | Mean and standard deviation | Intruders may log in from a location that a particular user rarely or never uses. |
| Time since last login | Operational | Break-in on a "dead" account. |
| Elapsed time per session | Mean and standard deviation | Significant deviations might indicate masquerader. |
| Quantity of output to location | Mean and standard deviation | Excessive amounts of data transmitted to remote locations could signify leakage of sensitive data. |
| Session resource utilization | Mean and standard deviation | Unusual processor or I/O levels could signal an intruder. |
| Password failures at login | Operational | Attempted break-in by password guessing. |
| Failures to login from specified terminals | Operational | Attempted break-in. |
| **Command or Program Execution Activity** | | |
| Execution frequency | Mean and standard deviation | May detect intruders, who are likely to use different commands, or a successful penetration by a legitimate user, who has gained access to privileged commands. |
| Program resource utilization | Mean and standard deviation | An abnormal value might suggest injection of a virus or Trojan horse, which performs side-effects that increase I/O or processor utilization. |
| Execution denials | Operational model | May detect penetration attempt by individual user who seeks higher privileges. |
| **File Access Activity** | | |
| Read, write, create, delete frequency | Mean and standard deviation | Abnormalities for read and write access for individual users may signify masquerading or browsing. |
| Records read, written | Mean and standard deviation | Abnormality could signify an attempt to obtain sensitive data by inference and aggregation. |
| Failure count for read, write, create, delete | Operational | May detect users who persistently attempt to access unauthorized files. |

system administrators and security analysts to collect a suite of known penetration scenarios and key events that threaten the security of the target system.

A simple example of the type of rules that can be used is found in NIDX, an early system that used heuristic rules that can be used to assign degrees of suspicion to activities [BAUE88]. Example heuristics are the following:

1. Users should not read files in other users' personal directories.
2. Users must not write other users' files.

3. Users who log in after hours often access the same files they used earlier.
4. Users do not generally open disk devices directly but rely on higher-level operating system utilities.
5. Users should not be logged in more than once to the same system.
6. Users do not make copies of system programs.

The penetration identification scheme used in IDES is representative of the strategy followed. Audit records are examined as they are generated, and they are matched against the rule base. If a match is found, then the user's *suspicion rating* is increased. If enough rules are matched, then the rating will pass a threshold that results in the reporting of an anomaly.

The IDES approach is based on an examination of audit records. A weakness of this plan is its lack of flexibility. For a given penetration scenario, there may be a number of alternative audit record sequences that could be produced, each varying from the others slightly or in subtle ways. It may be difficult to pin down all these variations in explicit rules. Another method is to develop a higher-level model independent of specific audit records. An example of this is a state transition model known as USTAT [ILGU93]. USTAT deals in general actions rather than the detailed specific actions recorded by the UNIX auditing mechanism. USTAT is implemented on a SunOS system that provides audit records on 239 events. Of these, only 28 are used by a preprocessor, which maps these onto 10 general actions (Table 20.3). Using just these actions and the parameters that are invoked with each action, a state transition diagram is developed that characterizes suspicious activity. Because a number of different auditable events map into a smaller number of actions, the rule-creation process is simpler. Furthermore, the state transition diagram model is easily modified to accommodate newly learned intrusion behaviors.

**Table 20.3** USTAT Actions versus SunOS Event Types

| USTAT Action | SunOS Event Type |
|---|---|
| Read | open_r, open_rc, open_rtc, open_rwc, open_rwtc, open_rt, open_rw, open_rwt |
| Write | truncate, ftruncate, creat, open_rtc, open_rwc, open_rwtc, open_rt, open_rw, open_rwt, open_w, open_wt, open_wc, open_wct |
| Create | mkdir, creat, open_rc, open_rtc, open_rwc, open_rwtc, open_wc, open_wtc, mknod |
| Delete | rmdir, unlink |
| Execute | exec, execve |
| Exit | exit |
| Modify_Owner | chown, fchown |
| Modify_Perm | chmod, fchmod |
| Rename | rename |
| Hardlink | link |

### The Base-Rate Fallacy

To be of practical use, an intrusion detection system should detect a substantial percentage of intrusions while keeping the false alarm rate at an acceptable level. If only a modest percentage of actual intrusions are detected, the system provides a false sense of security. On the other hand, if the system frequently triggers an alert when there is no intrusion (a false alarm), then either system managers will begin to ignore the alarms, or much time will be wasted analyzing the false alarms.

Unfortunately, because of the nature of the probabilities involved, it is very difficult to meet the standard of high rate of detections with a low rate of false alarms. In general, if the actual numbers of intrusions is low compared to the number of legitimate uses of a system, then the false alarm rate will be high unless the test is extremely discriminating. A study of existing intrusion detection systems, reported in [AXEL00], indicated that current systems have not overcome the problem of the base-rate fallacy. See Appendix 20A for a brief background on the mathematics of this problem.

### Distributed Intrusion Detection

Until recently, work on intrusion detection systems focused on single-system standalone facilities. The typical organization, however, needs to defend a distributed collection of hosts supported by a LAN or internetwork. Although it is possible to mount a defense by using stand-alone intrusion detection systems on each host, a more effective defense can be achieved by coordination and cooperation among intrusion detection systems across the network.

Porras points out the following major issues in the design of a distributed intrusion detection system [PORR92]:

- A distributed intrusion detection system may need to deal with different audit record formats. In a heterogeneous environment, different systems will employ different native audit collection systems and, if using intrusion detection, may employ different formats for security-related audit records.
- One or more nodes in the network will serve as collection and analysis points for the data from the systems on the network. Thus, either raw audit data or summary data must be transmitted across the network. Therefore, there is a requirement to assure the integrity and confidentiality of these data. Integrity is required to prevent an intruder from masking his or her activities by altering the transmitted audit information. Confidentiality is required because the transmitted audit information could be valuable.
- Either a centralized or decentralized architecture can be used. With a centralized architecture, there is a single central point of collection and analysis of all audit data. This eases the task of correlating incoming reports but creates a potential bottleneck and single point of failure. With a decentralized architecture, there are more than one analysis centers, but these must coordinate their activities and exchange information.

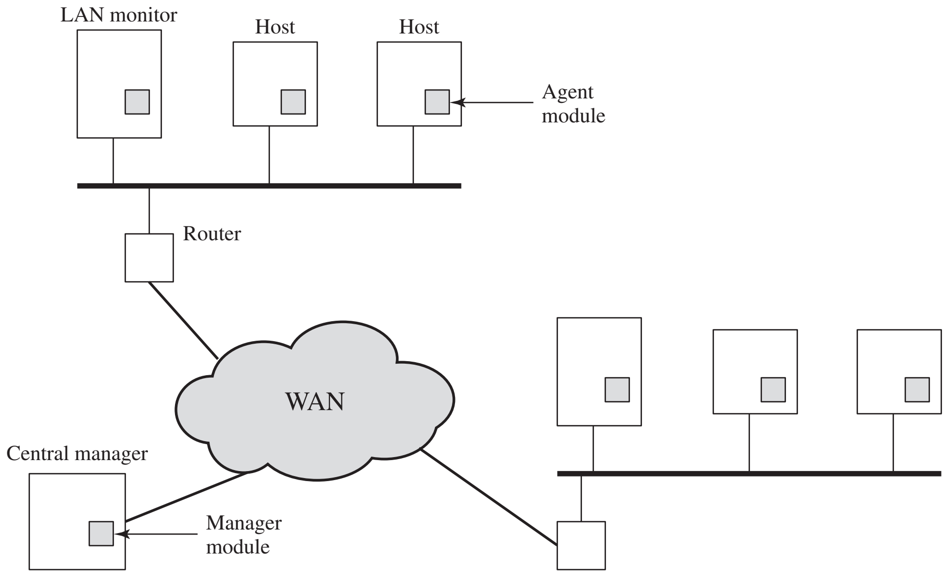



**Figure 20.2** Architecture for Distributed Intrusion Detection

A good example of a distributed intrusion detection system is one developed at the University of California at Davis [HEBE92, SNAP91]. Figure 20.2 shows the overall architecture, which consists of three main components:

- **Host agent module:** An audit collection module operating as a background process on a monitored system. Its purpose is to collect data on security-related events on the host and transmit these to the central manager.
- **LAN monitor agent module:** Operates in the same fashion as a host agent module except that it analyzes LAN traffic and reports the results to the central manager.
- **Central manager module:** Receives reports from LAN monitor and host agents and processes and correlates these reports to detect intrusion.

The scheme is designed to be independent of any operating system or system auditing implementation. Figure 20.3 [SNAP91] shows the general approach that is taken. The agent captures each audit record produced by the native audit collection system. A filter is applied that retains only those records that are of security interest. These records are then reformatted into a standardized format referred to as the host audit record (HAR). Next, a template-driven logic module analyzes the records for suspicious activity. At the lowest level, the agent scans for notable events that are of interest independent of any past events. Examples include failed file accesses, accessing system files, and changing a file's access control. At the next higher level, the agent looks for sequences of events, such as known attack patterns (signatures). Finally, the agent looks for anomalous behavior of an individual user based on a historical profile of that user, such as number of programs executed, number of files accessed, and the like.

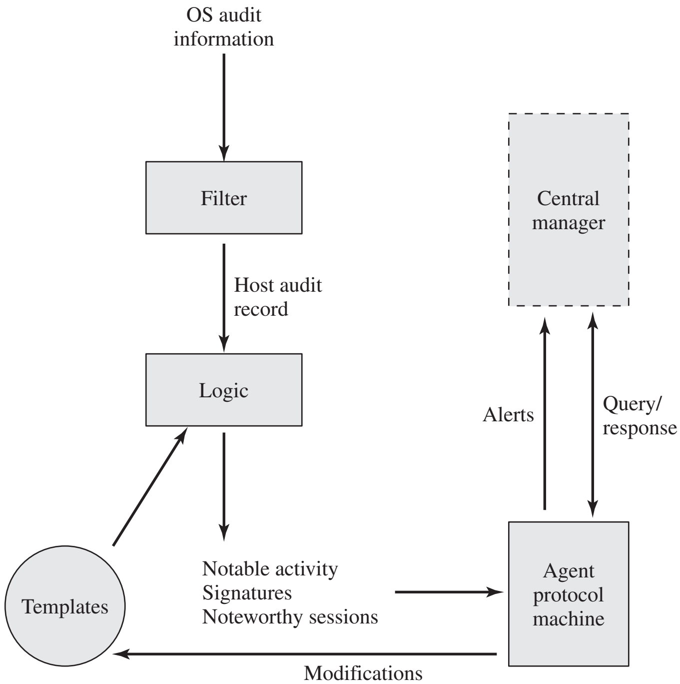



**Figure 20.3** Agent Architecture

When suspicious activity is detected, an alert is sent to the central manager. The central manager includes an expert system that can draw inferences from received data. The manager may also query individual systems for copies of HARs to correlate with those from other agents.

The LAN monitor agent also supplies information to the central manager. The LAN monitor agent audits host-host connections, services used, and volume of traffic. It searches for significant events, such as sudden changes in network load, the use of security-related services, and network activities such as *rlogin*.

The architecture depicted in Figures 20.2 and 20.3 is quite general and flexible. It offers a foundation for a machine-independent approach that can expand from stand-alone intrusion detection to a system that is able to correlate activity from a number of sites and networks to detect suspicious activity that would otherwise remain undetected.

## Honeypots

A relatively recent innovation in intrusion detection technology is the honeypot. Honeypots are decoy systems that are designed to lure a potential attacker away from critical systems. Honeypots are designed to

- divert an attacker from accessing critical systems
- collect information about the attacker's activity
- encourage the attacker to stay on the system long enough for administrators to respond

These systems are filled with fabricated information designed to appear valuable but that a legitimate user of the system wouldn't access. Thus, any access to the honeypot is suspect. The system is instrumented with sensitive monitors and event loggers that detect these accesses and collect information about the attacker's activities. Because any attack against the honeypot is made to seem successful, administrators have time to mobilize and log and track the attacker without ever exposing productive systems.

Initial efforts involved a single honeypot computer with IP addresses designed to attract hackers. More recent research has focused on building entire honeypot networks that emulate an enterprise, possibly with actual or simulated traffic and data. Once hackers are within the network, administrators can observe their behavior in detail and figure out defenses.

### Intrusion Detection Exchange Format

To facilitate the development of distributed intrusion detection systems that can function across a wide range of platforms and environments, standards are needed to support interoperability. Such standards are the focus of the IETF Intrusion Detection Working Group. The purpose of the working group is to define data formats and exchange procedures for sharing information of interest to intrusion detection and response systems and to management systems that may need to interact with them. The outputs of this working group include:

1. A requirements document, which describes the high-level functional requirements for communication between intrusion detection systems and requirements for communication between intrusion detection systems and with management systems, including the rationale for those requirements. Scenarios will be used to illustrate the requirements.
2. A common intrusion language specification, which describes data formats that satisfy the requirements.
3. A framework document, which identifies existing protocols best used for communication between intrusion detection systems, and describes how the devised data formats relate to them.

As of this writing, all of these documents are in an Internet-draft document stage.

## 20.3 PASSWORD MANAGEMENT

### Password Protection

The front line of defense against intruders is the password system. Virtually all multiuser systems require that a user provide not only a name or identifier (ID) but also a password. The password serves to authenticate the ID of the individual logging on to the system. In turn, the ID provides security in the following ways:

- The ID determines whether the user is authorized to gain access to a system. In some systems, only those who already have an ID filed on the system are allowed to gain access.

- The ID determines the privileges accorded to the user. A few users may have supervisory or "superuser" status that enables them to read files and perform functions that are especially protected by the operating system. Some systems have guest or anonymous accounts, and users of these accounts have more limited privileges than others.
- The ID is used in what is referred to as discretionary access control. For example, by listing the IDs of the other users, a user may grant permission to them to read files owned by that user.

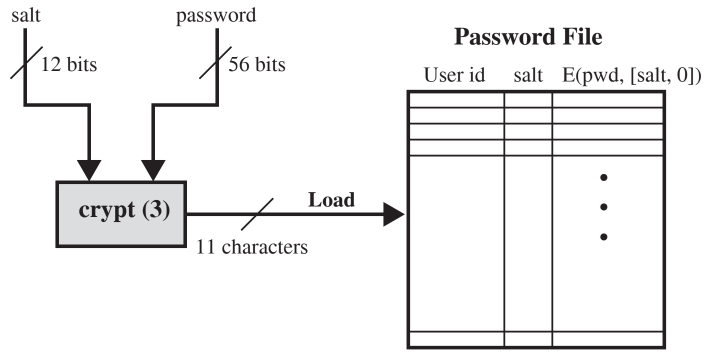
*The Vulnerability of Passwords* To understand the nature of the threat to password-based systems, let us consider a scheme that is widely used on UNIX, in which passwords are never stored in the clear. Rather, the following procedure is employed (Figure 20.4a). Each user selects a password of up to eight printable characters in length. This is converted into a 56-bit value (using 7-bit ASCII) that serves as the key input to an encryption routine. The encryption routine, known as crypt(3), is based on DES. The DES algorithm is modified using a 12-bit "salt" value. Typically, this value is related to the time at which the password is assigned to the user. The modified DES algorithm is exercised with a data input consisting of a 64-bit block of zeros. The output of the algorithm then serves as input for a second encryption. This process is repeated for a total of 25 encryptions. The resulting 64-bit output is then translated into an 11-character sequence. The hashed password is then stored, together with a plaintext copy of the salt, in the password file for the corresponding user ID. This method has been shown to be secure against a variety of cryptanalytic attacks [WAGN00].

The salt serves three purposes:

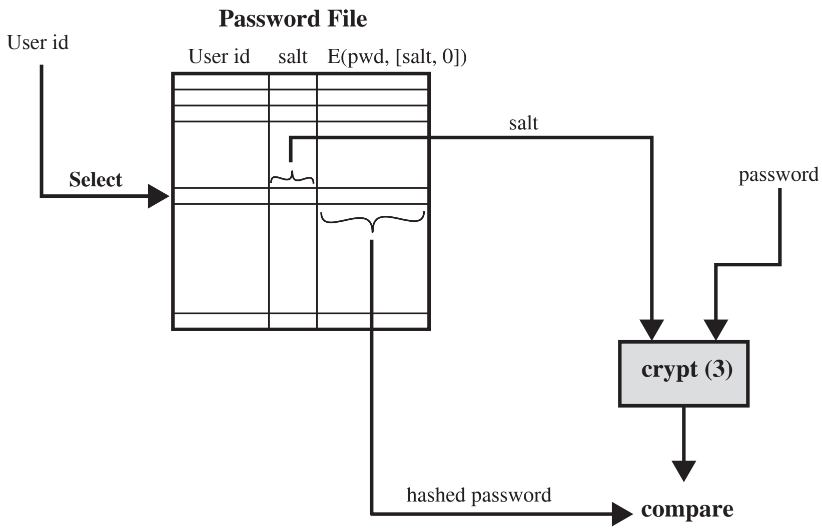
- It prevents duplicate passwords from being visible in the password file. Even if two users choose the same password, those passwords will be assigned at different times. Hence, the "extended" passwords of the two users will differ.
- It effectively increases the length of the password without requiring the user to remember two additional characters. Hence, the number of possible passwords is increased by a factor of 4096, increasing the difficulty of guessing a password.
- It prevents the use of a hardware implementation of DES, which would ease the difficulty of a brute-force guessing attack.

When a user attempts to log on to a UNIX system, the user provides an ID and a password. The operating system uses the ID to index into the password file and retrieve the plaintext salt and the encrypted password. The salt and user-supplied password are used as input to the encryption routine. If the result matches the stored value, the password is accepted.

The encryption routine is designed to discourage guessing attacks. Software implementations of DES are slow compared to hardware versions, and the use of 25 iterations multiplies the time required by 25. However, since the original design of this algorithm, two changes have occurred. First, newer implementations of the algorithm itself have resulted in speedups. For example, the Morris worm described in Chapter 21 was able to do online password guessing of a few hundred passwords




**(a) Loading a new password**




**(b) Verifying a password**

**Figure 20.4** UNIX Password Scheme

in a reasonably short time by using a more efficient encryption algorithm than the standard one stored on the UNIX systems that it attacked. Second, hardware performance continues to increase, so that any software algorithm executes more quickly.

Thus, there are two threats to the UNIX password scheme. First, a user can gain access on a machine using a guest account or by some other means and then run a password guessing program, called a password cracker, on that machine. The attacker should be able to check hundreds and perhaps thousands of possible passwords with little resource consumption. In addition, if an opponent is able to obtain a copy of the password file, then a cracker program can be run on another machine at leisure. This enables the opponent to run through many thousands of possible passwords in a reasonable period.

As an example, a password cracker was reported on the Internet in August 1993 [MADS93]. Using a Thinking Machines Corporation parallel computer, a performance of 1560 encryptions per second per vector unit was achieved. With four vector units per processing node (a standard configuration), this works out to 800,000 encryptions per second on a 128-node machine (which is a modest size) and 6.4 million encryptions per second on a 1024-node machine.

Even these stupendous guessing rates do not yet make it feasible for an attacker to use a dumb brute-force technique of trying all possible combinations of characters to discover a password. Instead, password crackers rely on the fact that some people choose easily guessable passwords.

Some users, when permitted to choose their own password, pick one that is absurdly short. The results of one study at Purdue University are shown in Table 20.4. The study observed password change choices on 54 machines, representing approximately 7000 user accounts. Almost 3% of the passwords were three characters or fewer in length. An attacker could begin the attack by exhaustively testing all possible passwords of length 3 or fewer. A simple remedy is for the system to reject any password choice of fewer than, say, six characters or even to require that all passwords be exactly eight characters in length. Most users would not complain about such a restriction.

Password length is only part of the problem. Many people, when permitted to choose their own password, pick a password that is guessable, such as their own name, their street name, a common dictionary word, and so forth. This makes the job of password cracking straightforward. The cracker simply has to test the password file against lists of likely passwords. Because many people use guessable passwords, such a strategy should succeed on virtually all systems.

One demonstration of the effectiveness of guessing is reported in [KLEI90]. From a variety of sources, the author collected UNIX password files, containing nearly 14,000 encrypted passwords. The result, which the author rightly characterizes

**Table 20.4** Observed Password Lengths [SPAF92a]

| Length | Number | Fraction of Total |
|---|---|---|
| 1 | 55 | .004 |
| 2 | 87 | .006 |
| 3 | 212 | .02 |
| 4 | 449 | .03 |
| 5 | 1260 | .09 |
| 6 | 3035 | .22 |
| 7 | 2917 | .21 |
| 8 | 5772 | .42 |
| Total | 13787 | 1.0 |

as frightening, is shown in Table 20.5. In all, nearly one-fourth of the passwords were guessed. The following strategy was used:

1. Try the user's name, initials, account name, and other relevant personal information. In all, 130 different permutations for each user were tried.
2. Try words from various dictionaries. The author compiled a dictionary of over 60,000 words, including the online dictionary on the system itself, and various other lists as shown.

**Table 20.5** Passwords Cracked from a Sample Set of 13,797 Accounts [KLEI90]

| Type of Password | Search Size | Number of Matches | Percentage of Passwords Matched | Cost/Benefit Ratio[a] |
|---|---|---|---|---|
| User/account name | 130 | 368 | 2.7% | 2.830 |
| Character sequences | 866 | 22 | 0.2% | 0.025 |
| Numbers | 427 | 9 | 0.1% | 0.021 |
| Chinese | 392 | 56 | 0.4% | 0.143 |
| Place names | 628 | 82 | 0.6% | 0.131 |
| Common names | 2239 | 548 | 4.0% | 0.245 |
| Female names | 4280 | 161 | 1.2% | 0.038 |
| Male names | 2866 | 140 | 1.0% | 0.049 |
| Uncommon names | 4955 | 130 | 0.9% | 0.026 |
| Myths & legends | 1246 | 66 | 0.5% | 0.053 |
| Shakespearean | 473 | 11 | 0.1% | 0.023 |
| Sports terms | 238 | 32 | 0.2% | 0.134 |
| Science fiction | 691 | 59 | 0.4% | 0.085 |
| Movies and actors | 99 | 12 | 0.1% | 0.121 |
| Cartoons | 92 | 9 | 0.1% | 0.098 |
| Famous people | 290 | 55 | 0.4% | 0.190 |
| Phrases and patterns | 933 | 253 | 1.8% | 0.271 |
| Surnames | 33 | 9 | 0.1% | 0.273 |
| Biology | 58 | 1 | 0.0% | 0.017 |
| System dictionary | 19683 | 1027 | 7.4% | 0.052 |
| Machine names | 9018 | 132 | 1.0% | 0.015 |
| Mnemonics | 14 | 2 | 0.0% | 0.143 |
| King James bible | 7525 | 83 | 0.6% | 0.011 |
| Miscellaneous words | 3212 | 54 | 0.4% | 0.017 |
| Yiddish words | 56 | 0 | 0.0% | 0.000 |
| Asteroids | 2407 | 19 | 0.1% | 0.007 |
| TOTAL | 62727 | 3340 | 24.2% | 0.053 |

[a]Computed as the number of matches divided by the search size. The more words that needed to be tested for a match, the lower the cost/benefit ratio.

3. Try various permutations on the words from step 2. This included making the first letter uppercase or a control character, making the entire word uppercase, reversing the word, changing the letter "o" to the digit "zero," and so on. These permutations added another 1 million words to the list.
4. Try various capitalization permutations on the words from step 2 that were not considered in step 3. This added almost 2 million additional words to the list.

Thus, the test involved in the neighborhood of 3 million words. Using the fastest Thinking Machines implementation listed earlier, the time to encrypt all these words for all possible salt values is under an hour. Keep in mind that such a thorough search could produce a success rate of about 25%, whereas even a single hit may be enough to gain a wide range of privileges on a system.

*Access Control* One way to thwart a password attack is to deny the opponent access to the password file. If the encrypted password portion of the file is accessible only by a privileged user, then the opponent cannot read it without already knowing the password of a privileged user. [SPAF92a] points out several flaws in this strategy:

- Many systems, including most UNIX systems, are susceptible to unanticipated break-ins. Once an attacker has gained access by some means, he or she may wish to obtain a collection of passwords in order to use different accounts for different logon sessions to decrease the risk of detection. Or a user with an account may desire another user's account to access privileged data or to sabotage the system.
- An accident of protection might render the password file readable, thus compromising all the accounts.
- Some of the users have accounts on other machines in other protection domains, and they use the same password. Thus, if the passwords could be read by anyone on one machine, a machine in another location might be compromised.

Thus, a more effective strategy would be to force users to select passwords that are difficult to guess.

### Password Selection Strategies

The lesson from the two experiments just described (Tables 20.4 and 20.5) is that, left to their own devices, many users choose a password that is too short or too easy to guess. At the other extreme, if users are assigned passwords consisting of eight randomly selected printable characters, password cracking is effectively impossible. But it would be almost as impossible for most users to remember their passwords. Fortunately, even if we limit the password universe to strings of characters that are reasonably memorable, the size of the universe is still too large to permit practical cracking. Our goal, then, is to eliminate guessable passwords while allowing the user to select a password that is memorable. Four basic techniques are in use:

- User education
- Computer-generated passwords

- Reactive password checking
- Proactive password checking

Users can be told the importance of using hard-to-guess passwords and can be provided with guidelines for selecting strong passwords. This **user education** strategy is unlikely to succeed at most installations, particularly where there is a large user population or a lot of turnover. Many users will simply ignore the guidelines. Others may not be good judges of what is a strong password. For example, many users (mistakenly) believe that reversing a word or capitalizing the last letter makes a password unguessable.

**Computer-generated passwords** also have problems. If the passwords are quite random in nature, users will not be able to remember them. Even if the password is pronounceable, the user may have difficulty remembering it and so be tempted to write it down. In general, computer-generated password schemes have a history of poor acceptance by users. FIPS PUB 181 defines one of the best-designed automated password generators. The standard includes not only a description of the approach but also a complete listing of the C source code of the algorithm. The algorithm generates words by forming pronounceable syllables and concatenating them to form a word. A random number generator produces a random stream of characters used to construct the syllables and words.

A **reactive password checking** strategy is one in which the system periodically runs its own password cracker to find guessable passwords. The system cancels any passwords that are guessed and notifies the user. This tactic has a number of drawbacks. First, it is resource intensive if the job is done right. Because a determined opponent who is able to steal a password file can devote full CPU time to the task for hours or even days, an effective reactive password checker is at a distinct disadvantage. Furthermore, any existing passwords remain vulnerable until the reactive password checker finds them.

The most promising approach to improved password security is a **proactive password checker**. In this scheme, a user is allowed to select his or her own password. However, at the time of selection, the system checks to see if the password is allowable and, if not, rejects it. Such checkers are based on the philosophy that, with sufficient guidance from the system, users can select memorable passwords from a fairly large password space that are not likely to be guessed in a dictionary attack.

The trick with a proactive password checker is to strike a balance between user acceptability and strength. If the system rejects too many passwords, users will complain that it is too hard to select a password. If the system uses some simple algorithm to define what is acceptable, this provides guidance to password crackers to refine their guessing technique. In the remainder of this subsection, we look at possible approaches to proactive password checking.

The first approach is a simple system for rule enforcement. For example, the following rules could be enforced:

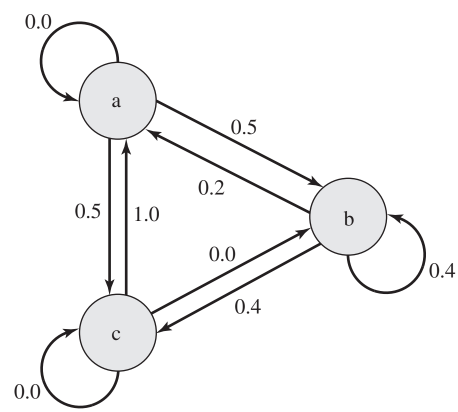
- All passwords must be at least eight characters long.
- In the first eight characters, the passwords must include at least one each of uppercase, lowercase, numeric digits, and punctuation marks.

These rules could be coupled with advice to the user. Although this approach is superior to simply educating users, it may not be sufficient to thwart password crackers. This scheme alerts crackers as to which passwords *not* to try but may still make it possible to do password cracking.

Another possible procedure is simply to compile a large dictionary of possible "bad" passwords. When a user selects a password, the system checks to make sure that it is not on the disapproved list. There are two problems with this approach:

- **Space:** The dictionary must be very large to be effective. For example, the dictionary used in the Purdue study [SPAF92a] occupies more than 30 megabytes of storage.
- **Time:** The time required to search a large dictionary may itself be large. In addition, to check for likely permutations of dictionary words, either those words most be included in the dictionary, making it truly huge, or each search must also involve considerable processing.

Two techniques for developing an effective and efficient proactive password checker that is based on rejecting words on a list show promise. One of these develops a Markov model for the generation of guessable passwords [DAVI93]. Figure 20.5 shows a simplified version of such a model. This model shows a language consisting of an alphabet of three characters. The state of the system at any time is the identity of the most recent letter. The value on the transition from one state to another represents the probability that one letter follows another. Thus, the probability that the next letter is b, given that the current letter is a, is 0.5.




M = {3, {a, b, c} , T, 1 } where

$$T = \begin{bmatrix} 0.0 & 0.5 & 0.5 \\ 0.2 & 0.4 & 0.4 \\ 1.0 & 0.0 & 0.0 \end{bmatrix}$$

e.g., string probably from this language: abbcacaba

e.g., string probably not from this language: aacccbaaa

**Figure 20.5** An Example Markov Model

In general, a Markov model is a quadruple $[m, A, \mathbf{T}, k]$, where $m$ is the number of states in the model, $A$ is the state space, $\mathbf{T}$ is the matrix of transition probabilities, and $k$ is the order of the model. For a $k$th-order model, the probability of making a transition to a particular letter depends on the previous $k$ letters that have been generated. Figure 20.5 shows a simple first-order model.

The authors report on the development and use of a second-order model. To begin, a dictionary of guessable passwords is constructed. Then the transition matrix is calculated as follows:

1. Determine the frequency matrix $\mathbf{f}$, where $\mathbf{f}(i, j, k)$ is the number of occurrences of the trigram consisting of the $i$th, $j$th, and $k$th character. For example, the password *parsnips* yields the trigrams par, ars, rsn, sni, nip, and ips.
2. For each bigram $ij$, calculate $\mathbf{f}(i, j, \infty)$ as the total number of trigrams beginning with $ij$. For example, $\mathbf{f}(\text{a}, \text{b}, \infty)$ would be the total number of trigrams of the form aba, abb, abc, and so on.
3. Compute the entries of $\mathbf{T}$ as follows:

$$\mathbf{T}(i, j, k) = \frac{\mathbf{f}(i, j, k)}{\mathbf{f}(i, j, \infty)}$$

The result is a model that reflects the structure of the words in the dictionary. With this model, the question "Is this a bad password?" is transformed into the question "Was this string (password) generated by this Markov model?" For a given password, the transition probabilities of all its trigrams can be looked up. Some standard statistical tests can then be used to determine if the password is likely or unlikely for that model. Passwords that are likely to be generated by the model are rejected. The authors report good results for a second-order model. Their system catches virtually all the passwords in their dictionary and does not exclude so many potentially good passwords as to be user unfriendly.

A quite different approach has been reported by Spafford [SPAF92a, SPAF92b]. It is based on the use of a Bloom filter [BLOO70]. To begin, we explain the operation of the Bloom filter. A Bloom filter of order $k$ consists of a set of $k$ independent hash functions $H_1(x), H_2(x), \ldots, H_k(x)$, where each function maps a password into a hash value in the range 0 to $N - 1$. That is,

$$H_i(X_j) = y \qquad 1 \le i \le k; \qquad 1 \le j \le D; \qquad 0 \le y \le N - 1$$

where

$X_j$ = $j$th word in password dictionary
$D$ = number of words in password dictionary

The following procedure is then applied to the dictionary:

1. A hash table of $N$ bits is defined, with all bits initially set to 0.
2. For each password, its $k$ hash values are calculated, and the corresponding bits in the hash table are set to 1. Thus, if $H_i(X_j) = 67$ for some $(i, j)$, then the sixty-seventh bit of the hash table is set to 1; if the bit already has the value 1, it remains at 1.

When a new password is presented to the checker, its $k$ hash values are calculated. If all the corresponding bits of the hash table are equal to 1, then the password is rejected. All passwords in the dictionary will be rejected. But there will also be some "false positives" (that is, passwords that are not in the dictionary but that produce a match in the hash table). To see this, consider a scheme with two hash functions. Suppose that the passwords *undertaker* and *hulkhogan* are in the dictionary, but *xG%#jj98* is not. Further suppose that

$$H_1(\text{undertaker}) = 25 \quad H_1(\text{hulkhogan}) = 83 \quad H_1(\text{xG\%\#jj98}) = 665$$

$$H_2(\text{undertaker}) = 998 \quad H_2(\text{hulkhogan}) = 665 \quad H_2(\text{xG\%\#jj98}) = 998$$

If the password xG%#jj98 is presented to the system, it will be rejected even though it is not in the dictionary. If there are too many such false positives, it will be difficult for users to select passwords. Therefore, we would like to design the hash scheme to minimize false positives. It can be shown that the probability of a false positive can be approximated by:

$$P \approx \left(1 - e^{kD/N}\right)^k = \left(1 - e^{k/R}\right)^k$$

or, equivalently,

$$R \approx \frac{-k}{\ln(1 - P^{1/k})}$$

where

$k =$ number of hash functions
$N =$ number of bits in hash table
$D =$ number of words in dictionary
$R = N/D$, ratio of hash table size (bits) to dictionary size (words)

Figure 20.6 plots $P$ as a function of $R$ for various values of $k$. Suppose we have a dictionary of 1 million words and we wish to have a 0.01 probability of rejecting a password not in the dictionary. If we choose six hash functions, the required ratio is $R = 9.6$. Therefore, we need a hash table of $9.6 \times 10^6$ bits or about 1.2 MBytes of storage. In contrast, storage of the entire dictionary would require on the order of 8 MBytes. Thus, we achieve a compression of almost a factor of 7. Furthermore, password checking involves the straightforward calculation of six hash functions and is independent of the size of the dictionary, whereas with the use of the full dictionary, there is substantial searching.[1]

[1]Both the Markov model and the Bloom filter involve the use of probabilistic techniques. In the case of the Markov model, there is a small probability that some passwords in the dictionary will not be caught and a small probability that some passwords not in the dictionary will be rejected. In the case of the Bloom filter, there is a small probability that some passwords not in the dictionary will be rejected. Again we see that taking a probabilistic approach simplifies the solution (e.g., see the Miller-Rabin algorithm in Chapter 8).

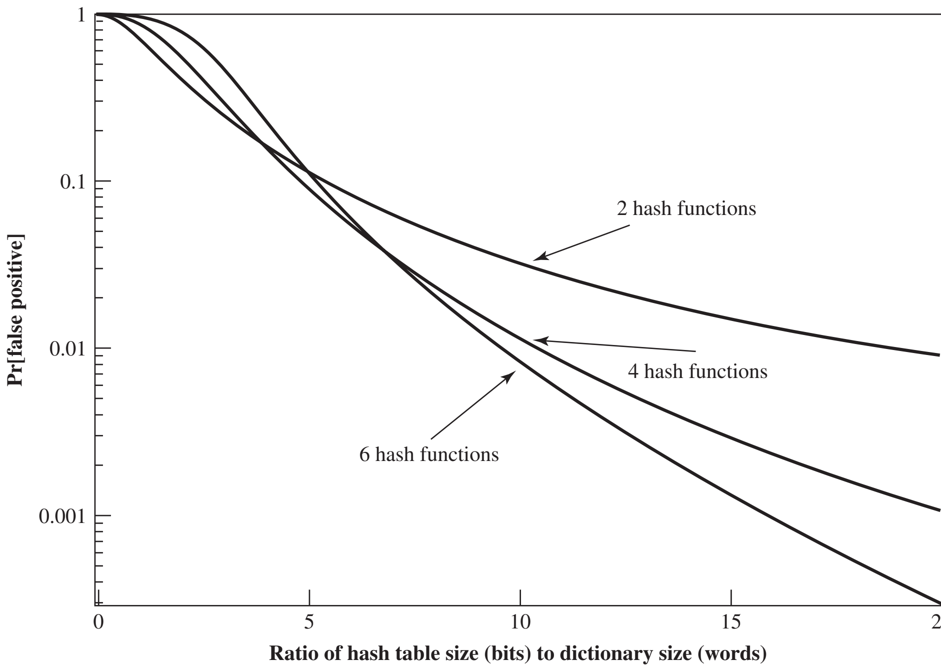



**Figure 20.6** Performance of Bloom Filter

## 20.4 RECOMMENDED READING AND WEB SITES

Two thorough treatments of intrusion detection are [BACE00] and [PROC01]. A more concise but very worthwhile treatment is [SCAR07]. Two short but useful survey articles on the subject are [KENT00] and [MCHU00]. [NING04] surveys recent advances in intrusion detection techniques. [HONE01] is the definitive account on honeypots and provides a detailed analysis of the tools and methods of hackers.


**BACE00** Bace, R. *Intrusion Detection.* Indianapolis, IN: Macmillan Technical Publishing, 2000.

**HONE01** The Honeynet Project. *Know Your Enemy: Revealing the Security Tools, Tactics, and Motives of the Blackhat Community.* Reading, MA: Addison-Wesley, 2001.

**KENT00** Kent, S. "On the Trail of Intrusions into Information Systems." *IEEE Spectrum*, December 2000.

**MCHU00** McHugh, J.; Christie, A.; and Allen, J. "The Role of Intrusion Detection Systems." *IEEE Software*, September/October 2000.

**NING04** Ning, P., et al. "Techniques and Tools for Analyzing Intrusion Alerts." *ACM Transactions on Information and System Security*, May 2004.

**PROC01** Proctor, P., *The Practical Intrusion Detection Handbook.* Upper Saddle River, NJ: Prentice Hall, 2001.

**SCAR07** Scarfone, K., and Mell, P. *Guide to Intrusion Detection and Prevention Systems.* NIST Special Publication SP 800-94, February 2007.

## Recommended Web Sites:

- **CERT Coordination Center:** The organization that grew from the computer emergency response team formed by the Defense Advanced Research Projects Agency. Site provides good information on Internet security threats, vulnerabilities, and attack statistics.
- **Packet Storm:** Resource of up-to-date and historical security tools, exploits, and advisories.
- **Honeynet Project:** A research project studying the techniques of predatory hackers and developing honeypot products.
- **Honeypots:** A good collection of research papers and technical articles.
- **Intrusion Detection Working Group:** IETF group developing standards for exchange formats and exchange procedures for intrusion detection systems. Includes RFCs and Internet drafts.
- **STAT Project:** A research and open-source project at the U. of California, Santa Barbara that focuses on signature-based intrusion detection tools for hosts, applications, and networks.
- **Password Usage and Generation:** NIST documents on this topic.

# 20.5 KEY TERMS, REVIEW QUESTIONS, AND PROBLEMS

## Key Terms

| | | |
|---|---|---|
| audit record | intruder | password |
| Bayes' Theorem | intrusion detection | rule-based intrusion detection |
| base-rate fallacy | intrusion detection exchange format | salt |
| honeypot | | statistical anomaly detection |

## Review Questions

**20.1** List and briefly define three classes of intruders.

**20.2** What are two common techniques used to protect a password file?

**20.3** What are three benefits that can be provided by an intrusion detection system?

**20.4** What is the difference between statistical anomaly detection and rule-based intrusion detection?

**20.5** What metrics are useful for profile-based intrusion detection?

**20.6** What is the difference between rule-based anomaly detection and rule-based penetration identification?

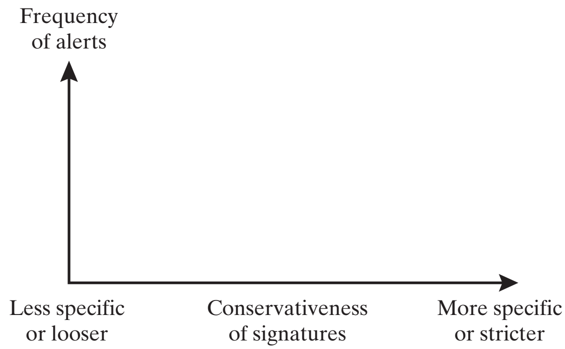
**20.7** What is a honeypot?

**20.8** What is a salt in the context of UNIX password management?

**20.9** List and briefly define four techniques used to avoid guessable passwords.

## Problems

**20.1** In the context of an IDS, we define a false positive to be an alarm generated by an IDS in which the IDS alerts to a condition that is actually benign. A false negative occurs when an IDS fails to generate an alarm when an alert-worthy condition is in effect. Using the following diagram, depict two curves that roughly indicate false positives and false negatives, respectively.




**20.2** The overlapping area of the two probability density functions of Figure 20.1 represents the region in which there is the potential for false positives and false negatives. Further, Figure 20.1 is an idealized and not necessarily representative depiction of the relative shapes of the two density functions. Suppose there is 1 actual intrusion for every 1000 authorized users, and the overlapping area covers 1% of the authorized users and 50% of the intruders.

**a.** Sketch such a set of density functions and argue that this is not an unreasonable depiction.

**b.** What is the probability that an event that occurs in this region is that of an authorized user? Keep in mind that 50% of all intrusions fall in this region.

**20.3** An example of a host-based intrusion detection tool is the tripwire program. This is a file integrity checking tool that scans files and directories on the system on a regular basis and notifies the administrator of any changes. It uses a protected database of cryptographic checksums for each file checked and compares this value with that recomputed on each file as it is scanned. It must be configured with a list of files and directories to check, and what changes, if any, are permissible to each. It can allow, for example, log files to have new entries appended, but not for existing entries to be changed. What are the advantages and disadvantages of using such a tool? Consider the problem of determining which files should only change rarely, which files may change more often and how, and which change frequently and hence cannot be checked. Hence consider the amount of work in both the configuration of the program and on the system administrator monitoring the responses generated.

**20.4** A taxicab was involved in a fatal hit-and-run accident at night. Two cab companies, the Green and the Blue, operate in the city. You are told that:

- 85% of the cabs in the city are Green and 15% are Blue.
- A witness identified the cab as Blue.

The court tested the reliability of the witness under the same circumstances that existed on the night of the accident and concluded that the witness was correct in identifying the color of the cab 80% of the time. What is the probability that the cab involved in the incident was Blue rather than Green?

**20.5** Explain the suitability or unsuitability of the following passwords:

**a.** YK 334 **b.** mfmitm (for "my favorite movie is tender mercies) **c.** Natalie1 **d.** Washington
**e.** Aristotle **f.** tv9stove **g.** 12345678 **h.** dribgib

**20.6** An early attempt to force users to use less predictable passwords involved computer-supplied passwords. The passwords were eight characters long and were taken from the character set consisting of lowercase letters and digits. They were generated by a pseudorandom number generator with $2^{15}$ possible starting values. Using the technology of the time, the time required to search through all character strings of length 8 from a 36-character alphabet was 112 years. Unfortunately, this is not a true reflection of the actual security of the system. Explain the problem.

**20.7** Assume that passwords are selected from four-character combinations of 26 alphabetic characters. Assume that an adversary is able to attempt passwords at a rate of one per second.

**a.** Assuming no feedback to the adversary until each attempt has been completed, what is the expected time to discover the correct password?

**b.** Assuming feedback to the adversary flagging an error as each incorrect character is entered, what is the expected time to discover the correct password?

**20.8** Assume that source elements of length $k$ are mapped in some uniform fashion into a target elements of length $p$. If each digit can take on one of $r$ values, then the number of source elements is $r^k$ and the number of target elements is the smaller number $r^p$. A particular source element $x_i$ is mapped to a particular target element $y_j$.

**a.** What is the probability that the correct source element can be selected by an adversary on one try?

**b.** What is the probability that a different source element $x_k (x_i \neq x_k)$ that results in the same target element, $y_j$, could be produced by an adversary?

**c.** What is the probability that the correct target element can be produced by an adversary on one try?

**20.9** A phonetic password generator picks two segments randomly for each six-letter password. The form of each segment is CVC (consonant, vowel, consonant), where $V = \langle a, e, i, o, u \rangle$ and $C = \overline{V}$.

**a.** What is the total password population?

**b.** What is the probability of an adversary guessing a password correctly?

**20.10** Assume that passwords are limited to the use of the 95 printable ASCII characters and that all passwords are 10 characters in length. Assume a password cracker with an encryption rate of 6.4 million encryptions per second. How long will it take to test exhaustively all possible passwords on a UNIX system?

**20.11** Because of the known risks of the UNIX password system, the SunOS-4.0 documentation recommends that the password file be removed and replaced with a publicly readable file called /etc/publickey. An entry in the file for user A consists of a user's identifier $ID_A$, the user's public key, $PU_a$, and the corresponding private key $PR_a$. This private key is encrypted using DES with a key derived from the user's login password $P_a$. When A logs in, the system decrypts $E(P_a, PR_a)$ to obtain $PR_a$.

**a.** The system then verifies that $P_a$ was correctly supplied. How?

**b.** How can an opponent attack this system?

**20.12** The encryption scheme used for UNIX passwords is one way; it is not possible to reverse it. Therefore, would it be accurate to say that this is, in fact, a hash code rather than an encryption of the password?

**20.13** It was stated that the inclusion of the salt in the UNIX password scheme increases the difficulty of guessing by a factor of 4096. But the salt is stored in plaintext in the same entry as the corresponding ciphertext password. Therefore, those two characters are known to the attacker and need not be guessed. Why is it asserted that the salt increases security?

**20.14** Assuming that you have successfully answered the preceding problem and understand the significance of the salt, here is another question. Wouldn't it be possible to thwart completely all password crackers by dramatically increasing the salt size to, say, 24 or 48 bits?

**20.15** Consider the Bloom filter discussed in Section 20.3. Define $k =$ number of hash functions; $N =$ number of bits in hash table; and $D =$ number of words in dictionary.

a. Show that the expected number of bits in the hash table that are equal to zero is expressed as

$$\phi = \left(1 - \frac{k}{N}\right)^D$$

b. Show that the probability that an input word, not in the dictionary, will be falsely accepted as being in the dictionary is

$$P = (1 - \phi)^k$$

c. Show that the preceding expression can be approximated as

$$P \approx \left(1 - e^{-kD/N}\right)^k$$

20.16 Design a file access system to allow certain users read and write access to a file, depending on authorization set up by the system. The instructions should be of the format:

READ (F, User A): attempt by User A to read file F
READ (F, User A): attempt by User A to store a possibly modified copy of F

Each file has a *header record,* which contains authorization privileges; that is, a list of users who can read and write. The file is to be encrypted by a key that is not shared by the users but known only to the system.

# APPENDIX 20A THE BASE-RATE FALLACY

We begin with a review of important results from probability theory, then demonstrate the base-rate fallacy.

## Conditional Probability and Independence

We often want to know a probability that is conditional on some event. The effect of the condition is to remove some of the outcomes from the sample space. For example, what is the probability of getting a sum of 8 on the roll of two dice, if we know that the face of at least one die is an even number? We can reason as follows. Because one die is even and the sum is even, the second die must show an even number. Thus, there are three equally likely successful outcomes: (2, 6), (4, 4) and (6, 2), out of a total set of possibilities of [36−(number of events with both faces odd)] = 36−(3 × 3) = 27. The resulting probability is 3/27 = 1/9.

Formally, the **conditional probability** of an event $A$ assuming the event $B$ has occurred, denoted by $\Pr[A|B]$, is defined as the ratio

$$\Pr[A \mid B] = \frac{\Pr[AB]}{\Pr[B]}$$

where we assume $\Pr[B]$ is not zero.

In our example, $A$ = {sum of 8} and $B$ = {at least one die even}. The quantity $\Pr[AB]$ encompasses all of those outcomes in which the sum is 8 and at least one die is even. As we have seen, there are three such outcomes. Thus, $\Pr[AB] = 3/36 = 1/12$. A moment's thought should convince you that $\Pr[B] = 3/4$. We can now calculate

$$\Pr[A \mid B] = \frac{1/12}{3/4} = \frac{1}{9}$$

This agrees with our previous reasoning.

Two events $A$ and $B$ are called **independent** if $\Pr[AB] = \Pr[A]\Pr[B]$. It can easily be seen that if $A$ and $B$ are independent, $\Pr[A|B] = \Pr[A]$ and $\Pr[B|A] = \Pr[B]$.

## Bayes' Theorem

One of the most important results from probability theory is known as Bayes' theorem. First we need to state the total probability formula. Given a set of mutually exclusive events $E_1, E_2, \ldots, E_n$, such that the union of these events covers all possible outcomes, and given an arbitrary event $A$, then it can be shown that

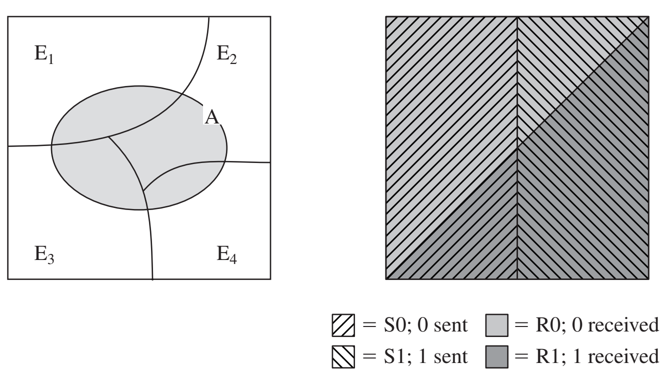
$$\Pr[A] = \sum_{i=1}^{n} \Pr[A|E_i]\Pr[E_i] \qquad \textbf{(20.1)}$$

Bayes' theorem may be stated as follows:

$$\Pr[E_i|A] = \frac{\Pr[A|E_i]P[E_i]}{\Pr[A]} = \frac{\Pr[A|E_i]P[E_i]}{\sum_{j=1}^{n}\Pr[A|E_j]\Pr[E_j]} \qquad \textbf{(20.2)}$$

Figure 20.7a illustrates the concepts of total probability and Bayes' theorem.

Bayes' theorem is used to calculate "posterior odds," that is, the probability that something really is the case, given evidence in favor of it. For example, suppose we are transmitting a sequence of zeroes and ones over a noisy transmission line. Let S0 and S1 be the events a zero is sent at a given time and a one is sent, respectively, and R0 and R1 be the events that a zero is received and a one is received. Suppose we know the probabilities of the source, namely $\Pr[S1] = p$ and $\Pr[S0] = 1 - p$. Now the line is observed to determine how frequently an error occurs when a one is sent and when a zero is sent, and the following probabilities are calculated: $\Pr[R0|S0] = p_a$ and $\Pr[R1|S0] = p_b$. If a zero is received, we can then




(a) Diagram to illustrate concepts (b) Example

**Figure 20.7** Illustration of Total Probability and Bayes' Theorem calculate the conditional probability of an error, namely the conditional probability that a one was sent given that a zero was received, using Bayes' theorem:

$$\Pr[S1 \mid R0] = \frac{\Pr[R0 \mid S1]\Pr[S1]}{\Pr[R0 \mid S1]\Pr[S1] + \Pr[R0 \mid S0]\Pr[S0]} = \frac{p_a p}{p_a p + (1 - p_b)(1 - p)}$$

Figure 20.7b illustrates the preceding equation. In the figure, the sample space is represented by a unit square. Half of the square corresponds to S0 and half to S1, so Pr[S0] = Pr[S1] = 0.5. Similarly, half of the square corresponds to R0 and half to R1, so Pr[R0] = Pr[R1] = 0.5. Within the area representing S0, 1/4 of that area corresponds to R1, so Pr[R1/S0] = 0.25. Other conditional probabilities are similarly evident.

## The Base-Rate Fallacy Demonstrated

Consider the following situation. A patient has a test for some disease that comes back positive (indicating he has the disease). You are told that

- The accuracy of the test is 87% (i.e., if a patient has the disease, 87% of the time, the test yields the correct result, and if the patient does not have the disease, 87% of the time, the test yields the correct result).
- The incidence of the disease in the population is 1%.

Given that the test is positive, how probable is it that the patient does not have the disease? That is, what is the probability that this is a false alarm? We need Bayes' theorem to get the correct answer:

$$\Pr[\text{well/positive}] = \frac{\Pr[\text{positive/well}]\Pr[\text{well}]}{\Pr[\text{positive/disease}]\Pr[\text{disease}] + \Pr[\text{positive/well}]\Pr[\text{well}]}$$

$$= \frac{(0.13)(0.99)}{(0.87)(0.01) + (0.13)(0.99)} = 0.937$$

Thus, in the vast majority of cases, when a disease condition is detected, it is a false alarm.

This problem, used in a study [PIAT91], was presented to a number of people. Most subjects gave the answer 13%. The vast majority, including many physicians, gave a number below 50%. Many physicians who guessed wrong lamented, "If you are right, there is no point in making clinical tests!" The reason most people get it wrong is that they do not take into account the basic rate of incidence (the base rate) when intuitively solving the problem. This error is known as the *base-rate fallacy*.

How could this problem be fixed? Suppose we could drive both of the correct result rates to 99.9%. That is, suppose we have Pr[positive/disease] = 0.999 and Pr[negative/well] = 0.999. Plugging these numbers into the Equation (20.2), we get Pr[well/positive] = 0.09. Thus, if we can accurately detect disease and accurately detect lack of disease at a level of 99.9%, then the rate of false alarms will be 9%. This is much better, but still not ideal. Moreover, again assume 99.9% accuracy, but now suppose that the incidence of the disease in the population is only 1/10000 = 0.0001. We then end up with a rate of false alarms of 91%. In actual situations, [AXEL00] found that the probabilities associated with intrusion detection systems were such that the false alarm rate was unsatisfactory.