CHAPTER 23

Internet Authentication Applications

Learning Objectives

After studying this chapter, you should be able to:

- Summarize the basic operation of Kerberos.
- Compare the functionality of Kerberos version 4 and version 5.
- Understand the format and function of X.509 certificates.
- Explain the public-key infrastructure concept.

This chapter examines some of the authentication functions that have been developed to support network-based authentication and digital signatures.

We begin by looking at one of the earliest and also one of the most widely used services: Kerberos. Next, we examine the X.509 public-key certificates. Then, we examine the concept of a public-key infrastructure (PKI).

23.1 KERBEROS

There are a number of approaches that organizations can use to secure networked servers and hosts. Systems that use one-time passwords thwart any attempt to guess or capture a user's password. These systems require special equipment such as smart cards or synchronized password generators to operate, and have been slow to gain acceptance for general networking use. Another approach is the use of biometric systems. These are automated methods of verifying or recognizing identity on the basis of some physiological characteristic, such as a fingerprint or iris pattern, or a behavioral characteristic, such as handwriting or keystroke patterns. Again, these systems require specialized equipment.

Another way to tackle the problem is the use of authentication software tied to a secure authentication server. This is the approach taken by Kerberos. Kerberos, initially developed at MIT, is a software utility available both in the public domain and in commercially supported versions. Kerberos has been issued as an Internet standard and is the de facto standard for remote authentication, including as part of Microsoft's Active Directory service.

The overall scheme of Kerberos is that of a trusted third-party authentication service. It is trusted in the sense that clients and servers trust Kerberos to mediate their mutual authentication. In essence, Kerberos requires that a user prove his or her identity for each service invoked and, optionally, requires servers to prove their identity to clients.

The Kerberos Protocol

Kerberos makes use of a protocol that involves clients, application servers, and a Kerberos server. That the protocol is complex reflects that fact that there are many ways for an opponent to penetrate security. Kerberos is designed to counter a variety of threats to the security of a client/server dialogue.

The basic idea is simple. In an unprotected network environment, any client can apply to any server for service. The obvious security risk is that of impersonation. An opponent can pretend to be another client and obtain unauthorized privileges on server machines. To counter this threat, servers must be able to confirm the identities of clients who request service. Each server can be required to undertake this task for each client/server interaction, but in an open environment, this places a substantial burden on each server. An alternative is to use an **authentication server (AS)** that knows the passwords of all clients and stores these in a centralized database. Then the user can log onto the AS for identity verification. Once the AS has verified the user's identity, it can pass this information on to an application server, which will then accept service requests from the client.

The trick is how to do all this in a secure way. It simply will not do to have the client send the user's password to the AS over the network: An opponent could observe the password on the network and later reuse it. It also will not do for Kerberos to send a plain message to a server validating a client: An opponent could impersonate the AS and send a false validation.

The way around this problem is to use encryption and a set of messages that accomplish the task (see Figure 23.1). The original version of Kerberos used the Data Encryption Standard (DES) as it's encryption algorithm.

The AS shares a unique secret key with each server. These keys have been distributed physically or in some other secure manner. This will enable the AS to send messages to application servers in a secure fashion. To begin, the user logs on to a workstation and requests access to a particular server. The client process representing the user sends a message to the AS that includes the user's ID and a request for what is known as a **ticket-granting ticket (TGT)**. The AS checks its database to find the password of this user. Then the AS responds with a TGT and a one-time encryption key, known as a session key, both encrypted using the user's password as the encryption key. When this message arrives back at the client, the client prompts the user for his or her password, generates the key, and attempts to decrypt the incoming message. If the correct password has been supplied, the ticket and session key are successfully recovered.

Notice what has happened. The AS has been able to verify the user's identity since this user knows the correct password, but it has been done in such a way that the password is never passed over the network. In addition, the AS has passed information to the client that will be used later on to apply to a server for service, and that information is secure since it is encrypted with the user's password.

The ticket constitutes a set of credentials that can be used by the client to apply for service. The ticket indicates that the AS has accepted this client and its user. The ticket contains the user's ID, the server's ID, a timestamp, a lifetime after which the ticket is invalid, and a copy of the same session key sent in the outer message to the client. The entire ticket is encrypted using a secret DES key shared by the AS and the server. Thus, no one can tamper with the ticket.

Now, Kerberos could have been set up so the AS would send back a ticket granting access to a particular application server. This would require the client to request a new ticket from the AS for each service the user wants to use during a logon session, which would in turn require the AS query the user for his or her password for each service request, or else to store the password in memory for the duration of the

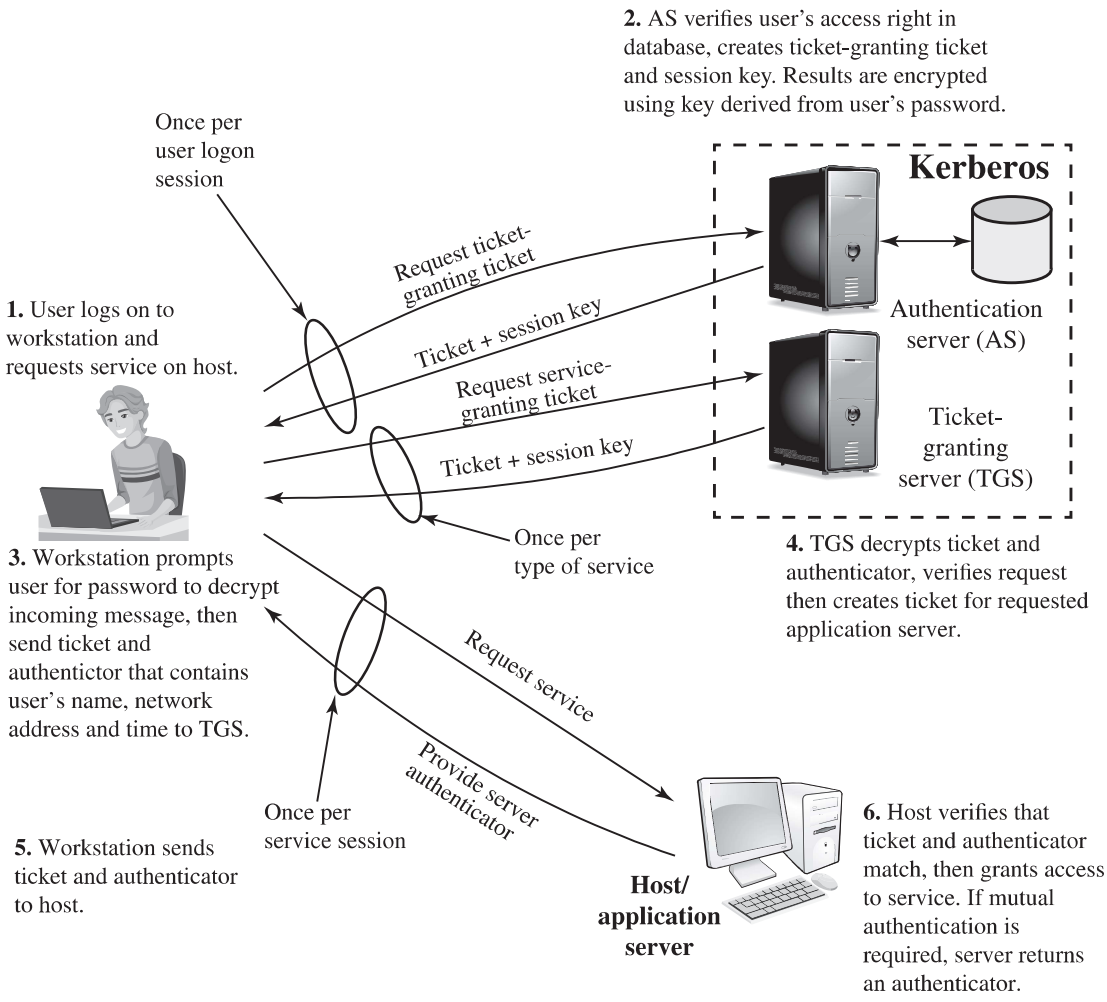


Figure 23.1 Overview of Kerberos

logon session. The first course is inconvenient for the user and the second course is a security risk. Therefore, the AS supplies a ticket good not for a specific application service, but for a special ticket-granting server (TGS). The AS gives the client a ticket that can be used to get more tickets!

The idea is that this ticket can be used by the client to request multiple service-granting tickets. So the ticket-granting ticket is to be reusable. However, we do not wish an opponent to be able to capture the ticket and use it. Consider the following scenario: An opponent captures the ticket and waits until the user has logged off the workstation. Then the opponent either gains access to that workstation or configures his workstation with the same network address as that of the victim. Then the opponent would be able to reuse the ticket to spoof the TGS. To counter this, the ticket includes a timestamp, indicating the date and time at which the ticket was issued, and a lifetime, indicating the length of time for which the ticket is valid (e.g., 8 hours). Thus, the client now has a reusable ticket and need not bother the user for a password for each new service request. Finally, note the ticket-granting ticket is encrypted with a secret key known only to the AS and the TGS. This prevents alteration of the ticket. The ticket is reencrypted with a key based on the user's password.

This assures that the ticket can be recovered only by the correct user, providing the authentication.

Let us see how this works. The user has requested access to server V. The client process representing the user (C) has obtained a ticket-granting ticket and a temporary session key. The client then sends a message to the TGS requesting a ticket for user X that will grant service to server V. The message includes the ID of server V and the ticket-granting ticket. The TGS decrypts the incoming ticket (remember, the ticket is encrypted by a key known only to the AS and the TGS) and verifies the success of the decryption by the presence of its own ID. It checks to make sure that the lifetime has not expired. Then it compares the user ID and network address with the incoming information to authenticate the user.

At this point, the TGS is almost ready to grant a service-granting ticket to the client. But there is one more threat to overcome. The heart of the problem is the lifetime associated with the ticket-granting ticket. If this lifetime is very short (e.g., minutes), then the user will be repeatedly asked for a password. If the lifetime is long (e.g., hours), then an opponent has a greater opportunity for replay. An opponent could eavesdrop on the network and capture a copy of the ticket-granting ticket then wait for the legitimate user to log out. Then the opponent could forge the legitimate user's network address and send a message to the TGS. This would give the opponent unlimited access to the resources and files available to the legitimate user.

To get around this problem, the AS has provided both the client and the TGS with a secret session key that they now share. The session key, recall, was in the message from the AS to the client, encrypted with the user's password. It was also buried in the ticket-granting ticket, encrypted with the key shared by the AS and TGS. In the message to the TGS requesting a service-granting ticket, the client includes an authenticator encrypted with the session key, which contains the ID and address of the user and a timestamp. Unlike the ticket, which is reusable, the authenticator is intended for use only once and has a very short lifetime. Now, TGS can decrypt the ticket with the key that it shares with the AS. This ticket indicates that user X has been provided with the session key. In effect, the ticket says, "Anyone who uses this session key must be X." TGS uses the session key to decrypt the authenticator. The TGS can then check the name and address from the authenticator with that of the ticket and with the network address of the incoming message. If all match, then the TGS is assured that the sender of the ticket is indeed the ticket's real owner. In effect, the authenticator says, "At the time of this authenticator, I hereby use this session key." Note the ticket does not prove anyone's identity, but is a way to distribute keys securely. It is the authenticator that proves the client's identity. Because the authenticator can be used only once and has a short lifetime, the threat of an opponent stealing both the ticket and the authenticator for presentation later is countered. Later, if the client wants to apply to the TGS for a new service-granting ticket, it sends the reusable ticket-granting ticket plus a fresh authenticator.

The next two steps in the protocol repeat the last two. The TGS sends a service-granting ticket and a new session key to the client. The entire message is encrypted with the old session key, so only the client can recover the message. The ticket is

encrypted with a secret key shared only by the TGS and server V. The client now has a reusable service-granting ticket for V.

Each time user X wishes to use service V, the client can then send this ticket plus an authenticator to server V. The authenticator is encrypted with the new session key.

If mutual authentication is required, the server can reply with the value of the timestamp from the authenticator, incremented by 1, and encrypted in the session key. The client can decrypt this message to recover the incremented timestamp. Because the message was encrypted by the session key, the client is assured that it could have been created only by V. The contents of the message assures C that this is not a replay of an old reply.

Finally, at the conclusion of this process, the client and server share a secret key. This key can be used to encrypt future messages between the two or to exchange a new session key for that purpose.

Kerberos Realms and Multiple Kerberi

A full-service Kerberos environment consisting of a Kerberos server, a number of clients, and a number of application servers, requires the following:

1. The Kerberos server must have the user ID and password of all participating users in its database. All users are registered with the Kerberos server.
2. The Kerberos server must share a secret key with each server. All servers are registered with the Kerberos server.

Such an environment is referred to as a realm. Networks of clients and servers under different administrative organizations generally constitute different realms (see Figure 23.2). That is, it generally is not practical, or does not conform to administrative policy, to have users and servers in one administrative domain registered with a Kerberos server elsewhere. However, users in one realm may need access to servers in other realms, and some servers may be willing to provide service to users from other realms, provided that those users are authenticated.

Kerberos provides a mechanism for supporting such interrealm authentication. For two realms to support interrealm authentication, the Kerberos server in each interoperating realm shares a secret key with the server in the other realm. The two Kerberos servers are registered with each other.

The scheme requires that the Kerberos server in one realm trust the Kerberos server in the other realm to authenticate its users. Furthermore, the participating servers in the second realm must also be willing to trust the Kerberos server in the first realm.

With these ground rules in place, we can describe the mechanism as follows (see Figure 23.2): A user wishing service on a server in another realm needs a ticket for that server. The user's client follows the usual procedures to gain access to the local TGS then requests a ticket-granting ticket for a remote TGS (TGS in another realm). The client can then apply to the remote TGS for a service-granting ticket for the desired server in the realm of the remote TGS.

The ticket presented to the remote server indicates the realm in which the user was originally authenticated. The server chooses whether to honor the remote request.

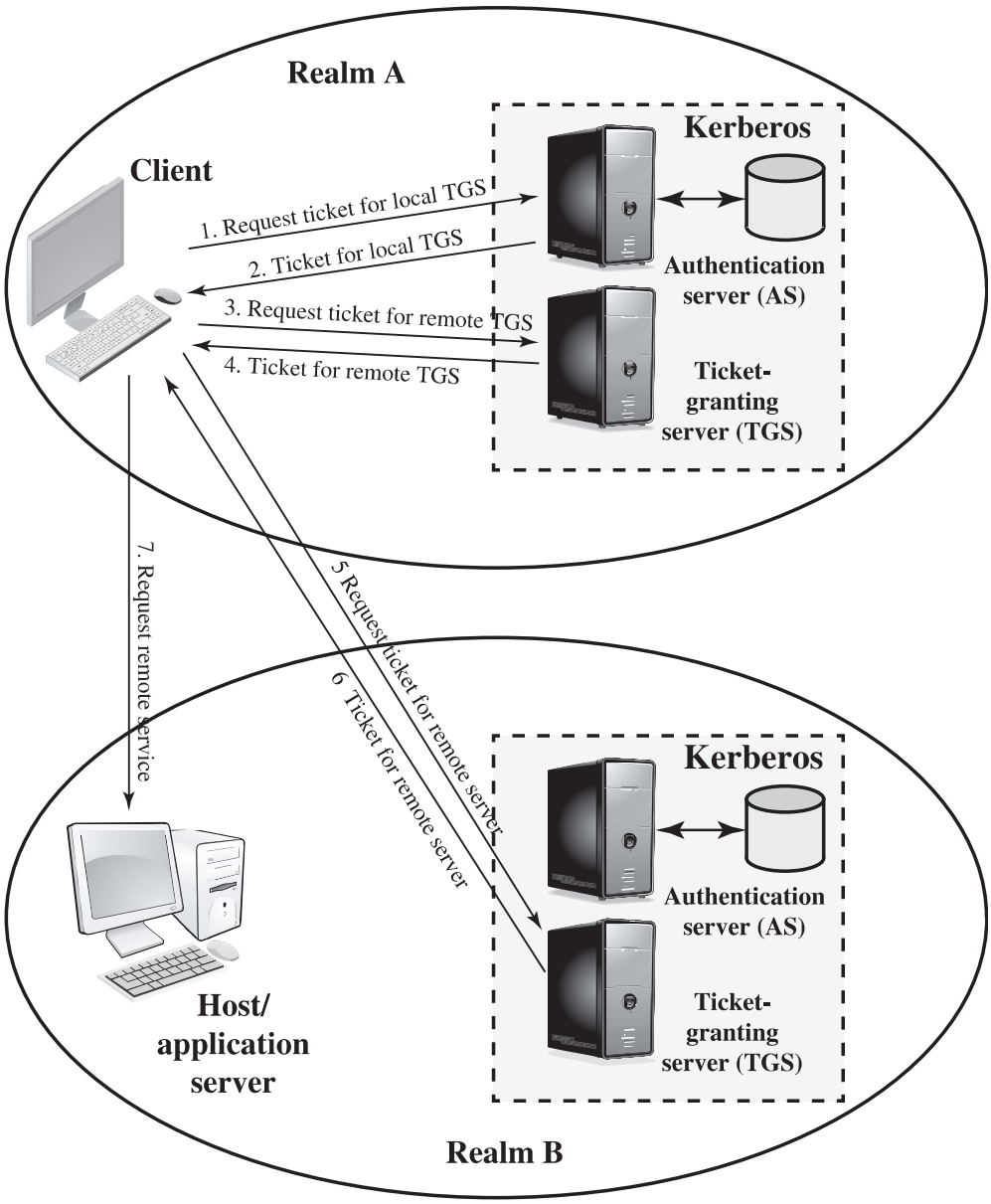


Figure 23.2 **Request for Service in Another Realm**

One problem presented by the foregoing approach is that it does not scale well to many realms. If there are N realms, then there must be $N(N-)/2$ secure key exchanges so that each realm can interoperate with all other Kerberos realms.

Version 4 and Version 5

The first version of Kerberos that was widely used was version 4, published in the late 1980s. An improved and extended version 5 was introduced in 1993, and updated in 2005. Kerberos version 5 is now widely implemented, including as part of Microsoft's Active Directory service, in most current UNIX and Linux systems, and in Apple's Mac OS X. It includes a number of improvements over version 4. First, in version 5, an encrypted message is tagged with an encryption algorithm identifier. This enables

users to configure Kerberos to use an algorithm other than DES, with the Advanced Encryption Standard (AES) now the default choice.

Version 5 also supports a technique known as authentication forwarding. Version 4 does not allow credentials issued to one client to be forwarded to some other host and used by some other client. Authentication forwarding enables a client to access a server and have that server access another server on behalf of the client. For example, a client issues a request to a print server that then accesses the client's file from a file server, using the client's credentials for access. Version 5 provides this capability.

Finally, version 5 supports a method for interrealm authentication that requires fewer secure key exchanges than in version 4.

Performance Issues

As client/server applications become more popular, larger client/server installations are appearing. A case can be made that the larger the scale of the networking environment, the more important it is to have logon authentication. But the question arises: What impact does Kerberos have on performance in a large-scale environment?

Fortunately, the answer is that there is very little performance impact if the system is properly configured. Keep in mind that tickets are reusable. Therefore, the amount of traffic needed for the granting ticket requests is modest. With respect to the transfer of a ticket for logon authentication, the logon exchange must take place anyway, so again the extra overhead is modest.

A related issue is whether the Kerberos server application requires a dedicated platform or can share a computer with other applications. It probably is not wise to run the Kerberos server on the same machine as a resource-intensive application such as a database server. Moreover, the security of Kerberos is best assured by placing the Kerberos server on a separate, isolated machine.

Finally, in a large system, is it necessary to go to multiple realms in order to maintain performance? Probably not. Rather, the motivation for multiple realms is administrative. If you have geographically separate clusters of machines, each with its own network administrator, then one realm per administrator may be convenient. However, this is not always the case.

23.2 X.509

Public-key certificates are mentioned briefly in Section 2.4. Recall that a certificate links a public key with the identity of the key's owner, with the whole block signed by a trusted third party. Typically, the third party is a **certificate authority (CA)** that is trusted by the user community, such as a government agency, financial institution, telecommunications company, or other trusted peak organization. A user can present his or her public key to the authority in a secure manner and obtain a certificate. The user can then publish the certificate, or send it to others. Anyone needing this user's public key can obtain the certificate and verify that it is valid by way of the attached trusted signature, provided they can verify the CA's public key. Figure 2.8 illustrates this process.

The X.509 ITU-T standard, also specified in RFC 5280 (*Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*, 2008), is the most widely accepted format for public-key certificates. X.509 certificates are used in most network security applications, including IP security (IPSEC), secure sockets layer (SSL), transport layer security (TLS), secure electronic transactions (SET), and S/MIME, as well as in eBusiness applications.

An X.509 certificate includes the elements shown in Figure 23.3a. Key elements include the key owning Subject's X.500 name and public-key information, the Period of validity dates, the CA's Issuer name, and their Signature that binds all this information together. Current X.509 certificates use the version 3 format that includes a general extension mechanism to provide more flexibility and to convey information needed in special circumstances. See [STAL17] for further information on the X.509 certificate format and elements.

One important extension, in the "Basic Constraints" set, specifies whether the certificate is that of a CA or not. A CA certificate is used only to sign other certificates. Otherwise, the certificate belongs to an "end-user" (or "end-entity"), and may be used for verifying server or client identities, signing or encrypting e-mail or other content, signing executable code, or other uses in applications such as those we listed above. The usage of any certificate's key can be restricted by including the "Key Usage" and "Extended Key Usage" extensions that specify a set of approved

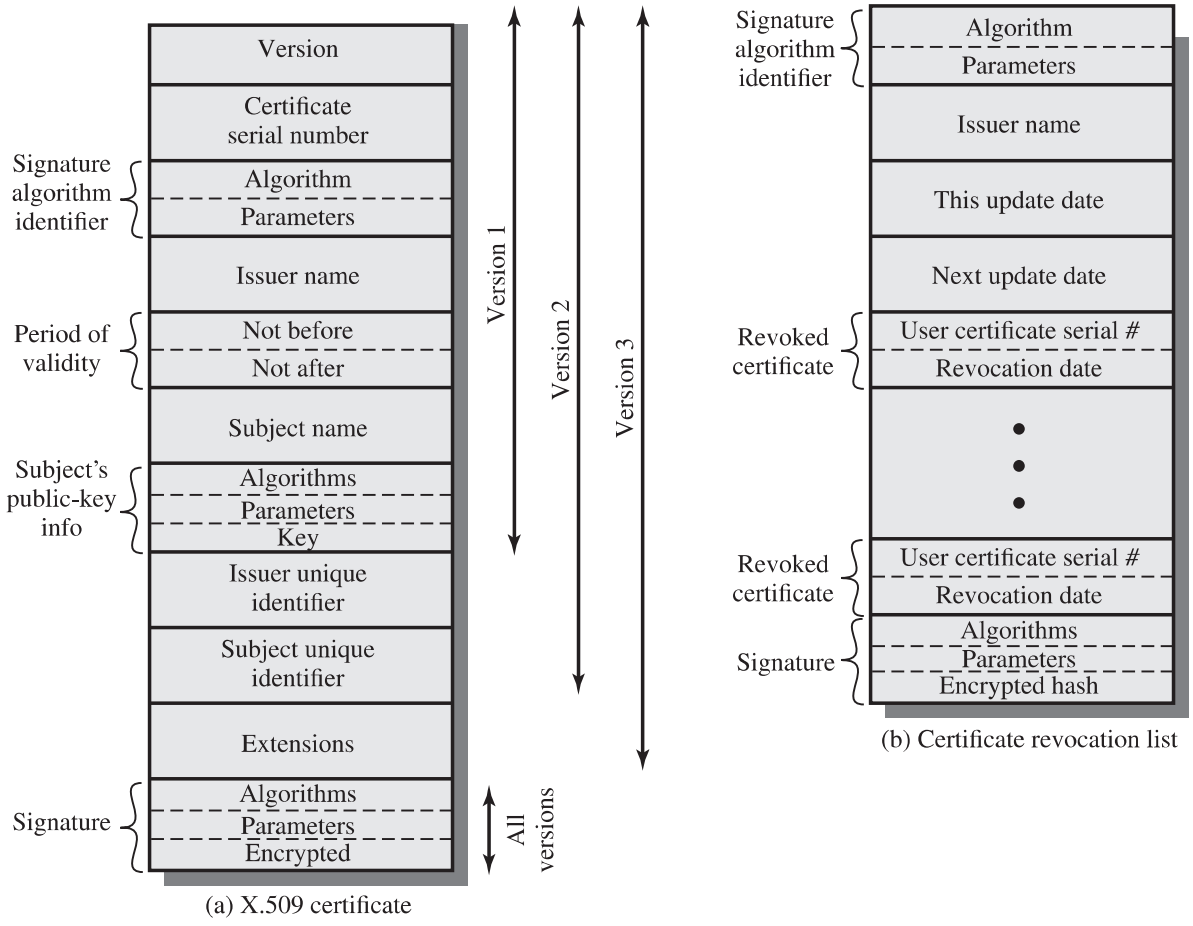


Figure 23.3 X.509 Formats

uses. "End-user" certificates are not permitted to sign other certificates, apart from the special case of proxy-certificates that we mention below.

The CA and "end user" certificates discussed above are the most common form of X.509 certificates. However, a number of specialized variants also exist, distinguished by particular element values or the presence of certain extensions. Variants include:

- **Conventional (long-lived) certificates** are the CA and "end user" certificates discussed above. They are typically issued for validity periods of months to years.
- **Short-lived certificates** are used to provide authentication for applications such as grid computing, while avoiding some of the overheads and limitations of conventional certificates [HSU98]. They have validity periods of hours to days, which limits the period of misuse if compromised. Because they are usually not issued by recognized CA's, there are issues with verifying them outside their issuing organization.
- **Proxy certificates** are now widely used to provide authentication for applications such as grid computing, while addressing some of the limitations of short-lived certificates. RFC 3820 (*Internet X.509 Public Key Infrastructure (PKI) Proxy Certificate Profile*, 2004) defines proxy certificates, which are identified by the presence of the "proxy certificate" extension. They allow an "end user" certificate to sign another certificate, which must be an extension of the existing certificate with a sub-set of their identity, validity period, and authorizations. They allow a user to easily create a credential to access resources in some environment, without needing to provide their full certificate and rights. There are other proposals to use proxy certificates as network access capability tickets, which authorize a user to access specific services with specific rights.
- **Attribute certificates** use a different certificate format, defined in RFC 5755 (*An Internet Attribute Certificate Profile for Authorization*, 2010), to link a user's identity to a set of attributes that are typically used for authorization and access control. A user may have a number of different attribute certificates, with different sets of attributes for different purposes, associated with their main conventional certificate. These attributes are defined in an "Attributes" extension. These extensions could also be included in a conventional certificate, but this is discouraged as being too inflexible. They may also be included in a proxy certificate, further restricting its use, and this is appropriate for some applications.

Before using any certificate, an application must check its validity, and ensure that it was not revoked before it expires. This may occur if the user wishes to cancel a key because it has been compromised, or because an upgrade in the user's software requires the generation of a new key.

The X.509 standard defines a certificate revocation list (CRL), signed by the issuer, that includes the elements shown in Figure 23.3b. Each revoked certificate entry contains a serial number of a certificate and the revocation date for that certificate. Because serial numbers are unique within a CA, the serial number is sufficient to identify the certificate. When an application receives a certificate, the X.509 standard states it should determine whether it has been revoked, by checking against

the current CRL for its issuing CA. However, due to the overheads in retrieving and storing these lists, very few applications actually do this. "The recent Heartbleed Open SSL bug, which has forced the revocation and replacement of very large numbers of server certificates, has dramatically highlighted deficiencies with the use of CRLs."

A more practical alternative is to use the RFC 6960 (*X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP*, 2013), to query the CA as to whether a specific certificate is valid. This lightweight protocol is increasingly used, including in recent versions of most common Web browsers. The "Authority Information Access" extension in a certificate can specify the address of the OCSP server to use, if the signing CA supports this protocol.

Originally, most X.509 certificates signed an MD5 hash of their contents. Unfortunately, research advances in creating MD5 collisions has led to the development of several techniques for forging new certificates for different identities that have the same hash, and hence can reuse the same signature, as an existing valid certificate [STEV07]. The Flame malware authors used this approach to forge what appeared to be a valid Microsoft code-signing certificate. This allowed the malware to remain undetected for more than 2 years before being identified in 2012. The use of MD5 was depreciated, and the SHA-1 hash algorithm recommended, in the 2000s. However the creation of SHA-1 collisions in 2017 means that, in turn, this algorithm is no longer considered secure. As of early 2017, most browsers now reject certificates using SHA-1 or MD5. The current requirement is to use one of the SHA-2 hash algorithms in certificates, with support for SHA-3 as an alternative likely soon.

23.3 PUBLIC-KEY INFRASTRUCTURE

RFC 4949 (*Internet Security Glossary, Version 2*, 2007) defines public-key infrastructure (PKI) as the set of hardware, software, people, policies, and procedures needed to create, manage, store, distribute, and revoke digital certificates based on asymmetric cryptography. The principal objective for developing a PKI is to enable secure, convenient, and efficient acquisition of public keys.

In order to verify a certificate, you need to know the public key of the signing CA. This could, in turn, be provided in another certificate, signed by a parent CA, with the CA's organized in a hierarchy. Eventually, however, you must reach the top of the hierarchy, and have a copy of the public key for that root CA. The X.509 standard describes a PKI model that originally assumed there would be a single internationally specified hierarchy of government regulated CAs. This did not happen. Instead, current X.509 PKI implementations come with a large list of CAs and their public keys, known as a "trust store." These CAs usually either directly sign "end-user" certificates, or sign a small number of Intermediate-CAs that in turn sign "end-user" certificates. Thus, all the hierarchies are very small, and all are equally trusted. Users and servers that want an automatically verified certificate must acquire it from one of these CAs. Alternatively, they can use either a "self-signed" certificate or a certificate signed by some other CA. However, in both these cases, such certificates will initially be recognized as "untrusted" and the user presented with stark warnings about accepting such certificates, even if they are actually legitimate.

There are many problems with this model of a PKI, and these have been known for many years [GUTM02, GRUS13]. Current implementations suffer from a number of critical issues. The first is the reliance on the user to make an informed decision when there is a problem verifying a certificate. Unfortunately, it is clear that most users do not understand what a certificate is and why there might be a problem. Hence they choose to accept a certificate, or not, for reasons that have little to do with their security, which may result in the compromise of their systems.

Another critical problem is the assumption that all of the CAs in the "trust store" are equally trusted, equally well-managed, and apply equivalent policies. This was dramatically illustrated by the compromise of the DigiNotar CA in 2011 that resulted in the fraudulent issue of certificates for many well-known organizations. It is widely believed these were used by the Iranian government to mount a "man-the-middle" attack on the secured communications of many of their citizens. As a consequence, the DigiNotar CA keys were removed from the "trust store" in many systems, and the company was declared bankrupt later that year. Another CA, Comodo, was also compromised in 2011, with a small number of fraudulent certificates issued.

A further concern is that different implementations, in the various Web browsers and operating systems, use different "trust stores," and hence present different security views to users.

Given these and other issues, several proposals exist to improve the practical handling of X.509 certificates. Some of these recognize that many applications do not require formal linking of a public key to a verified identity. In many Web applications, for example, all users really need is to know that if they visit the same secure site and are supplied with a certificate for it, that it is the same site and same key as when they previously visited. This is analogous to ensuring that if you visit the same physical store, you see the same company name and layout and staff as previously. And further, users want to know that it is the same site and same key as other users in other locations see.

The first of these, confirming continuity in time, can be provided by user's applications keeping a record of certificate details for all sites they visit, and checking against these on subsequent visits. Certificate pinning in applications can provide this feature, as is used in Google Chrome. The Firefox "Certificate Patrol" extension is another example of this approach.

The second, confirming continuity in space, requires the use of a number of widely separated "network notary servers" that keep records of certificates for all sites they view, that can be compared with a certificate provided to the user in any instance. The "Perspectives Project" is a practical implementation of this approach, which may be accessed using the Firefox "Perspectives" plugin. This also verifies the time history of certificates in use, thus providing both desired features for this approach. The "Google Certificate Catalog" and "Google Certificate Transparency" project are other examples of such notary servers.

In either of the above cases, identification of a different certificate and key to that seen at other times or places may well be an indication of attack or other problems. It may also simply be the result of certificates being updated as they

approach expiry, or of organizations incorrectly using multiple certificates and keys for the same, but replicated, server. These latter issues need to be managed by such extensions.

Public Key Infrastructure X.509 (PKIX)

The Internet Engineering Task Force (IETF) Public Key Infrastructure X.509 (PKIX) working group has been the driving force behind setting up a formal (and generic) model based on X.509 that is suitable for deploying a certificate-based architecture on the Internet. This section briefly describes the PKIX model. For more detail, see [STAL17].

Figure 23.4 shows the interrelationship among the key elements of the PKIX model. These elements include the **End entity** (e.g., user or server) for which the certificate for and the **Certificate authority** that issues the certificates. The CA's management functions may be further divided to include the **Registration authority (RA)** that handles end entity registration and the CRL issuer and Repository that manage CRLs.

PKIX identifies a number of management functions that potentially need to be supported by management protocols. These are indicated in Figure 23.4 and include user Registration, Initialization of key material, Certification in which a CA issues a certificate, Key pair recovery and update, Revocation request for a certificate, and Cross certification between CAs.

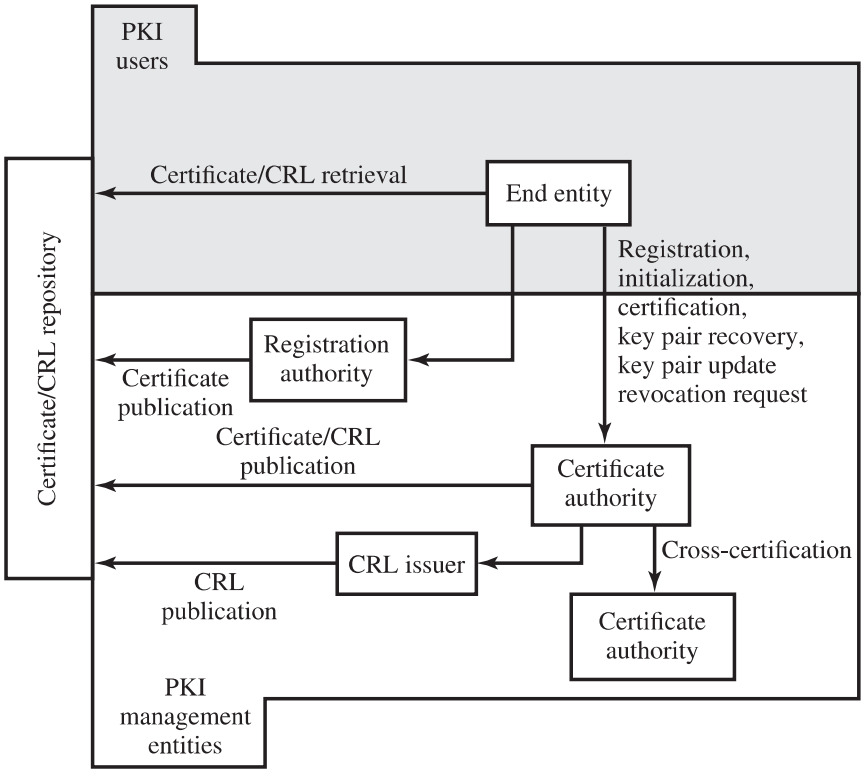


Figure 23.4 PKIX Architectural Model

23.4 KEY TERMS, REVIEW QUESTIONS, AND PROBLEMS

Key Terms

authentication server (AS)	Kerberos realm	ticket-granting ticket (TGT)
Certificate Authority (CA)	Public-Key Infrastructure (PKI)	ticket-granting server (TGS)
End entity	Registration authority (RA)	X.509
Kerberos		X.509 certificate

Review Questions

23.1 What information is carried by the ticket?
23.2 Describe the requirements of a full-service Kerberos environment.
23.3 What is the role of the authentication server in a Kerberos system?
23.4 How is authentication performed in a Kerberos system?
23.5 What are some extensions listed in the X.509 standard, and what are their uses?
23.6 What is the role of a CA in X.509?
23.7 What different types of X.509 certificates exist?
23.8 What alternatives exist to check that an X.509 certificate has not been revoked?
23.9 What are short-lived certificates?
23.10 How do most current X.509 implementations check the validity of signatures on a certificate?
23.11 What are some key problems with current public key infrastructure implementations?
23.12 List the key elements of the PKIX model.

Problems

23.1 CBC (cipher block chaining) has the property that if an error occurs in transmission of ciphertext block *CI*, then this error propagates to the recovered plaintext blocks *PI* and *PI* + 1. Version 4 of Kerberos uses an extension to CBC, called the propagating CBC (PCBC) mode. This mode has the property that an error in one ciphertext block is propagated to all subsequent decrypted blocks of the message, rendering each block useless. Thus, data encryption and integrity are combined in one operation. For PCBC, the input to the encryption algorithm is the XOR of the current plaintext block, the preceding cipher text block, and the preceding plaintext block:

$$C_n = E(K,[C_{n-1} \oplus P_{n-1} \oplus P_n])$$

On decryption, each ciphertext block is passed through the decryption algorithm. Then the output is XORed with the preceding ciphertext block and the preceding plaintext block.

a. Draw a diagram similar to those used in Chapter 20 to illustrate PCBC.
b. Use a Boolean equation to demonstrate that PCBC works.
c. Show that a random error in one block of ciphertext is propagated to all subsequent blocks of plaintext.

23.2 Suppose in PCBC mode, blocks *Ci* and *Ci* + 1 are interchanged during transmission. Show that this affects only the decrypted blocks *Pi* and *Pi* + 1, but not subsequent blocks.

23.3 Consider the details of the X.509 certificate shown below.

a. Identify the key elements in this certificate, including the owner's name and public key, its validity dates, the name of the CA that signed it, and the type and value of signature.

b. State whether this is a CA or end-user certificate, and why.

c. Indicate whether the certificate is valid or not, and why.

d. State whether there are any other obvious problems with the algorithms used in this certificate.

```
Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number: 3c:50:33:c2:f8:e7:5c:ca:07:c2:4e:83:f2:e8:0e:4f
    Signature Algorithm: md5WithRSAEncryption
    Issuer: O=VeriSign, Inc.,
           OU=VeriSign Trust Network,
           CN=VeriSign Class 1 CA Individual Persona Not Validated
    Validity
      Not Before: Jan 13 00:00:00 2000 GMT
      Not After : Mar 13 23:59:59 2000 GMT
    Subject: O=VeriSign, Inc.,
           OU=VeriSign Trust Network,
           OU=Persona Not Validated,
         OU=Digital ID Class 1 - Netscape
         CN=John Doe/Email=john.doe@adfa.edu.au
    Subject Public Key Info:
      Public Key Algorithm: rsaEncryption
      RSA Public Key: (512 bit)
        Modulus (512 bit):
            00:98:f2:89:c4:48:e1:3b:2c:c5:d1:48:67:80:53:
            d8:eb:4d:4f:ac:31:a9:fd:11:68:94:ba:44:d8:48:
            46:0d:fc:5c:6d:89:47:3f:9f:d0:c0:6d:3e:9a:8e:
            ec:82:21:48:9b:b9:78:cf:aa:09:61:92:f6:d1:cf:
            45:ca:ea:8f:df
        Exponent: 65537 (0x10001)
    X509v3 extensions:
      X509v3 Basic Constraints:
        CA:FALSE
      X509v3 Certificate Policies:
        Policy: 2.16.840.1.113733.1.7.1.1
          CPS: https://www.verisign.com/CPS
      X509v3 CRL Distribution Points:
        URI:http://crl.verisign.com/class1.crl
```

```
Signature Algorithm: md5WithRSAEncryption
  5a:71:77:c2:ce:82:26:02:45:41:a5:11:68:d6:99:f0:4c:ce:
  7a:ce:80:44:f4:a3:1a:72:43:e9:dc:e1:1a:9b:ec:64:f7:ff:
  21:f2:29:89:d6:61:e5:39:bd:04:e7:e5:3d:7b:14:46:d6:eb:
  8e:37:b0:cb:ed:38:35:81:1f:40:57:57:58:a5:c0:64:ef:55:
  59:c0:79:75:7a:54:47:6a:37:b2:6c:23:6b:57:4d:62:2f:94:
  d3:aa:69:9d:3d:64:43:61:a7:a3:e0:b8:09:ac:94:9b:23:38:
  e8:1b:0f:e5:1b:6e:e2:fa:32:86:f0:c4:0b:ed:89:d9:16:e4:
  a7:77
```

23.4 Using your Web browser, visit any secure Web site (i.e., one whose URL starts with "https"). Examine the details of the X.509 certificate used by that site. This is usually accessible by selecting the padlock symbol. Answer the same questions as for Problem 23.3.

23.5 Now access the "Trust Store" (list of certificates) used by your Web browser. This is usually accessed via its Preference settings. Access the list of Certificate Authority certificates used by the browser. Pick one, examine the details of its X.509 certificate, and answer the same questions as for Problem 23.3.

CHAPTER 24

WIRELESS NETWORK SECURITY

LEARNING OBJECTIVES

After studying this chapter, you should be able to:

- Present an overview of security threats and countermeasures for wireless networks.
- Understand the unique security threats posed by the use of mobile devices with enterprise networks.
- Describe the principal elements in a mobile device security strategy.
- Understand the essential elements of the IEEE 802.11 wireless LAN standard.
- Summarize the various components of the IEEE 802.11i wireless LAN security architecture.

Wireless networks and communication links have become pervasive for both personal and organizational communications. A wide variety of technologies and network types have been adopted, including Wi-Fi, Bluetooth, WiMAX, ZigBee, and cellular technologies. Although the security threats and countermeasures discussed throughout this text apply to wireless networks and communications links, there are some unique aspects to the wireless environment.

This chapter begins with a general overview of wireless security issues. We then focus on the relatively new area of mobile device security, examining threats and countermeasures for mobile devices used in the enterprise. Then, we look at the IEEE 802.11i standard for wireless LAN security. This standard is part of IEEE 802.11, also referred to as Wi-Fi. We begin the discussion with an overview of IEEE 802.11, then we look in some detail at IEEE 802.11i.

24.1 WIRELESS SECURITY

Wireless networks, and the wireless devices that use them, introduce a host of security problems over and above those found in wired networks. Some of the key factors contributing to the higher security risk of wireless networks compared to wired networks include the following [MA10]:

- **Channel:** Wireless networking typically involves broadcast communications, which is far more susceptible to eavesdropping and jamming than wired networks. Wireless networks are also more vulnerable to active attacks that exploit vulnerabilities in communications protocols.
- **Mobility:** Wireless devices are, in principal and usually in practice, far more portable and mobile than wired devices. This mobility results in a number of risks, described subsequently.
- **Resources:** Some wireless devices, such as smartphones and tablets, have sophisticated operating systems but limited memory and processing resources with which to counter threats, including denial of service and malware.

- **Accessibility:** Some wireless devices, such as sensors and robots, may be left unattended in remote and/or hostile locations. This greatly increases their vulnerability to physical attacks.

In simple terms, the wireless environment consists of three components that provide point of attack (see Figure 24.1). The wireless client can be a mobile phone, a Wi-Fi enabled laptop or tablet, a wireless sensor, a Bluetooth device, and so on. The wireless access point provides a connection to the network or service. Examples of access points are mobile phone towers, Wi-Fi hot spots, and wireless access points to wired local or wide-area networks. The transmission medium, which carries the radio waves for data transfer, is also a source of vulnerability.

Wireless Network Threats

[CHOI08] lists the following security threats to wireless networks:

- **Accidental association:** Company wireless LANs or wireless access points to wired LANs in close proximity (e.g., in the same or neighboring buildings) may create overlapping transmission ranges. A user intending to connect to one LAN may unintentionally lock on to a wireless access point from a neighboring network. Although the security breach is accidental, it nevertheless exposes resources of one LAN to the accidental user.
- **Malicious association:** In this situation, a wireless device is configured to appear to be a legitimate access point, enabling the operator to steal passwords from legitimate users then penetrate a wired network through a legitimate wireless access point.
- **Ad hoc networks:** These are peer-to-peer networks between wireless computers with no access point between them. Such networks can pose a security threat due to a lack of a central point of control.
- **Nontraditional networks:** Nontraditional networks and links, such as personal network Bluetooth devices, barcode readers, and handheld PDAs pose a security risk both in terms of eavesdropping and spoofing.
- **Identity theft (MAC spoofing):** This occurs when an attacker is able to eavesdrop on network traffic and identify the MAC address of a computer with network privileges.
- **Man-in-the middle attacks:** This type of attack was described in Chapter 21 in the context of the Diffie-Hellman key exchange protocol. In a broader sense, this attack involves persuading a user and an access point to believe that they are talking to each other, when in fact the communication is going through an

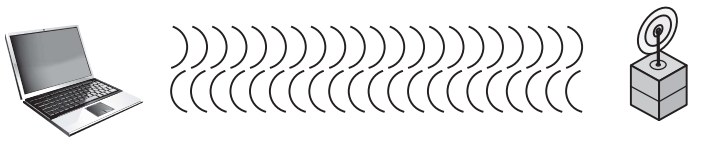


Figure 24.1 Wireless Networking Components

intermediate attacking device. Wireless networks are particularly vulnerable to such attacks.

- **Denial of service (DoS):** This type of attack was discussed in detail in Chapter 7. In the context of a wireless network, a DoS attack occurs when an attacker continually bombards a wireless access point, or some other accessible wireless port, with various protocol messages designed to consume system resources. The wireless environment lends itself to this type of attack, because it is so easy for the attacker to direct multiple wireless messages at the target.
- **Network injection:** A network injection attack targets wireless access points that are exposed to nonfiltered network traffic, such as routing protocol messages or network management messages. An example of such an attack is one in which bogus reconfiguration commands are used to affect routers and switches to degrade network performance.

Wireless Security Measures

Following [CHOI08], we can group wireless security measures into those dealing with wireless transmissions, wireless access points, and wireless networks (consisting of wireless routers and endpoints).

Securing Wireless Transmissions The principal threats to wireless transmission are eavesdropping, altering or inserting messages, and disruption. To deal with eavesdropping, two types of countermeasures are appropriate:

- **Signal-hiding techniques:** Organizations can take a number of measures to make it more difficult for an attacker to locate their wireless access points, including turning off service set identifier (SSID) broadcasting by wireless access points; assigning cryptic names to SSIDs; reducing signal strength to the lowest level that still provides requisite coverage; and locating wireless access points in the interior of the building, away from windows and exterior walls. Greater security can be achieved by the use of directional antennas and of signal-shielding techniques.
- **Encryption:** Encryption of all wireless transmission is effective against eavesdropping to the extent that the encryption keys are secured.

The use of encryption and authentication protocols is the standard method of countering attempts to alter or insert transmissions.

The methods discussed in Chapter 7 for dealing with denial of service apply to wireless transmissions. Organizations can also reduce the risk of unintentional DoS attacks. Site surveys can detect the existence of other devices using the same frequency range, to help determine where to locate wireless access points. Signal strengths can be adjusted and shielding used in an attempt to isolate a wireless environment from competing nearby transmissions.

Securing Wireless Access Points The main threat involving wireless access points is unauthorized access to the network. The principal approach for preventing such access is the IEEE 802.1X standard for port-based network access control. The standard provides an authentication mechanism for devices wishing to attach to a

LAN or wireless network. The use of 802.1X can prevent rogue access points and other unauthorized devices from becoming insecure backdoors.

Section 24.3 provides an introduction to 802.1X.

Securing Wireless Networks [CHOI08] recommends the following techniques for wireless network security:

1. Use encryption. Wireless routers are typically equipped with built-in encryption mechanisms for router-to-router traffic.
2. Use anti-virus and anti-spyware software, and a firewall. These facilities should be enabled on all wireless network endpoints.
3. Turn off identifier broadcasting. Wireless routers are typically configured to broadcast an identifying signal so that any device within range can learn of the router's existence. If a network is configured so authorized devices know the identity of routers, this capability can be disabled to thwart attackers.
4. Change the identifier on your router from the default. Again, this measure thwarts attackers who will attempt to gain access to a wireless network using default router identifiers.
5. Change your router's pre-set password for administration. This is another prudent step.
6. Allow only specific computers to access your wireless network. A router can be configured to only communicate with approved MAC addresses. Of course, MAC addresses can be spoofed, so this is just one element of a security strategy.

24.2 MOBILE DEVICE SECURITY

Prior to the widespread use of smartphones, the dominant paradigm for computer and network security in organizations was as follows. Corporate IT was tightly controlled. User devices were typically limited to Windows PCs. Business applications were controlled by IT and either run locally on endpoints or on physical servers in data centers. Network security was based upon clearly defined perimeters that separated trusted internal networks from the untrusted Internet. Today, there have been massive changes in each of these assumptions. An organization's networks must accommodate the following:

- **Growing use of new devices:** Organizations are experiencing significant growth in employee's use of mobile devices. In many cases, employees are allowed to use a combination of endpoint devices as part of their day-to-day activities.
- **Cloud-based applications:** Applications no longer run solely on physical servers in corporate data centers. Quite the opposite, applications can run anywhere—on traditional physical servers, on mobile virtual servers, or in the cloud. Additionally, end users can now take advantage of a wide variety of cloud-based applications and IT services for personal and professional use. Facebook can be used for an employee's personal profile or as a component

of a corporate marketing campaign. Employees depend upon Skype to speak with friends abroad or for legitimate business video conferencing. Dropbox and Box can be used to distribute documents between corporate and personal devices for mobility and user productivity.

- **De-perimeterization:** Given new device proliferation, application mobility, and cloud-based consumer and corporate services, the notion of a static network perimeter is all but gone. Now there are a multitude of network perimeters around devices, applications, users, and data. These perimeters have also become quite dynamic as they must adapt to various environmental conditions such as user role, device type, server virtualization mobility, network location, and time-of-day.
- **External business requirements:** The enterprise must also provide guests, third-party contractors, and business partners network access using various devices from a multitude of locations.

The central element in all of these changes is the mobile computing device. Mobile devices have become an essential element for organizations as part of the overall network infrastructure. Mobile devices such as smartphones, tablets, and portable USB storage devices provide increased convenience for individuals as well as the potential for increased productivity in the workplace. Because of their widespread use and unique characteristics, security for mobile devices is a pressing and complex issue. In essence, an organization needs to implement a security policy through a combination of security features built into the mobile devices and additional security controls provided by network components that regulate the use of the mobile devices.

Security Threats

Mobile devices need additional, specialized protection measures beyond those implemented for other client devices, such as desktop and laptop devices that are used only within the organization's facilities and on the organization's networks. NIST SP 800-124 (*Guidelines for Managing the Security of Mobile Devices in the Enterprise*, June 2013) lists seven major security concerns for mobile devices. We examine each of these in turn.

Lack of Physical Security Controls Mobile devices are typically under the complete control of the user, and are used and kept in a variety of locations outside the organization's control, including off premises. Even if a device is required to remain on premises, the user may move the device within the organization between secure and non secured locations. Thus, theft and tampering are realistic threats.

The security policy for mobile devices must be based on the assumption that any mobile device may be stolen or at least accessed by a malicious party. The threat is twofold: A malicious party may attempt to recover sensitive data from the device itself, or may use the device to gain access to the organization's resources.

Use of Untrusted Mobile Devices In addition to company-issued and company-controlled mobile devices, virtually all employees will have personal smartphones and/or tablets. The organization must assume that these devices are not trustworthy.

That is, the devices may not employ encryption and either the user or a third party may have installed a bypass to the built-in restrictions on security, operating system use, and so on.

Use of Untrusted Networks If a mobile device is used on premises, it can connect to organization resources over the organization's own in-house wireless networks. However, for off-premises use, the user will typically access organizational resources via Wi-Fi or cellular access to the Internet and from the Internet to the organization. Thus, traffic that includes an off-premises segment is potentially susceptible to eavesdropping or man-in-the-middle types of attacks. Thus, the security policy must be based on the assumption that the networks between the mobile device and the organization are not trustworthy.

Use of Untrusted Applications By design, it is easy to find and install third-party applications on mobile devices. This poses the obvious risk of installing malicious software. An organization has several options for dealing with this threat, as described subsequently.

Interaction with other Systems A common feature found on smartphones and tablets is the ability to automatically synchronize data, apps, contacts, photos, and so on with other computing devices and with cloud-based storage. Unless an organization has control of all the devices involved in synchronization, there is considerable risk of the organization's data being stored in an unsecured location, plus the risk of the introduction of malware.

Use of Untrusted Content Mobile devices may access and use content that other computing devices do not encounter. An example is the Quick Response (QR) code, which is a two-dimensional barcode. QR codes are designed to be captured by a mobile device camera and used by the mobile device. The QR code translates to a URL, so a malicious QR code could direct the mobile device to malicious Websites.

Use of Location Services The GPS capability on mobile devices can be used to maintain a knowledge of the physical location of the device. While this feature might be useful to an organization as part of a presence service, it creates security risks. An attacker can use the location information to determine where the device and user are located, which may be of use to the attacker.

Mobile Device Security Strategy

With the threats listed in the preceding discussion in mind, we outline the principal elements of a mobile device security strategy. They fall into three categories: device security, client/server traffic security, and barrier security (see Figure 24.2).

Device Security A number of organizations will supply mobile devices for employee use and pre-configure those devices to conform to the enterprise security policy. However, many organizations will find it convenient or even necessary to adopt a bring-your-own-device (BYOD) policy that allows the personal mobile devices of employees to have access to corporate resources. IT managers should be able to inspect each device before allowing network access. IT will want to establish

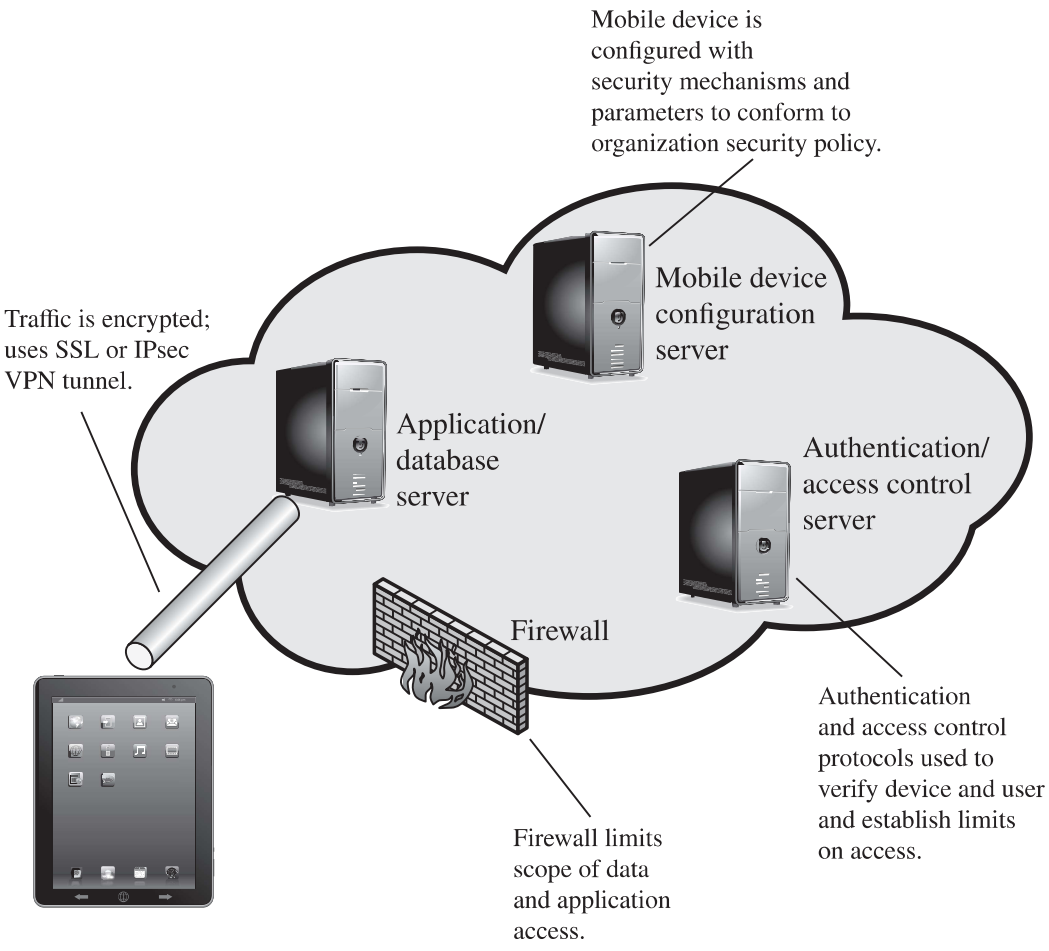


Figure 24.2 **Mobile Device Security Elements**

configuration guidelines for operating systems and applications. For example, "rooted" or "jail-broken" devices are not permitted on the network, and mobile devices cannot store corporate contacts on local storage. Whether a device is owned by the organization or BYOD, the organization should configure the device with security controls, including the following:

- Enable auto-lock, which causes the device to lock if it has not been used for a given amount of time, requiring the user to re-enter a four-digit PIN or a password to re-activate the device.
- Enable password or PIN protection. The PIN or password is needed to unlock the device. In addition, it can be configured so that e-mail and other data on the device are encrypted using the PIN or password and can only be retrieved with the PIN or password.
- Avoid using auto-complete features that remember user names or passwords.
- Enable remote wipe.
- Ensure that SSL protection is enabled, if available.
- Make sure that software, including operating systems and applications, is up to date.

- Install antivirus software as it becomes available.
- Sensitive data should be prohibited from storage on the mobile device or it should be encrypted.
- IT staff should also have the ability to remotely access devices, wipe all data of the device, then disable the device in the event of loss or theft.
- The organization may prohibit all installation of third-party applications, implement whitelisting to prohibit installation of all unapproved applications, or implement a secure sandbox that isolates the organization's data and applications from all other data and applications on the mobile device. Any application that is on an approved list should be accompanied by a digital signature and a public-key certificate from an approved authority.
- The organization can implement and enforce restrictions on what devices can synchronize and on the use of cloud-based storage.
- To deal with the threat of untrusted content, security responses can include training of personnel on the risks inherent in untrusted content and disabling camera use on corporate mobile devices.
- To counter the threat of malicious use of location services, the security policy can dictate that such service is disabled on all mobile devices.

Traffic Security Traffic security is based on the usual mechanisms for encryption and authentication. All traffic should be encrypted and travel by secure means, such as SSL or IPv6. Virtual private networks (VPNs) can be configured so all traffic between the mobile device and the organization's network is via a VPN.

A strong authentication protocol should be used to limit the access from the device to the resources of the organization. Often, a mobile device has a single device-specific authenticator, because it is assumed that the device has only one user. A preferable strategy is to have a two-layer authentication mechanism, which involves authenticating the device and then authenticating the user of the device.

Barrier Security The organization should have security mechanisms to protect the network from unauthorized access. The security strategy can also include firewall policies specific to mobile device traffic. Firewall policies can limit the scope of data and application access for all mobile devices. Similarly, intrusion detection and intrusion prevention systems can be configured to have tighter rules for mobile device traffic.

24.3 IEEE 802.11 WIRELESS LAN OVERVIEW

IEEE 802 is a committee that has developed standards for a wide range of local area networks (LANs). In 1990, the IEEE 802 Committee formed a new working group, IEEE 802.11, with a charter to develop a protocol and transmission specifications for wireless LANs (WLANs). Since that time, the demand for WLANs at different frequencies and data rates has exploded. Keeping pace with this demand, the IEEE 802.11 working group has issued an ever-expanding list of standards. Table 24.1 briefly defines key terms used in the IEEE 802.11 standard.

Table 24.1 IEEE 802.11 Terminology

Access point (AP)	Any entity that has station functionality and provides access to the distribution system via the wireless medium for associated stations
Basic service set (BSS)	A set of stations controlled by a single coordination function
Coordination function	The logical function that determines when a station operating within a BSS is permitted to transmit and may be able to receive PDUs
Distribution system (DS)	A system used to interconnect a set of BSSs and integrated LANs to create an ESS
Extended service set (ESS)	A set of one or more interconnected BSSs and integrated LANs that appear as a single BSS to the LLC layer at any station associated with one of these BSSs
MAC protocol data unit (MPDU)	The unit of data exchanged between two peer MAC entities using the services of the physical layer
MAC service data unit (MSDU)	Information that is delivered as a unit between MAC users
Station	Any device that contains an IEEE 802.11 conformant MAC and physical layer

The Wi-Fi Alliance

The first 802.11 standard to gain broad industry acceptance was 802.11b. Although 802.11b products are all based on the same standard, there is always a concern whether products from different vendors will successfully interoperate. To meet this concern, the Wireless Ethernet Compatibility Alliance (WECA), an industry consortium, was formed in 1999. This organization, subsequently renamed the Wi-Fi (Wireless Fidelity) Alliance, created a test suite to certify interoperability for 802.11b products. The term used for certified 802.11b products is *Wi-Fi*. Wi-Fi certification has been extended to 802.11g products. The Wi-Fi Alliance has also developed a certification process for 802.11a products, called *Wi-Fi5*. The Wi-Fi Alliance is concerned with a range of market areas for WLANs, including enterprise, home, and hot spots.

More recently, the Wi-Fi Alliance has developed certification procedures for IEEE 802.11 security standards, referred to as Wi-Fi Protected Access (WPA). The most recent version of WPA, known as WPA2, incorporates all of the features of the IEEE 802.11i WLAN security specification.

IEEE 802 Protocol Architecture

Before proceeding, we need to briefly preview the IEEE 802 protocol architecture. IEEE 802.11 standards are defined within the structure of a layered set of protocols. This structure, used for all IEEE 802 standards, is illustrated in Figure 24.3.

Physical Layer The lowest layer of the IEEE 802 reference model is the **physical layer**, which includes such functions as encoding/decoding of signals and

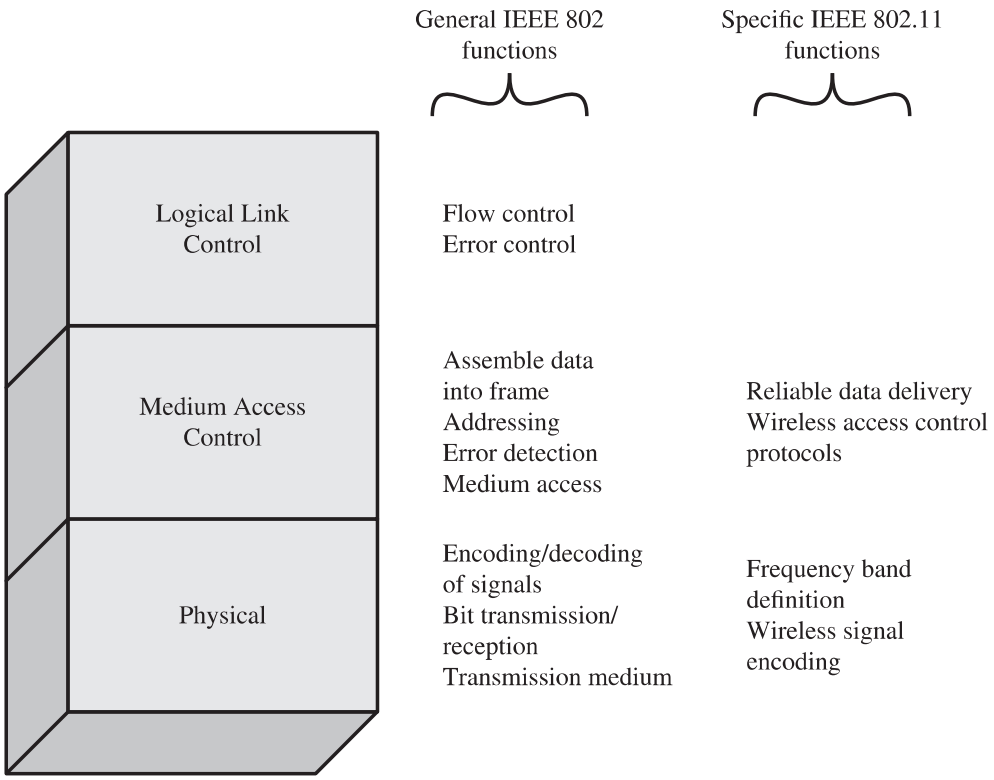


Figure 24.3 IEEE 802.11 Protocol Stack

bit transmission/reception. In addition, the physical layer includes a specification of the transmission medium. In the case of IEEE 802.11, the physical layer also defines frequency bands and antenna characteristics.

MEDIUM ACCESS CONTROL All LANs consist of collections of devices that share the network's transmission capacity. Some means of controlling access to the transmission medium is needed to provide an orderly and efficient use of that capacity. This is the function of a **medium access control (MAC)** layer. The MAC layer receives data from a higher-layer protocol, typically the logical link control (LLC) layer, in the form of a block of data known as the **MAC service data unit (MSDU)**. In general, the MAC layer performs the following functions:

- On transmission, assemble data into a frame, known as a **MAC protocol data unit (MPDU)** with address and error-detection fields.
- On reception, disassemble frame, and perform address recognition and error detection.
- Govern access to the LAN transmission medium.

The exact format of the MPDU differs somewhat for the various MAC protocols in use. In general, all of the MPDUs have a format similar to that of Figure 24.4. The fields of this frame are as follows:

- **MAC Control:** This field contains any protocol control information needed for the functioning of the MAC protocol. For example, a priority level could be indicated here.

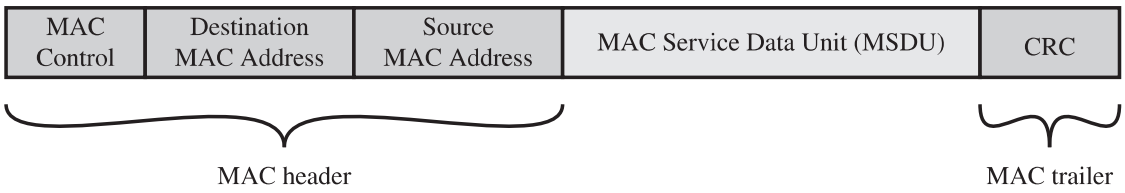


Figure 24.4 General IEEE 802 MPDU Format

- **Destination MAC Address:** The destination physical address on the LAN for this MPDU.
- **Source MAC Address:** The source physical address on the LAN for this MPDU.
- **MAC Service Data Unit:** The data from the next higher layer.
- **CRC:** The cyclic redundancy check field, also known as the Frame Check Sequence (FCS) field. This is an error-detecting code, such as that which is used in other data-link control protocols. The CRC is calculated based on the bits in the entire MPDU. The sender calculates the CRC and adds it to the frame. The receiver performs the same calculation on the incoming MPDU and compares that calculation to the CRC field in that incoming MPDU. If the two values do not match, then one or more bits have been altered in transit.

The fields preceding the MSDU field are referred to as the **MAC header**, and the field following the MSDU field is referred to as the **MAC trailer**. The header and trailer contain control information that accompany the data field and that are used by the MAC protocol.

Logical Link Control In most data-link control protocols, the data-link protocol entity is responsible not only for detecting errors using the CRC, but for recovering from those errors by retransmitting damaged frames. In the LAN protocol architecture, these two functions are split between the MAC and LLC layers. The MAC layer is responsible for detecting errors and discarding any frames that contain errors. The LLC layer optionally keeps track of which frames have been successfully received and retransmits unsuccessful frames.

IEEE 802.11 Network Components and Architectural Model

Figure 24.5 illustrates the model developed by the 802.11 working group. The smallest building block of a wireless LAN is a **basic service set (BSS)**, which consists of wireless stations executing the same MAC protocol and competing for access to the same shared wireless medium. A BSS may be isolated or it may connect to a backbone **distribution system (DS)** through an **access point (AP)**. The AP functions as a bridge and a relay point. In a BSS, client stations do not communicate directly with one another. Rather, if one station in the BSS wants to communicate with another station in the same BSS, the MAC frame is first sent from the originating station to the AP, then from the AP to the destination station. Similarly, a MAC frame from a station in the BSS to a remote station is sent from the local station to the AP then relayed by the AP over the DS on its way to the destination station. The BSS generally corresponds to what is referred to as a cell in the literature. The DS can be a switch, a wired network, or a wireless network.

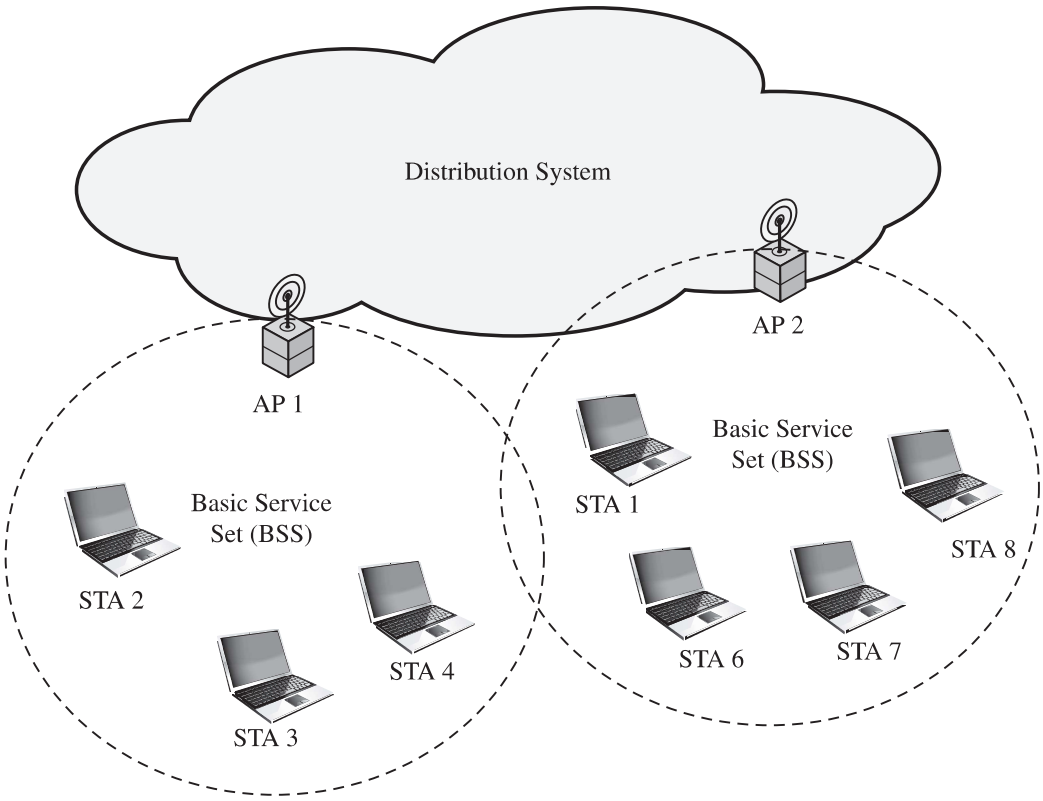


Figure 24.5 **IEEE 802.11 Extended Service Set**

When all the stations in the BSS are mobile stations that communicate directly with one another (not using an AP) the BSS is called an **independent BSS (IBSS)**. An IBSS is typically an ad hoc network. In an IBSS, the stations all communicate directly, and no AP is involved.

A simple configuration is shown in Figure 24.5, in which each station belongs to a single BSS; that is, each station is within wireless range only of other stations within the same BSS. It is also possible for two BSSs to overlap geographically, so that a single station could participate in more than one BSS. Furthermore, the association between a station and a BSS is dynamic. Stations may turn off, come within range, and go out of range.

An **extended service set (ESS)** consists of two or more basic service sets interconnected by a distribution system. The ESS appears as a single logical LAN to the LLC level.

IEEE 802.11 Services

IEEE 802.11 defines nine services that need to be provided by the wireless LAN to achieve functionality equivalent to that which is inherent to wired LANs. Table 24.2 lists the services and indicates two ways of categorizing them.

1. The service provider can be either the station or the DS. Station services are implemented in every 802.11 station, including AP stations. Distribution services are provided between BSSs; these services may be implemented in an AP, or in another special-purpose device attached to the distribution system.

Table 24.2 IEEE 802.11 Services

Service	Provider	Used to support
Association	Distribution system	MSDU delivery
Authentication	Station	LAN access and security
Deauthentication	Station	LAN access and security
Disassociation	Distribution system	MSDU delivery
Distribution	Distribution system	MSDU delivery
Integration	Distribution system	MSDU delivery
MSDU delivery	Station	MSDU delivery
Privacy	Station	LAN access and security
Reassociation	Distribution system	MSDU delivery

2. Three of the services are used to control IEEE 802.11 LAN access and confidentiality. Six of the services are used to support delivery of MSDUs between stations. If the MSDU is too large to be transmitted in a single MPDU, it may be fragmented and transmitted in a series of MPDUs.

Following the IEEE 802.11 document, we next discuss the services in an order designed to clarify the operation of an IEEE 802.11 ESS network. MSDU delivery, which is the basic service, has already been mentioned. Services related to security are introduced in Section 24.3.

Distribution of Messages Within a DS The two services involved with the distribution of messages within a DS are distribution and integration. Distribution is the primary service used by stations to exchange MPDUs when the MPDUs must traverse the DS to get from a station in one BSS to a station in another BSS. For example, suppose a frame is to be sent from station 2 (STA 2) to station 7 (STA 7) in Figure 24.5. The frame is sent from STA 2 to AP 1, which is the AP for this BSS. The AP gives the frame to the DS, which has the job of directing the frame to the AP associated with STA 7 in the target BSS. AP 2 receives the frame and forward it to STA 7. How the message is transported through the DS is beyond the scope of the IEEE 802.11 standard.

If the two stations that are communicating are within the same BSS, then the distribution service logically goes through the single AP of that BSS.

The integration service enables transfer of data between a station on an IEEE 802.11 LAN and a station on an integrated IEEE 802.x LAN. The term *integrated* refers to a wired LAN that is physically connected to the DS and whose stations may be logically connected to an IEEE 802.11 LAN via the integration service. The integration service takes care of any address translation and media conversion logic required for the exchange of data.

Association-Related Services The primary purpose of the MAC layer is to transfer MSDUs between MAC entities; this purpose is fulfilled by the distribution service. For that service to function, it requires information about stations

within the ESS that is provided by the association-related services. Before the distribution service can deliver data to or accept data from a station, that station must be *associated*. Before looking at the concept of association, we need to describe the concept of mobility. The standard defines three transition types, based on mobility:

- **No transition:** A station of this type is either stationary, or moves only within the direct communication range of the communicating stations of a single BSS.
- **BSS transition:** This is defined as a station movement from one BSS to another BSS within the same ESS. In this case, delivery of data to the station requires that the addressing capability be able to recognize the new location of the station.
- **ESS transition:** This is defined as a station movement from a BSS in one ESS to a BSS within another ESS. This case is supported only in the sense that the station can move. Maintenance of upper-layer connections supported by 802.11 cannot be guaranteed. In fact, disruption of service is likely to occur.

To deliver a message within a DS, the distribution service needs to know where the destination station is located. Specifically, the DS needs to know the identity of the AP to which the message should be delivered in order for that message to reach the destination station. To meet this requirement, a station must maintain an association with the AP within its current BSS. Three services relate to this requirement:

- **Association:** Establishes an initial association between a station and an AP. Before a station can transmit or receive frames on a wireless LAN, its identity and address must be known. For this purpose, a station must establish an association with an AP within a particular BSS. The AP can then communicate this information to other APs within the ESS to facilitate routing and delivery of addressed frames.
- **Reassociation:** Enables an established association to be transferred from one AP to another, allowing a mobile station to move from one BSS to another.
- **Disassociation:** A notification from either a station or an AP that an existing association is terminated. A station should give this notification before leaving an ESS or shutting down. However, the MAC management facility protects itself against stations that disappear without notification.

24.4 IEEE 802.11i WIRELESS LAN SECURITY

There are two characteristics of a wired LAN that are not inherent in a wireless LAN.

1. In order to transmit over a wired LAN, a station must be physically connected to the LAN. On the other hand, with a wireless LAN, any station within radio range of the other devices on the LAN can transmit. In a sense, there is a form of authentication with a wired LAN, in that it requires some positive and presumably observable action to connect a station to a wired LAN.

2. Similarly, in order to receive a transmission from a station that is part of a wired LAN, the receiving station also must be attached to the wired LAN. On the other hand, with a wireless LAN, any station within radio range can receive. Thus, a wired LAN provides a degree of privacy, limiting reception of data to stations connected to the LAN.

These differences between wired and wireless LANs suggest the increased need for robust security services and mechanisms for wireless LANs. The original 802.11 specification included a set of security features for privacy and authentication that were quite weak. For privacy, 802.11 defined the **Wired Equivalent Privacy (WEP)** algorithm. The privacy portion of the 802.11 standard contained major weaknesses. Subsequent to the development of WEP, the 802.11i task group has developed a set of capabilities to address the WLAN security issues. In order to accelerate the introduction of strong security into WLANs, the Wi-Fi Alliance promulgated **Wi-Fi Protected Access (WPA)** as a Wi-Fi standard. WPA is a set of security mechanisms that eliminates most 802.11 security issues and was based on the current state of the 802.11i standard. The final form of the 802.11i standard is referred to as **Robust Security Network (RSN)**. The Wi-Fi Alliance certifies vendors in compliance with the full 802.11i specification under the WPA2 program.

IEEE 802.11i Services

The 802.11i RSN security specification defines the following services:

- **Authentication:** A protocol is used to define an exchange between a user and an AS (authentication server) that provides mutual authentication and generates temporary keys to be used between the client and the AP over the wireless link.
- **Access control[1]:** This function enforces the use of the authentication function, routes the messages properly, and facilitates key exchange. It can work with a variety of authentication protocols.
- **Privacy with message integrity:** MAC-level data (e.g., an LLC PDU) are encrypted along with a message integrity code that ensures that the data have not been altered.

Figure 24.6a indicates the security protocols used to support these services, while Figure 24.6b lists the cryptographic algorithms used for these services.

IEEE 802.11i Phases of Operation

The operation of an IEEE 802.11i RSN can be broken down into five distinct phases. The exact nature of the phases will depend on the configuration and the end points of the communication. Possibilities include (see Figure 24.5):

1. Two wireless stations in the same BSS communicating via the access point for that BSS.

[1]In this context, we are discussing access control as a security function. This is a different function than medium access control, as described in Section 24.2. Unfortunately, the literature and the standards use the term *access control* in both contexts.

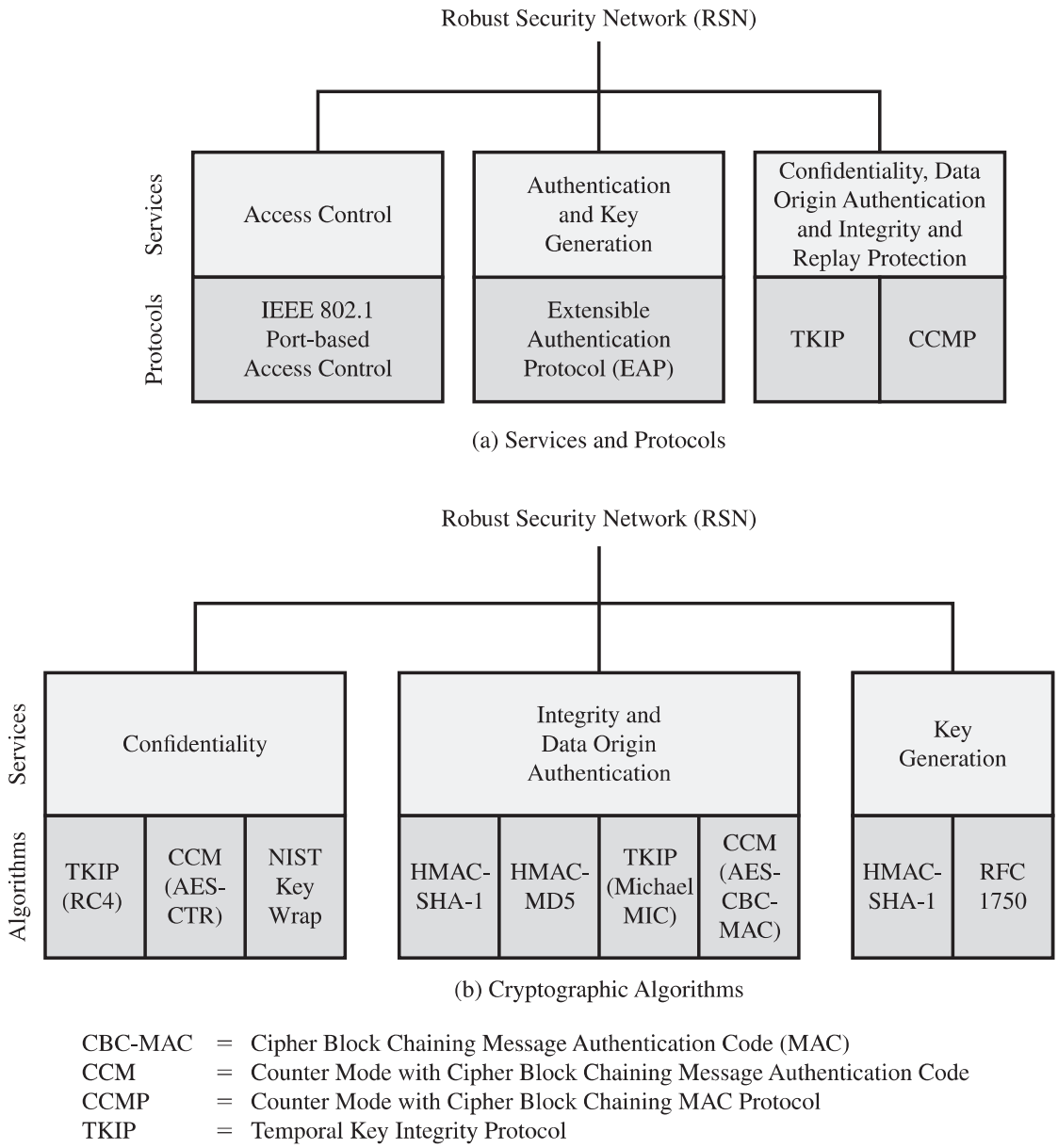


Figure 24.6 Elements of IEEE 802.11i

2. Two wireless stations (STAs) in the same ad hoc IBSS communicating directly with each other.
3. Two wireless stations in different BSSs communicating via their respective APs across a distribution system.
4. A wireless station communicating with an end station on a wired network via its AP and the distribution system.

IEEE 802.11i security is concerned only with secure communication between the STA and its AP. In case 1 in the preceding list, secure communication is assured if each STA establishes secure communications with the AP. Case 2 is similar, with the AP functionality residing in the STA. For case 3, security is not provided across the

distribution system at the level of IEEE 802.11, but only within each BSS. End-to-end security (if required) must be provided at a higher layer. Similarly, in case 4, security is only provided between the STA and its AP.

With these considerations in mind, Figure 24.7 depicts the five phases of operation for an RSN and maps them to the network components involved. One new component is the authentication server (AS). The rectangles indicate the exchange of sequences of MPDUs. The five phases are defined as follows:

- **Discovery:** An AP uses messages called Beacons and Probe Responses to advertise its IEEE 802.11i security policy. The STA uses these to identify an AP for a WLAN with which it wishes to communicate. The STA associates with the AP, which it uses to select the cipher suite and authentication mechanism when the Beacons and Probe Responses present a choice.
- **Authentication:** During this phase, the STA and AS prove their identities to each other. The AP blocks nonauthentication traffic between the STA and AS until the authentication transaction is successful. The AP does not participate in the authentication transaction other than forwarding traffic between the STA and AS.
- **Key Management:** The AP and the STA perform several operations that cause cryptographic keys to be generated and placed on the AP and the STA. Frames are exchanged only between the AP and STA.

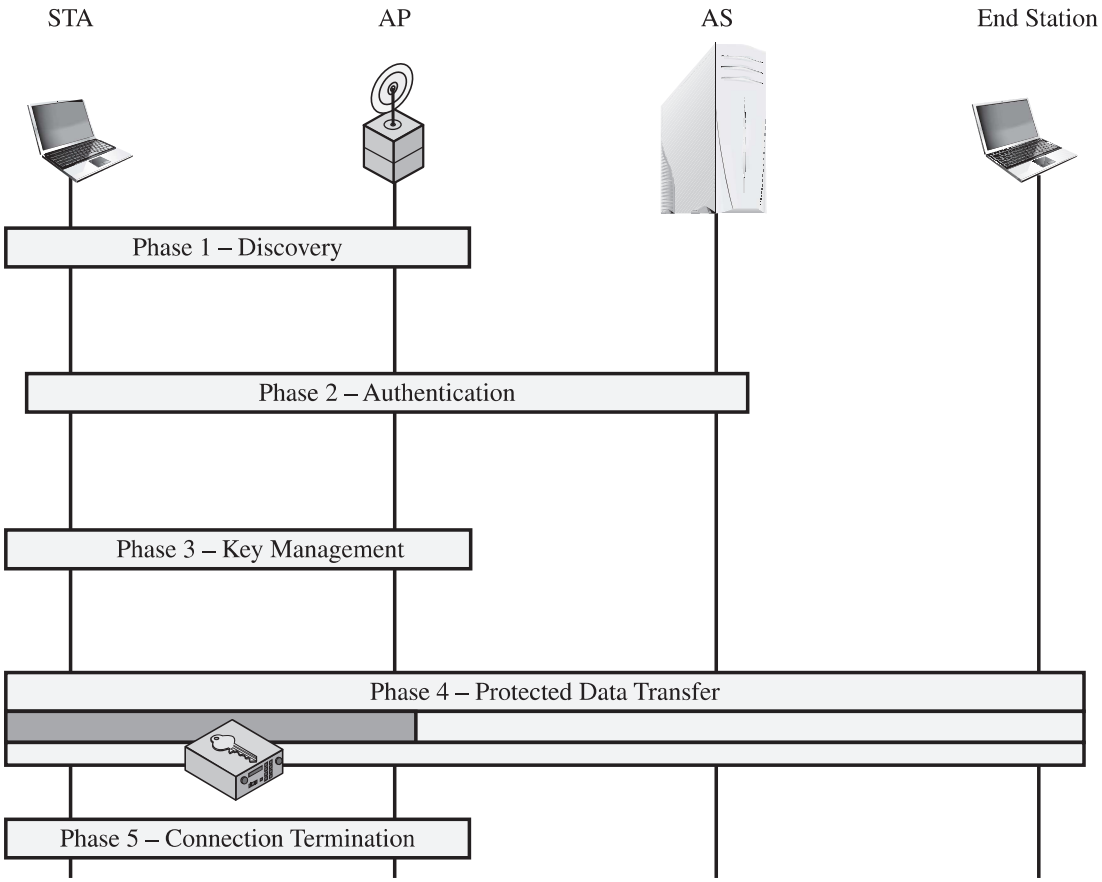


Figure 24.7 IEEE 802.11i Phases of Operation

- **Protected data transfer:** Frames are exchanged between the STA and the end station through the AP. As denoted by the shading and the encryption module icon, secure data transfer occurs between the STA and the AP only; security is not provided end-to-end.
- **Connection termination:** The AP and STA exchange frames. During this phase, the secure connection is torn down and the connection is restored to the original state.

Discovery Phase

We now look in more detail at the RSN phases of operation, beginning with the discovery phase, which is illustrated in the upper portion of Figure 24.8. The purpose

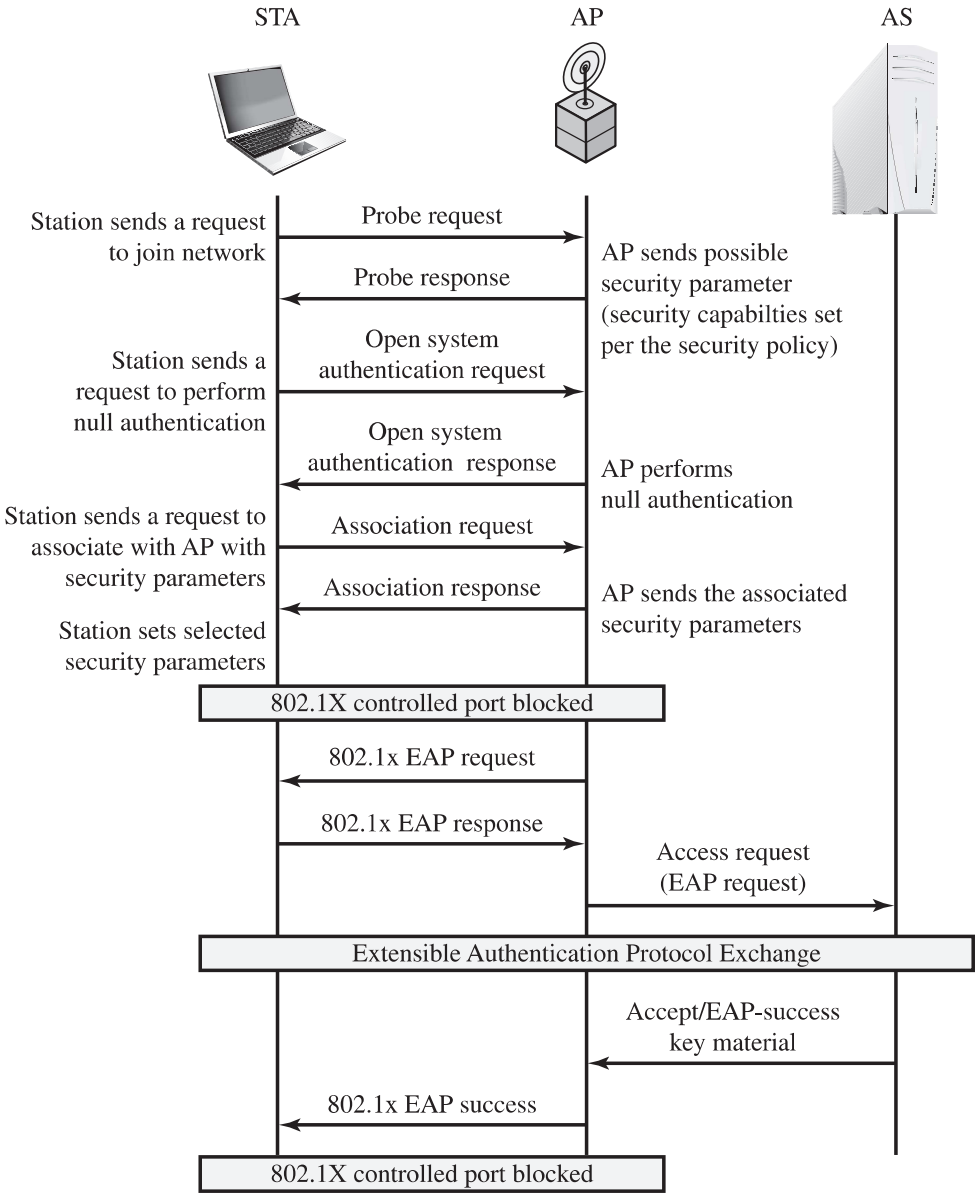


Figure 24.8 IEEE 802.11i Phases of Operation: Capability Discovery, Authentication, and Association

of this phase is for an STA and an AP to recognize each other, agree on a set of security capabilities, and establish an association for future communication using those security capabilities.

Security Capabilities During this phase, the STA and AP decide on specific techniques in the following areas:

- Confidentiality and MPDU integrity protocols for protecting unicast traffic (traffic only between this STA and AP)
- Authentication method
- Cryptography key management approach

Confidentiality and integrity protocols for protecting multicast/broadcast traffic are dictated by the AP, since all STAs in a multicast group must use the same protocols and ciphers. The specification of a protocol, along with the chosen key length (if variable), is known as a *cipher suite*. The options for the confidentiality and integrity cipher suite are:

- WEP, with either a 40-bit or 104-bit key, which allows backward compatibility with older IEEE 802.11 implementations
- TKIP
- CCMP
- Vendor-specific methods

The other negotiable suite is the authentication and key management (AKM) suite, which defines (1) the means by which the AP and STA perform mutual authentication and (2) the means for deriving a root key from which other keys may be generated. The possible AKM suites are:

- IEEE 802.1X
- Pre-shared key (no explicit authentication takes place and mutual authentication is implied if the STA and AP share a unique secret key)
- Vendor-specific methods

MPDU Exchange The discovery phase consists of three exchanges:

- **Network and security capability discovery:** During this exchange, STAs discover the existence of a network with which to communicate. The AP either periodically broadcasts its security capabilities (not shown in figure), indicated by RSN IE (Robust Security Network Information Element), in a specific channel through the Beacon frame; or responds to a station's Probe Request through a Probe Response frame. A wireless station may discover available access points and corresponding security capabilities by either passively monitoring the Beacon frames or actively probing every channel.
- **Open system authentication:** The purpose of this frame sequence, which provides no security, is simply to maintain backward compatibility with the IEEE 802.11 state machine, as implemented in existing IEEE 802.11 hardware. In essence, the two devices (STA and AP) simply exchange identifiers.

- **Association:** The purpose of this stage is to agree on a set of security capabilities to be used. The STA then sends an Association Request frame to the AP. In this frame, the STA specifies one set of matching capabilities (one authentication and key management suite, one pairwise cipher suite, and one group-key cipher suite) from among those advertised by the AP. If there is no match in capabilities between the AP and the STA, the AP refuses the Association Request. The STA blocks it too, in case it has associated with a rogue AP or someone is inserting frames illicitly on its channel. As shown in Figure 24.8, the IEEE 802.1X controlled ports are blocked, and no user traffic goes beyond the AP. The concept of blocked ports is explained subsequently.

Authentication Phase

As was mentioned, the authentication phase enables mutual authentication between an STA and an authentication server located in the DS. Authentication is designed to allow only authorized stations to use the network and to provide the STA with assurance that it is communicating with a legitimate network.

IEEE 802.1X Access Control Approach IEEE 802.11i makes use of another standard that was designed to provide access control functions for LANs. The standard is IEEE 802.1X, Port-Based Network Access Control. The authentication protocol that is used, the Extensible Authentication Protocol (EAP), is defined in the IEEE 802.1X standard. IEEE 802.1X uses the terms *supplicant*, *authenticator*, and *authentication server*. In the context of an 802.11 WLAN, the first two terms correspond to the wireless station and the AP. The AS is typically a separate device on the wired side of the network (i.e., accessible over the DS) but could also reside directly on the authenticator.

Until the AS authenticates a supplicant (using an authentication protocol), the authenticator only passes control and authentication messages between the supplicant and the AS; the 802.1X control channel is unblocked, but the 802.11 data channel is blocked. Once a supplicant is authenticated and keys are provided, the authenticator can forward data from the supplicant, subject to predefined access control limitations for the supplicant to the network. Under these circumstances, the data channel is unblocked.

As indicated in Figure 24.9, 802.1X uses the concepts of controlled and uncontrolled ports. Ports are logical entities defined within the authenticator and refer to physical network connections. For a WLAN, the authenticator (the AP) may have only two physical ports: one connecting to the DS, and one for wireless communication within its BSS. Each logical port is mapped to one of these two physical ports. An uncontrolled port allows the exchange of PDUs between the supplicant and the other AS, regardless of the authentication state of the supplicant. A controlled port allows the exchange of PDUs between a supplicant and other systems on the LAN only if the current state of the supplicant authorizes such an exchange.

The 802.1X framework, with an upper-layer authentication protocol, fits nicely with a BSS architecture that includes a number of wireless stations and an AP. However, for an IBSS, there is no AP. For an IBSS, 802.11i provides a more complex solution that, in essence, involves pairwise authentication between stations on the IBSS.

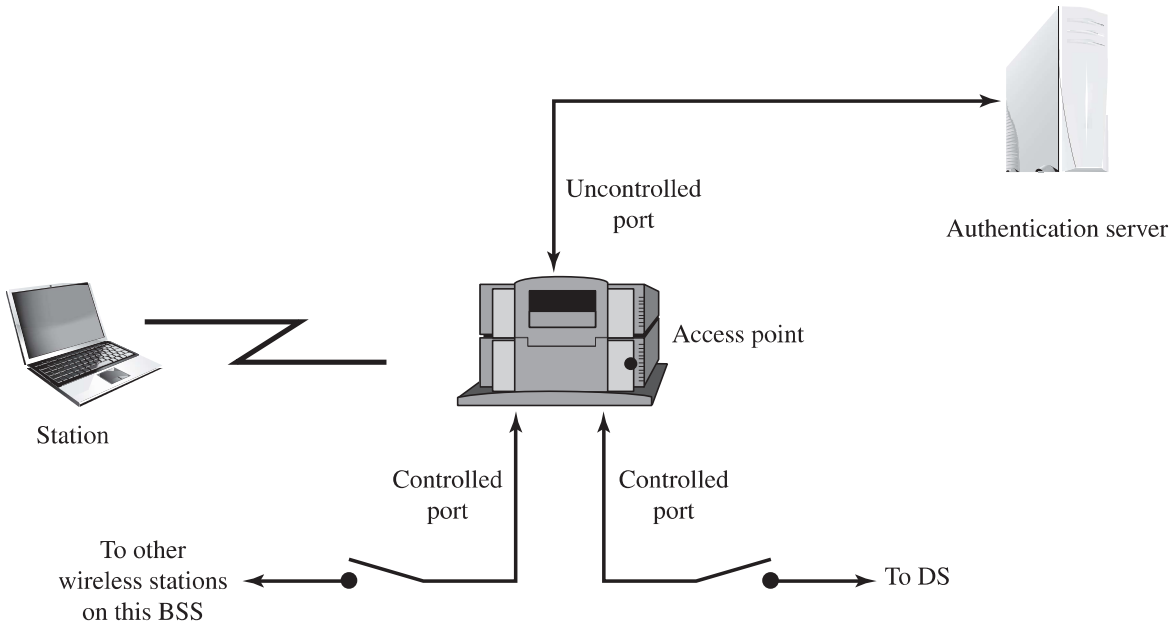


Figure 24.9 802.1X Access Control

MPDU EXCHANGE The lower part of Figure 24.8 shows the MPDU exchange dictated by IEEE 802.11 for the authentication phase. We can think of authentication phase as consisting of the following three phases.

- **Connect to AS:** The STA sends a request to its AP (the one with which it has an association) for connection to the AS. The AP acknowledges this request and sends an access request to the AS.
- **EAP exchange:** This exchange authenticates the STA and AS to each other. A number of alternative exchanges are possible, as explained subsequently.
- **Secure key delivery:** Once authentication is established, the AS generates a master session key (MSK), also known as the Authentication, Authorization, and Accounting (AAA) key, and sends it to the STA. As explained subsequently, all the cryptographic keys needed by the STA for secure communication with its AP are generated from this MSK. IEEE 802.11i does not prescribe a method for secure delivery of the MSK but relies on EAP for this. Whatever method is used, it involves the transmission of an MPDU containing an encrypted MSK from the AS, via the AP, to the AS.

EAP EXCHANGE As mentioned, there are a number of possible EAP exchanges that can be used during the authentication phase. Typically, the message flow between STA and AP employs the EAP over LAN (EAPOL) protocol, and the message flow between the AP and AS uses the Remote Authentication Dial In User Service (RADIUS) protocol, although other options are available for both STA-to-AP and AP-to-AS exchanges. NIST SP 800-97 (*Establishing Wireless Robust Security Networks: A Guide to IEEE 802.11i*, February 2007) provides the following summary of the authentication exchange using EAPOL and RADIUS.

1. The EAP exchange begins with the AP issuing an EAP-Request/Identity frame to the STA.

2. The STA replies with an EAP-Response/Identity frame, which the AP receives over the uncontrolled port. The packet is then encapsulated in RADIUS over EAP and passed on to the RADIUS server as a RADIUS-Access-Request packet.
3. The AAA server replies with a RADIUS-Access-Challenge packet, which is then passed on to the STA as an EAP-Request. This request is of the appropriate authentication type and contains relevant challenge information.
4. The STA formulates an EAP-Response message and sends it to the AS. The response is translated by the AP into a Radius-Access-Request with the response to the challenge as a data field. Steps 3 and 4 may be repeated multiple times, depending on the EAP method in use. For TLS tunneling methods, it is common for authentication to require 10–20 round trips.
5. The AAA server grants access with a Radius-Access-Accept packet. The AP issues an EAP-Success frame. (Some protocols require confirmation of the EAP success inside the TLS tunnel for authenticity validation.) The controlled port is authorized, and the user may begin to access the network.

Note from Figure 24.8 that the AP controlled port is still blocked to general user traffic. Although the authentication is successful, the ports remain blocked until the temporal keys are installed in the STA and AP, which occurs during the 4-way handshake.

Key Management Phase

During the key management phase, a variety of cryptographic keys are generated and distributed to STAs. There are two types of keys: pairwise keys used for communication between an STA and an AP, and group keys used for multicast communication. Figure 24.10, based on [FRAN07], shows the two key hierarchies, and Table 24.3 defines the individual keys.

Pairwise Keys Pairwise keys are used for communication between a pair of devices, typically between an STA and an AP. These keys form a hierarchy beginning with a master key from which other keys are derived dynamically and used for a limited period of time.

At the top level of the hierarchy are two possibilities. A **pre-shared key (PSK)** is a secret key shared by the AP and a STA and installed in some fashion outside the scope of IEEE 802.11i. The other alternative is the **master session key (MSK)**, also known as the AAAK, which is generated using the IEEE 802.1X protocol during the authentication phase, as described previously. The actual method of key generation depends on the details of the authentication protocol used. In either case (PSK or MSK), there is a unique key shared by the AP with each STA with which it communicates. All the other keys derived from this master key are also unique between an AP and an STA. Thus, each STA, at any time, has one set of keys, as depicted in the hierarchy of Figure 24.10a, while the AP has one set of such keys for each of its STAs.

The **pairwise master key (PMK)** is derived from the master key. If a PSK is used, then the PSK is used as the PMK; if a MSK is used, then the PMK is derived

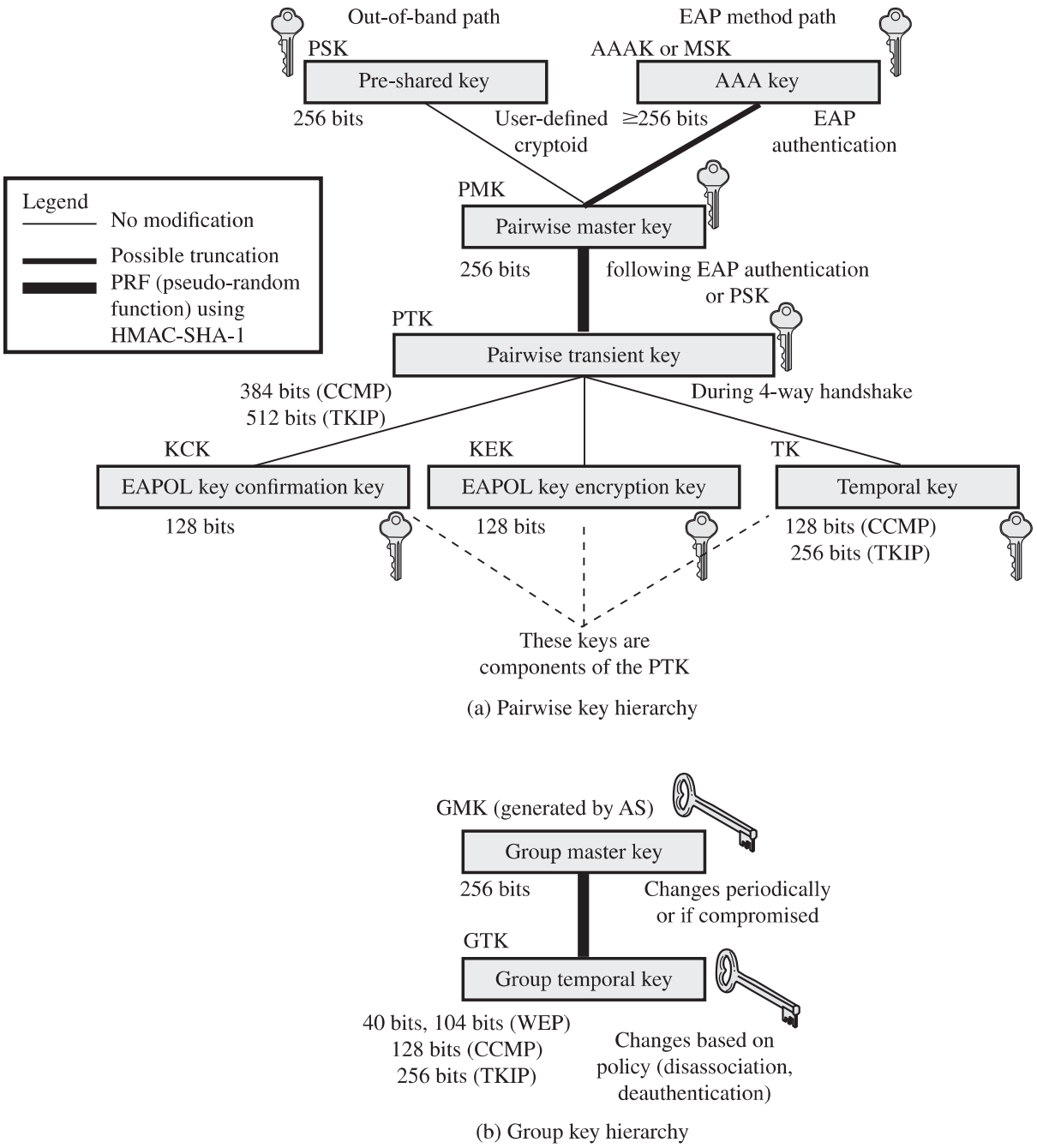


Figure 24.10 IEEE 802.11i Key Hierarchies

from the MSK by truncation (if necessary). By the end of the authentication phase, marked by the 802.1x EAP Success message (see Figure 24.8), both the AP and the STA have a copy of their shared PMK.

The PMK is used to generate the **pairwise transient key (PTK)**, which in fact consists of three keys to be used for communication between an STA and an AP after they have been mutually authenticated. To derive the PTK, the HMAC-SHA-1 function is applied to the PMK, the MAC addresses of the STA and AP, and nonces

Table 24.3 IEEE 802.11i Keys for Data Confidentiality and Integrity Protocols

Abbreviation	Name	Description/Purpose	Size (bits)	Type
AAA Key	Authentication, Accounting, and Authorization Key	Used to derive the PMK. Used with the IEEE 802.1X authentication and key management approach. Same as MMSK.	≥ 256	Key generation key, root key
PSK	Pre-Shared Key	Becomes the PMK in pre-shared key environments.	256	Key generation key, root key
PMK	Pairwise Master Key	Used with other inputs to derive the PTK.	256	Key generation key
GMK	Group Master Key	Used with other inputs to derive the GTK.	128	Key generation key
PTK	Pairwise Transient Key	Derived from the PMK. Comprises the EAPOL-KCK, EAPOL-KEK, and TK and (for TKIP) the MIC key.	512 (TKIP) 384 (CCMP)	Composite key
TK	Temporal Key	Used with TKIP or CCMP to provide confidentiality and integrity protection for unicast user traffic.	256 (TKIP) 128 (CCMP)	Traffic key
GTK	Group Temporal Key	Derived from the GMK. Used to provide confidentiality and integrity protection for multicast/broadcast user traffic.	256 (TKIP) 128 (CCMP) 40, 104 (WEP)	Traffic key
MIC Key	Message Integrity Code Key	Used by TKIP's Michael MIC to provide integrity protection of messages.	64	Message integrity key
EAPOL-KCK	EAPOL-Key Confirmation Key	Used to provide integrity protection for key material distributed during the 4-way handshake.	128	Message integrity key
EAPOL-KEK	EAPOL-Key Encryption Key	Used to ensure the confidentiality of the GTK and other key material in the 4-way handshake.	128	Traffic key/key encryption key
WEP Key	Wired Equivalent Privacy Key	Used with WEP.	40, 104	Traffic key

generated when needed. Using the STA and AP addresses in the generation of the PTK provides protection against session hijacking and impersonation; using nonces provides additional random keying material.

The three parts of the PTK are as follows:

- **EAP Over LAN (EAPOL) Key Confirmation Key (EAPOL-KCK):** Supports the integrity and data origin authenticity of STA-to-AP control frames during operational setup of an RSN. It also performs an access control function: proof-of-possession of the PMK. An entity that possesses the PMK is authorized to use the link.
- **EAPOL Key Encryption Key (EAPOL-KEK):** Protects the confidentiality of keys and other data during some RSN association procedures.
- **Temporal Key (TK):** Provides the actual protection for user traffic.

Group Keys Group keys are used for multicast communication in which one STA sends MPDUs to multiple STAs. At the top level of the group key hierarchy is the **group master key (GMK)**. The GMK is a key-generating key used with other inputs to derive the **group temporal key (GTK)**. Unlike the PTK, which is generated using material from both AP and STA, the GTK is generated by the AP and transmitted to its associated STAs. Exactly how this GTK is generated is undefined. IEEE 802.11i, however, requires that its value is computationally indistinguishable from random. The GTK is distributed securely using the pairwise keys that are already established. The GTK is changed every time a device leaves the network.

Pairwise Key Distribution The upper part of Figure 24.11 shows the MPDU exchange for distributing pairwise keys. This exchange is known as the **4-way handshake**. The STA and AP use this handshake to confirm the existence of the PMK, verify the selection of the cipher suite, and derive a fresh PTK for the following data session. The four parts of the exchange are as follows:

- **AP → STA:** Message includes the MAC address of the AP and a nonce (Anonce)
- **STA → AP:** The STA generates its own nonce (Snonce) and uses both nonces and both MAC addresses, plus the PMK, to generate a PTK. The STA then sends a message containing its MAC address and Snonce, enabling the AP to generate the same PTK. This message includes a message integrity code (MIC)[2] using HMAC-MD5 or HMAC-SHA-1-128. The key used with the MIC is KCK.
- **AP → STA:** The AP is now able to generate the PTK. The AP then sends a message to the STA, containing the same information as in the first message, but this time including a MIC.
- **STA → AP:** This is merely an acknowledgment message, again protected by a MIC.

[2]While *MAC* is commonly used in cryptography to refer to a message authentication code, the term *MIC* is used instead in connection with 802.11i because *MAC* has another standard meaning, medium access control, in networking.

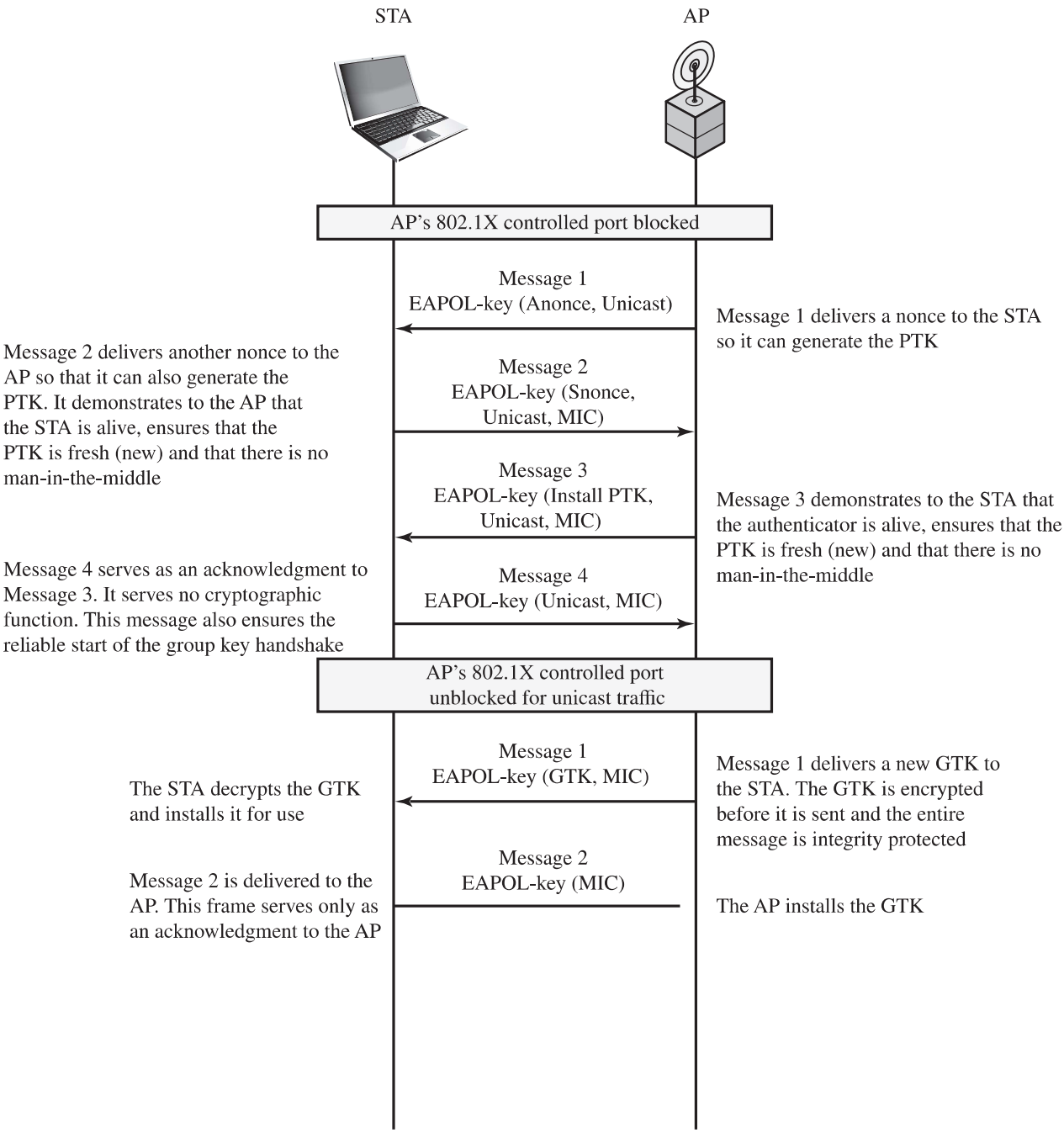


Figure 24.11 IEEE 802.11i Phases of Operation: 4-Way Handshake and Group Key Handshake

GROUP KEY DISTRIBUTION For group key distribution, the AP generates a GTK and distributes it to each STA in a multicast group. The two-message exchange with each STA consists of the following:

- **AP → STA:** This message includes the GTK, encrypted either with RC4 or with AES. The key used for encryption is KEK. A MIC value is appended.
- **STA → AP:** The STA acknowledges receipt of the GTK. This message includes a MIC value.

Protected Data Transfer Phase

IEEE 802.11i defines two schemes for protecting data transmitted in 802.11 MPDUs: the Temporal Key Integrity Protocol (TKIP) and the Counter Mode-CBC MAC Protocol (CCMP).

TKIP TKIP is designed to require only software changes to devices that are implemented with the older wireless LAN security approach called Wired Equivalent Privacy (WEP). TKIP provides two services:

- **Message integrity:** TKIP adds a message integrity code to the 802.11 MAC frame after the data field. The MIC is generated by an algorithm, called Michael, that computes a 64-bit value using as input the source and destination MAC address values and the data field, plus key material.
- **Data confidentiality:** Data confidentiality is provided by encrypting the MPDU plus MIC value using RC4.

The 256-bit TK (see Figure 24.10) is employed as follows. Two 64-bit keys are used with the Michael message digest algorithm to produce a message integrity code. One key is used to protect STA-to-AP messages, and the other key is used to protect AP-to-STA messages. The remaining 128 bits are truncated to generate the RC4 key used to encrypt the transmitted data.

For additional protection, a monotonically increasing TKIP sequence counter (TSC) is assigned to each frame. The TSC serves two purposes. First, the TSC is included with each MPDU and is protected by the MIC to protect against replay attacks. Second, the TSC is combined with the session TK to produce a dynamic encryption key that changes with each transmitted MPDU, thus making cryptanalysis more difficult.

CCMP CCMP is intended for newer IEEE 802.11 devices that are equipped with the hardware to support this scheme. As with TKIP, CCMP provides two services:

- **Message integrity:** CCMP uses the cipher block chaining message authentication code (CBC-MAC), described in Chapter 12.
- **Data confidentiality:** CCMP uses the CTR block cipher mode of operation with AES for encryption. CTR is described in Chapter 20.

The same 128-bit AES key is used for both integrity and confidentiality. The scheme uses a 48-bit packet number to construct a nonce to prevent replay attacks.

The IEEE 802.11i Pseudorandom Function

At a number of places in the IEEE 802.11i scheme, a pseudorandom function (PRF) is used. For example, it is used to generate nonces, to expand pairwise keys, and to generate the GTK. Best security practice dictates that different pseudorandom number streams be used for these different purposes. However, for implementation efficiency, we would like to rely on a single pseudorandom number generator function.

The PRF is built on the use of HMAC-SHA-1 to generate a pseudorandom bit stream. Recall that HMAC-SHA-1 takes a message (block of data) and a key of length at least 160 bits and produces a 160-bit hash value. SHA-1 has the property that the change of a single bit of the input produces a new hash value with no apparent connection to the preceding hash value. This property is the basis for pseudorandom number generation.

The IEEE 802.11i PRF takes four parameters as input and produces the desired number of random bits. The function is of the form PRF(K, A, B, Len), where

K = a secret key

A = a text string specific to the application (e.g., nonce generation or pairwise key expansion)

B = some data specific to each case

Len = desired number of pseudorandom bits

For example, for the pairwise transient key for CCMP:

PTK = PRF(PMK, "Pairwise key expansion," min(AP-Addr, STA-Addr) ‖ max (AP-Addr, STA-Addr) ‖ min(Anonce, Snonce) ‖ max(Anonce, Snonce), 384)

So, in this case, the parameters are

K = PMK

A = the text string "Pairwise key expansion"

B = a sequence of bytes formed by concatenating the two MAC addresses and the two nonces

Len = 384 bits

Similarly, a nonce is generated by

Nonce = PRF(Random Number, "Init Counter," MAC ‖ Time, 256)

Where Time is a measure of the network time known to the nonce generator. The group temporal key is generated by:

GTK = PRF(GMK, "Group key expansion," MAC ‖ Gnonce, 256)

Figure 24.12 illustrates the function PRF(K, A, B, Len). The parameter K serves as the key input to HMAC. The message input consists of four items concatenated together: the parameter A, a byte with value 0, the parameter B, and a counter i. The counter is initialized to 0. The HMAC algorithm is run once, producing a 160-bit hash value. If more bits are required, HMAC is run again with the same inputs, except that i is incremented each time until the necessary number of bits is generated. We can express the logic as

```
PRF(K, A, B, Len)
R ← null string
for i ← 0 to ((Len + 159)/160 - 1) do
R ← R‖HMAC-SHA-1(K, A‖0‖B‖i)
Return Truncate-to-Len(R, Len)
```

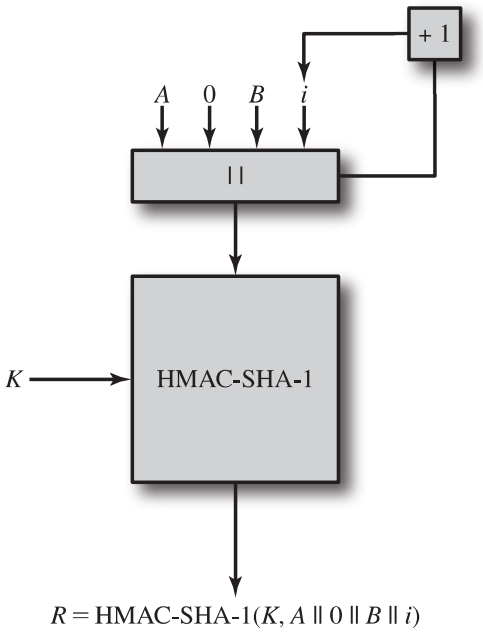


Figure 24.12 **IEEE 802.11i Pseudorandom Function**

24.5 KEY TERMS, REVIEW QUESTIONS, AND PROBLEMS

Key Terms

4-way handshake	independent BSS (IBSS)	physical layer
access point (AP)	logical link control (LLC)	pseudorandom function
basic service set (BSS)	medium access control (MAC)	Robust Security Network (RSN)
Counter Mode-CBC MAC Protocol (CCMP)	MAC header	Temporal Key Integrity Protocol (TKIP)
distribution system (DS)	MAC protocol data unit (MPDU)	Wi-Fi
extended service set (ESS)	MAC service data unit (MSDU)	Wi-Fi Protected Access (WPA)
group keys	MAC trailer	Wired Equivalent Privacy (WEP)
IEEE 802.1X	message integrity code (MIC)	wireless LAN (WLAN)
IEEE 802.11	Michael	
IEEE 802.11i	pairwise keys	

Review Questions

24.1 What is the basic building block of an 802.11 WLAN?
24.2 Define an extended service set.
24.3 List and briefly define IEEE 802.11 services.
24.4 Which assumptions form the basis of security policy for mobile devices?
24.5 List the seven major security concerns for mobile devices.
24.6 Briefly describe the pseudorandom stream generation of the IEEE 802.11i scheme and list some uses of the pseudorandom function.
24.7 Briefly describe the four IEEE 802.11i phases of operation.
24.8 What is the difference between TKIP and CCMP?

Problems

24.1 In IEEE 802.11, open system authentication simply consists of two communications. An authentication is requested by the client, which contains the station ID (typically the MAC address). This is followed by an authentication response from the AP/router containing a success or failure message. An example of when a failure may occur is if the client's MAC address is explicitly excluded in the AP/router configuration.

a. What are the benefits of this authentication scheme?

b. What are the security vulnerabilities of this authentication scheme?

24.2 Prior to the introduction of IEEE 802.11i, the security scheme for IEEE 802.11 was Wired Equivalent Privacy (WEP). WEP assumed all devices in the network share a secret key. The purpose of the authentication scenario is for the STA to prove that it possesses the secret key. Authentication proceeds as shown in Figure 24.13. The STA sends a message to the AP requesting authentication. The AP issues a challenge, which is a sequence of 128 random bytes, sent as plaintext. The STA encrypts the challenge with the shared key and returns it to the AP. The AP decrypts the incoming value and compares it to the challenge that it sent. If there is a match, the AP confirms that authentication has succeeded.

a. What are the benefits of this authentication scheme?

b. This authentication scheme is incomplete. What is missing and why is this important? *Hint*: The addition of one or two messages would fix the problem.

c. What is a cryptographic weakness of this scheme?

24.3 For WEP, data integrity and data confidentiality are achieved using the RC4 stream encryption algorithm. The transmitter of an MPDU performs the following steps, referred to as encapsulation:

1. The transmitter selects an initial vector (IV) value.
2. The IV value is concatenated with the WEP key shared by transmitter and receiver to form the seed, or key input, to RC4.

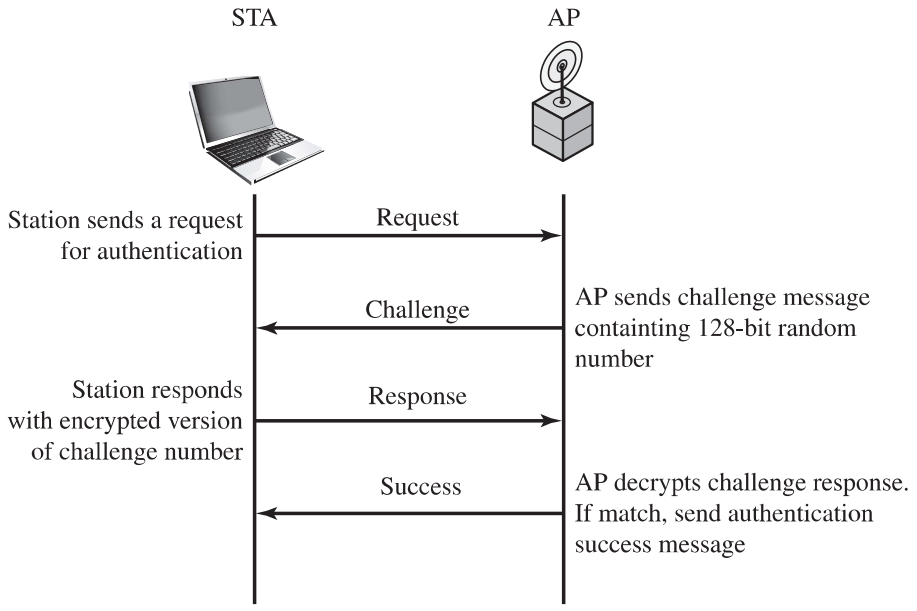


Figure 24.13 WEP Authentication

3. A 32-bit cyclic redundancy check (CRC) is computed over all the bits of the MAC data field and appended to the data field. The CRC is a common error-detection code used in data link control protocols. In this case, the CRC serves as a integrity check value (ICV).
4. The result of step 3 is encrypted using RC4 to form the ciphertext block.
5. The plaintext IV is prepended to the ciphertext block to form the encapsulated MPDU for transmission.

a. Draw a block diagram that illustrates the encapsulation process.

b. Describe the steps at the receiver end to recover the plaintext and perform the integrity check.

c. Draw a block diagram that illustrates part b.

24.4 A potential weakness of the CRC as an integrity check is that it is a linear function. This means that you can predict which bits of the CRC are changed if a single bit of the message is changed. Furthermore, it is possible to determine which combination of bits could be flipped in the message so the net result is no change in the CRC. Thus, there are a number of combinations of bit flippings of the plaintext message that leave the CRC unchanged, so message integrity is defeated. However, in WEP, if an attacker does not know the encryption key, the attacker does not have access to the plaintext, only to the ciphertext block. Does this mean that the ICV is protected from the bit flipping attack? Explain.

APPENDIX A

PROJECTS AND OTHER STUDENT EXERCISES FOR TEACHING COMPUTER SECURITY

Many instructors believe that research or implementation projects are crucial to the clear understanding of computer security. Without projects, it may be difficult for students to grasp some of the basic concepts and interactions among security functions. Projects reinforce the concepts introduced in the book, give the student a greater appreciation of how a cryptographic algorithm or security function works, and can motivate students and give them confidence that they are capable of not only understanding but implementing the details of a security capability.

In this text, we have tried to present the concepts of computer security as clearly as possible and have provided numerous homework problems to reinforce those concepts. However, many instructors will wish to supplement this material with projects. This appendix provides some guidance in that regard and describes support material available in the **Instructor's Resource Center (IRC)** for this book, accessible from Pearson for instructors. The support material covers 11 types of projects and other student exercises:

- Hacking projects
- Laboratory exercise
- Security education (SEED) projects
- Research projects
- Programming projects
- Practical security assessments
- Firewall projects
- Case studies
- Reading/report assignments
- Writing assignments
- Webcasts for teaching computer security

A.1 HACKING PROJECT

The aim of this project is to hack into a corporation's network through a series of steps. The corporation is named Extreme In Security Corporation. As the name indicates, the corporation has some security holes in it and a clever hacker is able to access critical information by hacking into its network. The IRC includes what is needed to set up the Website. The student's goal is to capture the secret

information about the price on the quote the corporation is placing next week to obtain a contract for a governmental project.

The student should start at the Website and find his or her way into the network. At each step, if the student succeeds, there are indications as to how to proceed on to the next step as well as the grade until that point.

The project can be attempted in three ways:

1. Without seeking any sort of help
2. Using some provided hints
3. Using exact directions

The IRC includes the files needed for this project:

1. Web Security project named extremeinsecure (extremeinsecure.zip)
2. Web Hacking exercises (XSS and Script-attacks) covering client-side and server-side vulnerability exploitations respectively (webhacking.zip)
3. Documentation for installation and use for the above (description.doc)
4. A PowerPoint file describing Web hacking (Web_Security.ppt). This file is crucial to understanding how to use the exercises, since it clearly explains the operation using screen shots.

This project was designed and implemented by Professor Sreekanth Malladi of Dakota State University.

A.2 LABORATORY EXERCISES

Professor Sanjay Rao and Ruben Torres of Purdue University have prepared a set of laboratory exercises that are part of the IRC. These are implementation projects designed to be programmed on Linux, but could be adapted for any UNIX environment. These laboratory exercises provide realistic experience in implementing security functions and applications.

A.3 SECURITY EDUCATION (SEED) PROJECTS

The SEED projects are a set of hands-on exercises, or labs, covering a wide range of security topics. They were designed by Professor Wenliang Du of Syracuse University for use by other instructors [DU11]. The SEED lab exercises are designed so no dedicated physical laboratory is needed. All SEED labs can be carried out on students' personal computers or in a general computing laboratory. The collection consists of three types of lab exercises:

- **Vulnerability and attack labs:** These 12 labs cover many common vulnerabilities and attacks. In each lab, students are given a system (or program) with hidden vulnerabilities. Based upon the hints provided, students must find these vulnerabilities, then devise strategies to exploit them. Students also need to

demonstrate ways to defend against the attacks or comment on the prevailing mitigating methods and their effectiveness.

- **Exploration labs:** The objective of these 9 labs is to enhance students' learning via observation, playing, and exploration, so they can understand what security principles feel like in a real system; and to provide students with opportunities to apply security principles in analyzing and evaluating systems.
- **Design and implementation labs:** In security education, students should also be given opportunities to apply security principles in designing and implementing systems. The challenge is to design meaningful assignments that do not require a major commitment of time. The 9 labs in this category meet this requirement.

Table A.1 maps the 30 lab exercises in the SEED repertoire to the relevant chapters in the book, together with an estimate of the number of weeks required for the typical student to complete a lab, assuming about 10 hours per week devoted to the task.

Table A.1 Mapping of SEED Labs to Textbook Chapters

Types	Labs	Time (weeks)	Chapters
Vulnerability and Attack Labs (Linux-based)	Buffer Overflow Vulnerability	1	10
	Return-to-libc Attack	1	10
	Format String Vulnerability	1	11
	Race Condition Vulnerability	1	11
	Set-UID Program Vulnerability	1	11
	Chroot Sandbox Vulnerability	1	12
	Cross-Site Request Forgery Attack	1	11
	Cross-Site Scripting Attack	1	11
	SQL Injection Attack	1	5
	Clickjacking Attack	1	6
	TCP/IP Attacks	2	7, 22
	DNS Pharming Attacks	2	22
Exploration Labs (Linux-based)	Pack Sniffing & Spoofing	1	22
	Pluggable Authentication Module	1	3
	Web Access Control	1	4, 6
	SYN Cookie	1	7, 22
	Linux Capability-Based Access Control	1	4, 12
	Secret-Key Encryption	1	20
	One-Way Hash Function	1	21
	Public-Key Infrastructure	1	21, 23
	Linux Firewall Exploration	1	9

(Continued)

Table A.1 *(Continued)*

Types	Labs	Time (weeks)	Chapters
Design and Implementation Labs	Virtual Private Network (Linux)	4	22
	IPsec (Minix)	4	22
	Firewall (Linux)	2	9
	Firewall (Minix)	2	9
	Role-Based Access Control (Minix)	4	4
	Capability-Based Access Control (Minix)	3	4
	Encrypted File System (Minix)	4	12
	Address Space Randomization (Minix)	2	12
	Set-Random UID Sandbox (Minix)	1	12

A Webpage accessible through the Companion Website provides links to all the labs, organized by chapter. Each lab includes student instructions, relevant documents, and any software needed to perform the lab. In addition, a link is provided for instructors to enable them to obtain the instructor manual.

A.4 RESEARCH PROJECTS

An effective way of reinforcing basic concepts from the course and for teaching students research skills is to assign a research project. Such a project could involve a literature search as well as an Internet search of vendor products, research lab activities, and standardization efforts. Projects could be assigned to teams or, for smaller projects, to individuals. In any case, it is best to require some sort of project proposal early in the term, giving the instructor time to evaluate the proposal for appropriate topic and appropriate level of effort. Student handouts for research projects should include:

- A format for the proposal
- A format for the final report
- A schedule with intermediate and final deadlines
- A list of possible project topics

The students can select one of the topics listed in the IRC or devise their own comparable project. The instructor's supplement includes a suggested format for the proposal and final report as well as a list of possible research topics.

The following individuals have supplied the research and programming projects suggested in the instructor's supplement: Henning Schulzrinne of Columbia University; Cetin Kaya Koc of Oregon State University; David M. Balenson of Trusted Information Systems and George Washington University; Dan Wallach of Rice University; and David Evans of the University of Virginia.

A.5 PROGRAMMING PROJECTS

The programming project is a useful pedagogical tool. There are several attractive features of stand-alone programming projects that are not part of an existing security facility:

1. The instructor can choose from a wide variety of cryptography and computer security concepts to assign projects.
2. The projects can be programmed by the students on any available computer and in any appropriate language; they are platform- and language-independent.
3. The instructor need not download, install, and configure any particular infrastructure for stand-alone projects.

There is also flexibility in the size of projects. Larger projects give students more a sense of achievement, but students with less ability or fewer organizational skills can be left behind. Larger projects usually elicit more overall effort from the best students. Smaller projects can have a higher concepts-to-code ratio, and because more of them can be assigned, the opportunity exists to address a variety of different areas.

Again, as with research projects, the students should first submit a proposal. The student handout should include the same elements listed in the preceding section. The IRC includes a set of 12 possible programming projects.

The following individuals have supplied the research and programming projects suggested in the IRC: Henning Schulzrinne of Columbia University; Cetin Kaya Koc of Oregon State University; and David M. Balenson of Trusted Information Systems and George Washington University.

A.6 PRACTICAL SECURITY ASSESSMENTS

Examining the current infrastructure and practices of an existing organization is one of the best ways of developing skills in assessing its security posture. The IRC contains a description of the tasks needed to conduct a security assessment. Students, working either individually or in small groups, select a suitable small- to medium-sized organization. They then interview some key personnel in that organization to conduct a suitable selection of security risk assessment and review tasks as it relates to the organization's IT infrastructure and practices. As a result, they can then recommend suitable changes, which can improve the organization's IT security. These activities help students develop an appreciation of current security practices, and the skills needed to review these and recommend changes.

A.7 FIREWALL PROJECTS

The implementation of network firewalls can be a difficult concept for students to grasp initially. The IRC includes Network Firewall Visualization tool to convey and teach network security and firewall configuration. This tool is intended to teach

and reinforce key concepts including the use and purpose of a perimeter firewall, the use of separated subnets, the purposes behind packet filtering, and the shortcomings of a simple packet filter firewall.

The IRC includes a .jar file that is fully portable, and a series of exercises. The tool and exercises were developed at U.S. Air Force Academy.

A.8 CASE STUDIES

Teaching with case studies engages students in active learning. The IRC includes case studies in the following areas:

- Disaster recovery
- Firewalls
- Incidence response
- Physical security
- Risk
- Security policy
- Virtualization

Each case study includes learning objectives, case description, and a series of case discussion questions. Each case study is based on real-world situations and includes papers or reports describing the case.

The case studies were developed at North Carolina A&T State University.

A.9 READING/REPORT ASSIGNMENTS

Another excellent way to reinforce concepts from the course and to give students research experience is to assign papers from the literature to be read and analyzed. The IRC includes a suggested list of papers to be assigned, organized by chapter. The Premium Content Website provides a copy of each of the papers. The IRC also includes a suggested assignment wording.

A.10 WRITING ASSIGNMENTS

Writing assignments can have a powerful multiplier effect in the learning process in a technical discipline such as computer security. Adherents of the Writing Across the Curriculum (WAC) movement (http://wac.colostate.edu/) report substantial benefits of writing assignments in facilitating learning. Writing assignments lead to more detailed and complete thinking about a particular topic. In addition, writing assignments help to overcome the tendency of students to pursue a subject with a minimum of personal engagement, just learning facts and problem-solving techniques without obtaining a deep understanding of the subject matter.

The IRC contains a number of suggested writing assignments, organized by chapter. Instructors may ultimately find that this is the most important part of their

approach to teaching the material. We would greatly appreciate any feedback on this area and any suggestions for additional writing assignments.

A.11 WEBCASTS FOR TEACHING COMPUTER SECURITY

The Companion Website provides a link to a catalog of webcast sites that can be used to enhance the course. An effective way of using this catalog is to select, or allow the student to select, one or a few videos to watch, then assign the student to write a report/analysis of the video.

ACRONYMS

3DES	Triple Data Encryption Standard
ABAC	Attribute-Based Access Control
AES	Advanced Encryption Standard
AH	Authentication Header
ANSI	American National Standards Institute
ATM	Automatic Teller Machine
CBC	Cipher Block Chaining
CC	Common Criteria
CFB	Cipher Feedback
CMAC	Cipher-Based Message Authentication Code
DAC	Discretionary Access Control
DBMS	Database Management System
DDoS	Distributed Denial of Service
DES	Data Encryption Standard
DMZ	Demilitarized Zone
DoS	Denial of Service
DSA	Digital Signature Algorithm
DSS	Digital Signature Standard
ECB	Electronic Codebook
ESP	Encapsulating Security Payload
FIPS	Federal Information Processing Standard
IAB	Internet Architecture Board
ICMP	Internet Control Message Protocol
IDS	Intrusion Detection System
IETF	Internet Engineering Task Force
IP	Internet Protocol
IPsec	IP Security
ISO	International Organization for Standardization
ITU	International Telecommunication Union
ITU-T	ITU Telecommunication Standardization Sector
IV	Initialization Vector
KDC	Key Distribution Center
MAC	Mandatory Access Control
MAC	Message Authentication Code
MIC	Message Integrity Code
MIME	Multipurpose Internet Mail Extension
MLS	Multilevel Security
MTU	Maximum Transmission Unit
NIDA	Network-Based IDS
NIST	National Institute of Standards and Technology
NSA	National Security Agency
OFB	Output Feedback
PIN	Personal Identification Number
PIV	Personal Identity Verification
PKI	Public Key Infrastructure
PRNG	Pseudorandom Number Generator
RDBMS	Relational Database Management System
RBAC	Role-Based Access Control
RFC	Request for Comments
RNG	Random Number Generator
RSA	Rivest-Shamir-Adleman
SHA	Secure Hash Algorithm
SHS	Secure Hash Standard
S/MIME	Secure MIME
SQL	Structured Query Language
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TPM	Trusted Platform Module
UDP	User Datagram Protocol
VPN	Virtual Private Network

LIST OF NIST AND ISO DOCUMENTS

ABBREVIATIONS

FIPS	Federal Information Processing Standard
NIST	National Institute of Standards and Technology
NISTIR	NIST Internal/Interagency Report
SP	Special Publication

NIST DOCUMENTS

FIPS 46	Data Encryption Standard, January 1977.
FIPS 113	Computer Data Authentication, May 1985.
FIPS 140-3	Security Requirements for Cryptographic Modules, September 2009.
FIPS 180-4	Secure Hash Standard (SHS), August 2015.
FIPS 181	Automated Password Generator (APG), October 1993 (withdrawn October 2015)
FIPS 186-4	Digital Signature Standard (DSS), July 2013
FIPS 197	Advanced Encryption Standard, November 2001.
FIPS 199	Standards for Security Categorization of Federal Information and Information Systems, February 2004.
FIPS 200	Minimum Security Requirements for Federal Information and Information Systems, March 2006
FIPS 201-2	Personal Identity Verification (PIV) of Federal Employees and Contractors, August 2013
FIPS 202	SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions, August 2015
NISTIR 7298	Glossary of Key Information Security Terms, May 2013.
SP 500-292	NIST Cloud Computing Reference Architecture, September 2011.
SP 800-12	An Introduction to Computer Security: The NIST Handbook, October 1995
SP 800-16	A Role-Based Model for Federal Information Technology/ Cybersecurity Training, March 2014
SP 800-18	Guide for Developing Security Plans for Federal Information Systems, February 2006.
SP 800-28	Guidelines on Active Content and Mobile Code, March 2008.
SP 800-30	Guide for Conducting Risk Assessments, September 2012.
SP 800-38A	Recommendation for Block Cipher Modes of Operation: Methods and Techniques, December 2001
SP 800-39	Managing Information Security Risk: Organization, Mission, and Information System View, March 2011
SP 800-41	Guidelines on Firewalls and Firewall Policy, September 2009.
SP 800-53	Security and Privacy Controls for Federal Information Systems and Organizations, January 2015.
SP 800-61	Computer Security Incident Handling Guide, August 2012.

SP 800-63-3 Digital Authentication Guideline, August 2016.
SP 800-82 Guide to Industrial Control Systems (ICS) Security, May 2015.
SP 800-83 Guide to Malware Incident Prevention and Handling for Desktops and Laptops, July 2013.
SP 800-92 Guide to Computer Security Log Management, September 2006
SP 800-94 Guide to Intrusion Detection and Prevention Systems, July 2012.
SP 800-97 Establishing Wireless Robust Security Networks: A Guide to IEEE 802.11i, February 2007
SP 800-100 Information Security Handbook: A Guide for Managers, October 2006
SP 800-116 A Recommendation for the Use of PIV Credentials in Physical Access Control Systems (PACS), December 2015
SP 800-124 Guidelines for Managing the Security of Mobile Devices in the Enterprise, June 2013
SP 800-137 Information Security Continuous Monitoring (ISCM) for Federal Information Systems and Organizations, September 2011
SP 800-144 Guidelines on Security and Privacy in Public Cloud Computing, December 2011.
SP 800-145 The NIST Definition of Cloud Computing, September 2011.
SP 800-146 Cloud Computing Synopsis and Recommendations, May 2012.
SP 800-162 Guide to Attribute Based Access Control (ABAC) Definition and Considerations, January 2014.
SP 800-171 Protecting Controlled Unclassified Information in Nonfederal Information Systems and Organizations, December 2016.

ISO DOCUMENTS

12207 Information technology - Software lifecycle processes, 1997
13335 Management of information and communications technology security, 2004
27000 ISMS—Overview and Vocabulary, February 2016
27001 ISMS—Requirements, October 2013
27002 Code of Practice for Information Security Controls, October 2013
27003 Information security management system implementation guidance, 2010
27004 Information security management - Measurement, 2009
27005 Information Security Risk Management, June 2011
27006 Requirements for bodies providing audit and certification of information security management systems, 2015
31000 Risk management - Principles and guidelines, 2009

See Appendix C for further information on the NIST and ISO standards setting organizations.

REFERENCES

ABBREVIATIONS

ACM	Association for Computing Machinery
IEEE	Institute of Electrical and Electronics Engineers
RFC	Request for Comments

ACM04 The Association for Computing Machinery. *USACM Policy Brief: Digital Millennium Copyright Act (DMCA).* February 6, 2004. http://www.acm.org/usacm/Issues/DMCA.htm

ACUN13 Acunetix, Inc. *Web Application Security—Check your Site for Web Application Vulnerabilities.* 2013. http://www.acunetix.com/websitesecurity/webapp-security/2013

AGOS06 Agosta, J., et al. "Towards Autonomic Enterprise Security: Self-Defending Platforms, Distributed Detection, and Adaptive Feedback." *Intel Technology Journal*, November 9, 2006.

ANDE80 Anderson, J. *Computer Security Threat Monitoring and Surveillance.* Fort Washington, PA: James P. Anderson Co., April 1980.

ANLE07 Anley, C., et al. *The Shellcoder's Handbook: Discovering and Exploiting Security Holes*. Hoboken, NJ: John Wiley & Sons, 2007.

ANTE06 Ante, S., and Grow, B. "Meet the Hackers." *Business Week*, May 29, 2006.

ANTH07 Anthes, G. "Computer Security: Adapt or Die." *ComputerWorld*, January 8, 2007.

ARBO10 Arbor Networks. *Worldwide Infrastructure Security Report.* January 2010.

ARMY10 Department of the Army. *Physical Security.* Field Manual FM 3-99.32, August 2010.

AROR11 Arora, K.; Kumar, K.; and Sachdeva, M. "Impact Analysis of Recent DDoS Attacks." *International Journal on Computer Science and Engineering*, Vol. 3, No. 2, February 2011.

AROR12 Arora, M. "How Secure Is AES against Brute-Force Attack?" *EE Times*, May 7, 2012.

AXEL00 Axelsson, S. "The Base-Rate Fallacy and the Difficulty of Intrusion Detection." *ACM Transactions and Information and System Security*, August 2000.

AYCO06 Aycock, J. *Computer Viruses and Malware.* New York: Springer, 2006.

BAIL05 Bailey, M., et al. "The Internet Motion Sensor: A Distributed Blackhole," *Proceedings of the Network and Distributed System Security Symposium Conference*, February 2005.

BALA98 Balasubramaniyan, J., et al. "An Architecture for Intrusion Detection Using Autonomous Agents." *Proceedings, 14th Annual Computer Security Applications Conference*, 1998.

BARD12 Bardou, R., et al, "Efficient Padding Oracle Attacks on Cryptographic Hardware." INRIA, Rapport de recherche RR-7944, April 2012. http://hal.inria.fr/hal-00691958

BASU12 Basu, A. *Intel AES-NI Performance Testing over Full Disk Encryption.* Intel Corp., May 2012.

BELL94 Bellovin, S., and Cheswick, W. "Network Firewalls." *IEEE Communications Magazine*, September 1994.

BELL96 Bellare, M.; Canetti, R.; and Krawczyk, H. "Keying Hash Functions for Message Authentication." *Proceedings, CRYPTO '96*, August 1996; published by Springer-Verlag. An expanded version is available at http://www-cse.ucsd.edu/users/mihir

BELL16 Bellovin, S. "Attack Surfaces." *IEEE Security & Privacy*, May–June 2016.

BENN06 Ben-Natan, R. *Data Security, Governance & Privacy: Protecting the Core of Your Business.* Guardium White Paper, 2006. www.guardium.com

BEUC13 Beuchelt, G. "Securing Web Applications, Services, and Servers." In [VACC13].

BIDG06 Bidgoli, H., ed. *Handbook of Information Security.* Hoboken, NJ: Wiley, 2006.

BINS10 Binsalleeh, H., et al. "On the Analysis of the Zeus Botnet Crimeware Toolkit." *Proceedings of the 8th Annual International Conference on Privacy, Security and Trust*, IEEE, September 2010.

BLEI98 Bleichenbacher, D. "Chosen Ciphertext Attacks against Protocols Based on the RSA Encryption Standard PKCS #1." *CRYPTO '98*, 1998.

BLOO70 Bloom, B. "Space/time Trade-offs in Hash Coding with Allowable Errors." *Communications of the ACM*, July 1970.

BONN12 Bonneau, J. "The Science of Guessing: Analyzing an Anonymized Vorpus of 70 Million Passwords." *IEEE Symposium on Security and Privacy*, 2012.

BOSW14 Bosworth, S.; Kabay, M.; and Whyne, E., eds. *Computer Security Handbook.* New York: Wiley, 2014.

BRAU01 Braunfeld, R., and Wells, T. "Protecting Your Most Valuable Asset: Intellectual Property." *IT Pro*, March/April 2001.

CARL06 Carl, G., et al. "Denial-of-Service Attack-Detection Techniques." *IEEE Internet Computing*, January-February 2006.

CARN03 Carnegie-Mellon Software Engineering Institute. *Handbook for Computer Security Incident Response Teams (CSIRTs).* CMU/SEI-2003-HB-002, April 2003.

CASS01 Cass, S. "Anatomy of Malice." *IEEE Spectrum*, November 2001.

CCPS12a Common Criteria Project Sponsoring Organisations. *Common Criteria for Information Technology Security Evaluation, Part 1: Introduction and General Model.* CCIMB-2012-09-001, September 2012.

CCPS12b Common Criteria Project Sponsoring Organisations. *Common Criteria for Information Technology Security Evaluation, Part 2: Security Functional Components.* CCIMB-2012-09-002, September 2012.

CHOI08 Choi, M., et al. "Wireless Network Security: Vulnerabilities, Threats and Countermeasures." *International Journal of Multimedia and Ubiquitous Engineering*, July 2008.

CHAN02 Chang, R. "Defending against Flooding-Based Distributed Denial-of-Service Attacks: A Tutorial." *IEEE Communications Magazine*, October 2002.

CHAN09 Chandola, V.; Banerjee, A.; and Kumar, V. "Anomaly Detection: A Survey." *ACM Computing Surveys*, July 2009.

CHAN11 Chandrashekhar, R., et al. "SQL Injection Attack Mechanisms and Prevention Techniques." *Proceedings of the 2011 international Conference on Advanced Computing, Networking and Security*, 2011

CHEN11 Chen, T., and Abu-Nimeh, S. "Lessons from Stuxnet" *IEEE Computer*, Vol. 44 No. 4, pp. 91–93, April 2011.

CLAR15 Clark, K.; Duckham, M.; Guillemin, M.; Hunter, A.; McVernon, J.; O'Keefe, C.; Pitkin, C.; Prawer, S.; Sinnott, R.; Warr, D.; and Waycott, J. *Guidelines for the Ethical use of Digital Data in Human Research*, The University of Melbourne, Melbourne, 2015.

CLEE09 van Cleeff, A.; Pieters, W.; and Wieringa, R. "Security Implications of Virtualization: A Literature Study." *International Conference on Computational Science and Engineering*, IEEE, 2009.

COHE94 Cohen, F. *A Short Course on Computer Viruses.* New York: Wiley, 1994.

COLL06 Collett, S. "Encrypting Data at Rest." *Computerworld*, March 27, 2006.

CONR02 Conry-Murray, A. "Behavior-Blocking Stops Unknown Malicious Code." *Network Magazine*, June 2002.

CREE13 Creech, G. *Developing a high-accuracy cross platform Host-Based Intrusion Detection System capable of reliably detecting zero-day attacks.* PhD Thesis, The University of New South Wales, 2013.

CSA11 Cloud Security Alliance. *Security as a Service (SecaaS).* CSA Report, 2011.

CSA13 Cloud Security Alliance. *The Notorious Nine Cloud Computing Top Threats in 2013.* CSA Report, February 2013.

DAMI03 Damiani, E., et al. "Balancing Confidentiality and Efficiency in Untrusted Relational Databases." *ACM Conference on Computer and Communications Security*, 2003.

DAMI05 Damiani, E., et al. "Key Management for Multi-User Encrypted Databases." *Proceedings, 2005 ACM Workshop on Storage Security and Survivability*, 2005.

DAMO12 Damon, E., et al. "Hands-on denial of service lab exercises using SlowLoris and RUDY" In *Proceedings of the 2012 Information Security Curriculum Development Conference*, ACM, 2012.

DAMR03 Damron, J. "Identifiable Fingerprints in Network Applications."; *login*, December 2003.

DAUG04 Daugman, J. "Iris Recognition Border-Crossing System in the UEA." *International Airport Review*, Issue 2, 2004.

DAVI89 Davies, D., and Price, W. *Security for Computer Networks.* New York: Wiley, 1989.

DAWS96 Dawson, E., and Nielsen, L. "Automated Cryptoanalysis of XOR Plaintext Strings." *Cryptologia*, April 1996.

DEFW96 Dean, D.; Felten, E.; and Wallach, D. "Java Security: From HotJava to Netscape and Beyond." *Proceedings IEEE Symposium on Security and Privacy*, IEEE, May 1996.

DENN71 Denning, P. "Third Generation Computer Systems." *ACM Computing Surveys*, December 1971.

DIFF76 Diffie, W., and Hellman, M. "New Directions in Cryptography." *Proceedings of the AFIPS National Computer Conference*, June 1976.

DIFF79 Diffie, W., and Hellman, M. "Privacy and Authentication: An Introduction to Cryptography." *Proceedings of the IEEE*, March 1979.

DIMI07 Dimitriadis, C. "Analyzing the Security of Internet Banking Authentication Mechanisms." *Information Systems Control Journal*, Vol. 3, 2007.

DOJ00 U.S. Department of Justice. *The Electronic Frontier: The Challenge of Unlawful Conduct Involving the Use of the Internet.* March 2000. http://www.justice.gov/publications/publications_e.html

DU11 Du, W. "SEED: Hands-On Lab Exercises for Computer Security Education." *IEEE Security & Privacy*, September/October 2011.

EGEL12 Egele, M.; Scholte, T.; Kirda, E.; and Kruegel, C. "A Survey on Automated Dynamic Malware Analysis Techniques and Tools." *ACM Computing Surveys*, Vol. 44, No. 2, Article 6, February 2012.

EMBL08 Embleton, S.; Sparks, S.; and Zou, C. "SMM Rootkits: a New Breed of OS-Independent Malware." *Proceedings of the 4th International Conference on Security and Privacy in Communication Networks*, ACM, September 2008.

ENGE80 Enger, N., and Howerton, P. *Computer Security.* New York: Amacom, 1980.

ENIS09 European Network and Information Security Agency. *Cloud Computing: Benefits, Risks and Recommendations for Information Security.* ENISA Report, November 2009.

ENIS15 European Network and Information Security Agency. *Cloud Security Guide for SMEs.* ENISA Report, April 2015.

FEIS73 Feistel, H. "Cryptography and Computer Privacy." *Scientific American*, May 1973.

FLUH01 Fluhrer, S.; Mantin, I.; and Shamir, A. "Weakness in the Key Scheduling Algorithm of RC4." *Proceedings, Workshop in Selected Areas of Cryptography*, 2001.

FORR06 Forristal, J. "Physical/Logical Convergence." *Network Computing*, November 23, 2006.

FOSS10 Fossi M. et al, "Symantec Report on Attack Kits and Malicious Websites." Symantec, 2010.

FRAH15 Frahim, J., et al. *Securing the Internet of Things: A Proposed Framework.* Cisco White Paper, March 2015.

GARC09 Garcia-Teodoro, P., et al. "Anomaly-Based Network Intrusion Detection: Techniques, Systems and Challenges." *Computer & Security*, Vol. 28, 2009.

GAUD00 Gaudin, S. "The Omega Files." *Network World*, June 26, 2000.

GEOR12 Georgiev, M., et al. "The Most Dangerous Code in the World: Validating SSL Certificates in Non-Browser Software." *ACM Conference on Computer and Communications Security*, 2012.

GOLD10 Gold, S. "Social Engineering Today: Psychology, Strategies and Tricks." *Network Security*, November 2010.

GOOD11 Goodin, D. "Hackers Break SSL Encryption Used by Millions of Sites." *The Register*, September 19, 2011.

GOOD12a Goodin, D. "Why Passwords Have Never Been Weaker—and Crackers Have Never Been Stronger." *Ars Technica*, August 20, 2012.

GOOD12b Goodin, D. "Crack in Internet's Foundation of Trust Allows HTTPS Session Hijacking." *Ars Technica*, September 13, 2012.

GOOD14 Goodin, D. "Critical Crypto Bug in OpenSSL Opens Two-Thirds of the Web to Eavesdropping." *Ars Technica*, April 7, 2014.

GOOD17 Goodin, D. "Wanna Decryptor ransomware: What is it, and how does it work?." Ars Technica, May 15, 2017.

GOTT99 Gotterbarn, D. "How the New Software Engineering Code of Ethics Affects You." IEEE Software, November/December 1999.

GOWA01 Goldberg, I., and Wagner, D. "Randomness and the Netscape Browser." *Dr. Dobb's Journal*, July 22, 2001.

GOYE99 Goyeneche, J., and Souse, E. "Loadable Kernel Modules." *IEEE Software*, January/February 1999.

GRAH72 Graham, G., and Denning, P. "Protection—Principles and Practice." *Proceedings, AFIPS Spring Joint Computer Conference*, 1972.

GRAH12 Graham-Rowe, D. "Ageing Eyes Hinder Biometric Scans." *Nature*, May 2, 2012.

GRIF76 Griffiths, P., and Wade, B. "An Authorization Mechanism for a Relational Database System." *ACM Transactions on Database Systems*, September 1976.

GRUS13 Gruschka, N.; Iacono, L.; and Sorge, C. "Analysis of the Current State in Website Certificate Validation." *Security and Communication Networks*, Wiley, 2013.

GUTM96 Gutmann, P. "Secure Deletion of Data from Magnetic and Solid-State Memory." *Proceedings of the Sixth USENIX Security Symposium*, San Jose, California, July 22–25, 1996.

GUTM02 Gutmann, P. "PKI: It's Not Dead, Just Resting." *Computer*, August 2002.

HACI02 Hacigumus, H., et al. "Executing SQL over Encrypted Data in the Database-Service-Provider Model." *Proceedings, 2002 ACM SIGMOD International Conference on Management of Data*, 2002.

HADS10 Hadsell, E., Successful SIEM and Log Management Strategies for Audit and Compliance, SANS Whitepaper, Nov 2010. https://www.sans.org/reading-room/whitepapers/auditing/successful-siem-log-management-strategies-audit-compliance-33528

HALF06 Halfond, W.; Viegas, J.; and Orso, A. "A Classification of SQL Injection Attacks and Countermeasures." *Proceedings of the IEEE International Symposium on Secure Software Engineering*, 2006.

HANS04 Hansman, S., and Hunt, R. "A Taxonomy of Network and Computer Attacks." *Computers & Security*, 2004.

HARR76 Harrison, M.; Ruzzo, W.; and Ullman, J. "Protection in Operating Systems." *Communications of the ACM*, August 1976.

HEBE92 Heberlein, L.; Mukherjee, B.; and Levitt, K. "Internetwork Security Monitor: An Intrusion-Detection System for Large-Scale Networks." *Proceedings, 15th National Computer Security Conference,* October 1992.

HERL12 Herley, C., and Oorschot, P. "A Research Agenda Acknowledging the Persistence of Passwords." *IEEE Security & Privacy*, January/February 2012.

HILT06 Hiltgen, A.; Kramp, T.; and Wiegold, T. "Secure Internet Banking Authentication." *IEEE Security and Privacy*, Vol. 4, No. 2, 2006.

HONE05 The Honeynet Project. *Knowing Your Enemy: Tracking Botnets. Honeynet White Paper*, March 2005. http://honeynet.org/papers/bots

HORO15 Horvitz, E. and Mulligan, D. "Data, Privacy, and the Greater Good." *Science*, *349*(6245), July 2015.

HOWA03 Howard, M.; Pincus, J.; and Wing, J. "Measuring Relative Attack Surfaces." *Proceedings, Workshop on Advanced Developments in Software and Systems Security*, 2003.

HOWA07 Howard, M., and LeBlanc, D. *Writing Secure Code for Windows Vista*, Redmond, WA: Microsoft Press, 2007.

HSU98 Hsu, Y. and Seymour, S. "An Intranet Security Framework Based on Short-Lived Certificates." *IEEE Internet Computing*, March/April 1998.

HUIT98 Huitema, C. *IPv6: The New Internet Protocol.* Upper Saddle River, NJ: Prentice Hall, 1998.

IMPE13 Imperva Corp. *Web Application Attack Report.* July 2013. www.imperva.com

IQBA12 Iqbal, Z. "Toward a Semantic-Enhanced Attribute-Based Access Control for Cloud Services." *IEEE 111th International Conference on Trust, Security and Privacy in Computing and Communications*, 2012.

ISF13 Information Security Forum. *The Standard of Good Practice for Information Security.* 2013. www.securityforum.org

JAME06 James, A. "UTM Thwarts Blended Attacks." *Network World*, October 2, 2006.

JUDY14 Judy, H., et al. "Privacy in Cyberspace." In [BOSW14].

JUEN85 Jueneman, R.; Matyas, S.; and Meyer, C. "Message Authentication." *IEEE Communications Magazine*, September 1985.

JUN99 Jun, B., and Kocher, P. *The Intel Random Number Generator.* Intel White Paper, April 22, 1999.

KABA14 Kabay, M., and Robertson, B. "Employment Practices and Policies." In [BOSW14].

KAND05 Kandula, S. "Surviving DDoS Attacks." *;login*, October 2005.

KELL12 Kelley, P., et al. "Guess again (and Again and Again): Measuring Password Strength by Simulating Password-Cracking Algorithms." *IEEE Symposium on Security and Privacy*, 2012.

KEPH97a Kephart, J., et al. "Fighting Computer Viruses." *Scientific American*, November 1997.

KEPH97b Kephart, J., et al. "Blueprint for a Computer Immune System." *Proceedings, Virus Bulletin International Conference*, October 1997.

KERA16 Keragala D., "Detecting Malware and Sandbox Evasion Techniques." *SANS Institute InfoSec Reading Room*, 2016.

KING06 King, N. "E-Mail and Internet Use Policy." In [BIDG06].

KIRK06 Kirk, J. "Tricky New Malware Challenges Vendors." *Network World*, October 30, 2006.

KLEI90 Klein, D. "Foiling the Cracker: A Survey of, and Improvements to, Password Security." *Proceedings, UNIX Security Workshop II*, August 1990.

KOBL92 Koblas, D., and Koblas, M. "SOCKS." *Proceedings, UNIX Security Symposium III*, September 1992.

KOCH96 Kocher, P. "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems." *Proceedings, Crypto '96*, August 1996.

KOMA11 Komanduri, S. "Of Passwords and People: Measuring the Effect of Password-Composition Policies." *CHI Conference on Human Factors in Computing Systems*, 2011.

KREI09 Kreibich, C., et al. "Spamcraft: An Inside Look At Spam Campaign Orchestration." *Proceedings of the Second USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET'09)*, April 2009.

KSHE06 Kshetri, N. "The Simple Economics of Cybercrimes." *IEEE Security and Privacy*, January/February 2006.

KUMA11 Kumar, M. "The Hacker's Choice Releases SSL DOS Tool." The *Hacker News*, October 24, 2011. http://thehackernews.com/2011/10/hackers-choice-releases-ssl-ddos-tool.html

KUPE99 Kuperman, B., and Spafford, E. "Generation of Application Level Audit Data via Library Interposition." *CERIAS Tech Report 99-11*. Purdue U., October 1999. www.cerias.purdue.edu

KUPE04 Kuperman, B. *A Categorization of Computer Security Monitoring Systems and the Impact on the Design of Audit Sources*. CERIAS Tech Report 2004-26; Purdue U. Ph.D. Thesis, August 2004. www.cerias.purdue.edu/

KURU12 Kurundkar, G.; Naik, N.; and Khamitkar, S. "Network Intrusion Detection Using SNORT." *International Journal of Engineering Research and Applications*, March–April 2012.

KUSH13 Kushner, D. "The Real Story of Stuxnet." *IEEE Spectrum*, March 2013.

LAMP69 Lampson, B. "Dynamic Protection Structures." *Proceedings, AFIPS Fall Joint Computer Conference*, 1969.

LAMP71 Lampson, B. "Protection." *Proceedings*, *Fifth Princeton Symposium on Information Sciences and Systems*, March 1971; Reprinted in *Operating Systems Review*, January 1974.

LAMP04 Lampson, B. "Computer Security in the Real World." *Computer*, June 2004.

LAW06 Law, Y.; Doumen, J.; and Hartel, P. "Survey and Benchmark of Block Ciphers for Wireless Sensor Networks." *ACM Transactions on Sensor Networks*, February 2006.

LAWT09 Lawton, G. "On the Trail of the Conficker Worm." *Computer*, June 2009.

LAZA05 Lazarevic, A.; Kumar, V.; and Srivastava, J. "Intrusion Detection: A Survey." In "Managing Cyber Threats: Issues, Approaches and Challenges," Springer, 2005.

LEUT94 Leutwyler, K. "Superhack." *Scientific American*, July 1994.

LEVI06 Levine, J.; Grizzard, J.; and Owen, H. "Detecting and Categorizing Kernel-Level Rootkits to Aid Future Detection." *IEEE Security and Privacy*, January–February 2006.

LEVI12 Levis, P. "Experiences from a Decade of TinyOS Development." *10th USENIX Symposium on Operating Systems Design and Implementation*, 2012.

LEVY96 Levy, E. "Smashing The Stack For Fun And Profit." *Phrack Magazine*, File 14, Issue 49, November 1996.

LHEE03 Lhee, K., and Chapin, S. "Buffer Overflow and Format String Overflow Vulnerabilities." *Software—Practice and Experience*, Volume 33, 2003.

LIPM00 Lipmaa, H.; Rogaway, P.; and Wagner, D. "CTR Mode Encryption." *NIST First Modes of Operation Workshop*, October 2000.

LIU03 Liu, Q.; Safavi-Naini, R.; and Sheppard, N. "Digital Rights Management for Content Distribution." *Proceedings, Australasian Information Security Workshop 2003 (AISW2003)*, 2003.

LOSH16 Loshin, P. "Details Emerging on Dyn DNS DDoS Attack, Mirai IoT Botnet." *TechTarget*, October 28, 2016. http://searchsecurity.techtarget.com/news/450401962/Details-emerging-on-Dyn-DNS-DDoS-attack-Mirai-IoT-botnet

LUK07 Luk, M., et al. "MiniSec: A Secure Sensor Network Communication Architecture." *International Conf. on Information Processing in Sensor Networks*, 2007.

LUNT89 Lunt, T. "Aggregation and Inference: Facts and Fallacies." *Proceedings, 1989 IEEE Symposium on Security and Privacy*, 1989.

LYON15 Lyon, D. "The Snowden Stakes: Challenges for Understanding Surveillance Today." *Surveillance & Society*, Vol. 13, No. 2, p. 139, 2015.

MA10 Ma, D., and Tsudik, G. "Security and Privacy in Emerging Wireless Networks." *IEEE Wireless Communications*, October 2010.

MANA11 Manadhata, P., and Wing, J. "An Attack Surface Metric." *IEEE Transactions on Software Engineering*, Vol. 37, No. 3, 2011.

MAND13 Mandiant. "APT1: Exposing One of China's Cyber Espionage Units." 2013. http://intelreport.mandiant.com

MANS01 Mansfield, T.; et al. Biometric Product Testing Final Report. National Physics Laboratory, United Kingdom, March 2001.

MART73 Martin, J. *Security, Accuracy, and Privacy in Computer Systems.* Englewood Cliffs, NJ: Prentice Hall, 1973.

MAUW05 Mauw, S., and Oostdijk, M. "Foundations of Attack Trees." *International Conference on Information Security and Cryptology*, 2005.

MAZU13 Mazurek, M., et al. "Measuring Password Guessability for an Entire University." *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, November 2013.

MCGR06 McGraw, G. *Software Security: Building Security In*. Reading, MA: Addison-Wesley, 2006.

MCCL05 McClure, R., and Kruger, I. "SQL DOM: Compile Time Checking of Dynamic SQL Statements." *27th International Conference on Software Engineering*, 2005.

MCCL12 McClure, S.; Scambray, J.; and Kurtz, G. *Hacking Exposed 7: Network Security Secrets & Solutions.* New York, NY. McGraw-Hill, 2012.

MEER10 Meer, H. "Memory Corruption Attacks the (almost) Complete History." Black Hat, Las Vegas, 2010.

MESS06 Messner, E. " All-in-one Security Devices Face Challenges." *Network World*, August 14, 2006.

MEYE13 Meyer, C.; Schwenk, J.; and Gortz, H. "Lessons Learned From Previous SSL/TLS Attacks – A Brief Chronology Of Attacks And Weaknesses." *Cryptology ePrint Archive*, 2013. http://eprint.iacr.org/2013/

MICH06 Michael, M. "Physical Security Measures." In [BIDG06].

MILL07 Miller, B.; Cooksey, G.; and Moore, F. "An Empirical Study of the Robustness of MacOS Applications Using Random Testing." *ACM SIGOPS Operating Systems Review*, Volume 41, Issue 1, January 2007.

MILL11 Miller, K. "Moral Responsibility for Computing Artifacts: The Rules." *ITPro*, May/June 2011.

MIRA05 Michael, C., and Radosevich, W. *Black Box Security Testing Tools*, US DHS BuildSecurityIn, Cigital, December 2005.

MIRK04 Mirkovic, J., and Relher, P. "A Taxonomy of DDoS Attack and DDoS Defense Mechanisms." *ACM SIGCOMM Computer Communications Review*, April 2004.

MOOR01 Moore, A.; Ellison, R.; and Linger, R. "Attack Modeling for Information Security and Survivability." *Carnegie-Mellon University Technical Note CMU/SEI-2001-TN-001*, March 2001.

MOOR02 Moore, D.; Shannon, C.; and Claffy, K. "Code-Red: A Case Study on the Spread and Victims of an Internet Worm." *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet Measurement*, November 2002.

MOOR06 Moore, D., et al. "Inferring Internet Denial-of-Service Activity." *ACM Transactions on Computer Systems*, May 2006.

MORR79 Morris, R., and Thompson, K. "Password Security: A Case History." *Communications of the ACM*, November 1979.

NARA05 Narayanan, A., and Shmatikov, V. "Fast Dictionary Attacks on Passwords Using Time-Space Tradeoff." *Proceedings, ACM Conference on Computer and Communications Security (CCS)*, 2005.

NBSP08 National Biometric Security Project. *Biometric Technology Application Manual Volume 2: Applying Biometrics.* Winter 2008.

NCAE13 National Centers of Academic Excellence in Information Assurance/Cyber Defense. *NCAE IA/CD Knowledge Units.* June 2013.

NEME10 Nemeth, E., et al. *UNIX and Linux Administration Handbook*, Fourth Edition, Upper Saddle River, NJ: Prentice Hall, 2010.

NIEM11 Niemietz, M. "UI Redressing: Attacks and Countermeasures Revisited." December 2011. http://ui-redressing.mniemietz.de

NRC91 National Research Council. *Computers at Risk: Safe Computing in the Information Age.* Washington, DC: National Academy Press, 1991.

NRC02 National Research Council. *Cybersecurity: Today and Tomorrow.* Washington, DC: National Academy Press, 2002.

NSTC11 National Science and Technology Council. *The National Biometrics Challenge.* September 2011.

OECH03 Oechslin, P. "Making a Faster Cryptanalytic Time–Memory Trade-Off." *Proceedings, Crypto 03*, 2003.

OGOR03 O'Gorman, L. "Comparing Passwords, Tokens and Biometrics for User Authentication." *Proceedings of the IEEE*, December 2003.

OPEN13 Openwall.com. *John the Ripper Password Cracker.* http://www.openwall.com/john/doc/

ORMA03 Orman, H. "The Morris Worm: A Fifteen-Year Perspective." *IEEE Security and Privacy*, September/October 2003.

OWAS13 Open Web Application Security Project. *OWASP Top 10—2013: The Ten Most Critical Web Application Security Risks.* 2013. www.owasp.org

PARK88 Parker, D.; Swope, S.; and Baker, B. *Ethical Conflicts in Information and Computer Science, Technology and Business.* Final Report, SRI Project 2609, SRI International 1988.

PENG07 Peng, T.; Leckie, C.; and Rammohanarao, K. "Survey of Network-Based Defense Mechanisms Countering the DoS and DDoS Problems." *ACM Computing Surveys*, April 2007.

PERR03 Perrine, T. "The End of crypt() Passwords . . . Please?" *;login*, December 2003.

PIEL08 Pielke, R., et al. "Normalized Hurricane Damage in the United States: 1900–2005." *Natural Hazards Review*, February 2008.

PLAT13 Plate, H.; Basile, C.; and Paraboschi, S. "Policy-Driven System Management." In [VACC13].

PLAT14 Platt, F. "Physical Threats to the Information Infrastructure." In [BOSW14].

POLL12 Poller, A., et al. "Electronic Identity Cards for User Authentication—Promise and Practice." *IEEE Security & Privacy*, January/February 2012.

POLO13 Polonetsky, J. and Tene, O., "Privacy and big data: making ends meet." *Stanford Law Review*, *66*(25), September 2013.

PORR92 Porras, P. *STAT: A State Transition Analysis Tool for Intrusion Detection.* Master's Thesis, University of California at Santa Barbara, July 1992.

PROV99 Provos, N., and Mazieres, D. "A Future-Adaptable Password Scheme." *Proceedings of the 1999 USENIX Annual Technical Conference*, 1999.

RAJA05 Rajab, M., Monrose, F., and Terzis, A., "On the Effectiveness of Distributed Worm Monitoring." *Proceedings, 14th USENIX Security Symposium*, 2005.

RIBE96 Ribenboim, P. *The New Book of Prime Number Records.* New York: Springer-Verlag, 1996.

RIVE78 Rivest, R.; Shamir, A.; and Adleman, L. "A Method for Obtaining Digital Signatures and Public Key Cryptosystems." *Communications of the ACM*, February 1978.

ROBB06a Robb, D. "Desktop Defenses." *ComputerWorld*, May 22, 2006.

ROBB06b Robb, D. "Better Security Pill Gets Suite-r." *Business Communications Review*, October 2006.

ROBS95 Robshaw, M. *Stream Ciphers*. RSA Laboratories Technical Report TR-701, July 1995.

ROGA01 Rogaway, P.; Bellare, M.; Black. J.; and Krovetz, T. "OCB: A Block-Cipher Mode of Operation for Efficient Authenticated Encryption." NIST Proposed Block Cipher Mode, August 2001. http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/ocb/ocb-spec.pdf

ROGA03 Rogaway, P.; Bellare, M.; and Black.J. "OCB: A Block-Cipher Mode of Operation for Efficient Authenticated Encryption." *ACM Transactions on Information and System Security*, August 2003.

ROSA14 Rosado, T., and Bernardino, J. "An Overview of OpenStack Architecture." *ACM IDEAS '14*, July 2014.

ROTH05 Roth, D., and Mehta, S. "The Great Data Heist." *Fortune*, May 16, 2005.

RYAN16 Ryan, M.H., "Persona Non Data: How Courts in the EU, UK and Canada Are Addressing the Issue of Communications Data Surveillance vs. Privacy Rights." Social Science Research Network, https://ssrn.com/abstract=2742057, 2016.

SA04 Standards Australia. "HB 231:2004—Information Security Risk Management Guidelines." 2004.

SADO03 Sadowsky, G. et al. *Information Technology Security Handbook.* Washington, DC: The World Bank, 2003. http://www.infodev.org/articles/information-technology-security-handbook

SALT75 Saltzer, J., and Schroeder, M. "The Protection of Information in Computer Systems." *Proceedings of the IEEE*, September 1975.

SAND94 Sandhu, R., and Samarati, P. "Access Control: Principles and Practice." *IEEE Communications Magazine*, February 1994.

SAND96 Sandhu, R., et al. "Role-Based Access Control Models." *Computer*, February 1996.

SASN13 Standards Australia and Standards New Zealand. "HB 98:2013—Security Risk Management." 2013.

SCHA01 Schaad. A.; Moffett, J.; and Jacob, J. "The Role-Based Access Control System of a European Bank: A Case Study and Discussion." *Proceedings, SACMAT '01*, May 2001.

SCHN99 Schneier, B. "Attack Trees: Modeling Security Threats." *Dr. Dobb's Journal*, December 1999.

SCHN00 Schneier, B. *Secrets and Lies: Digital Security in a Networked World.* New York: Wiley, 2000.

SCHN14 Schneier, B. "The Internet of Things Is Wildly Insecure—and Often Unpatchable." *Wired*, January 6, 2014.

SEAG08 Seagate Technology. *128-Bit Versus 256-Bit AES Encryption.* Seagate Technology Paper, 2008.

SEFR12 Serfaoui, O.; Aissaoui, M.; and Eleuldj, M. "OpenStack: Toward an Open-Source Solution for Cloud Computing." *International Journal of Computer Applications*, October 2012.

SEI06 Software Engineering Institute. *Capability Maturity Model for Development Version 1.2*. Carnegie-Mellon, August 2006.

SHAR13 Shar, L., and Tan, H. "Defeating SQL Injection." *Computer*, March 2013.

SIDI05 Sidiroglou, S., and Keromytis, A. "Countering Network Worms through Automatic Patch Generation." *IEEE Security & Privacy*, November–December 2005.

SIMP11 Simpson, S., ed. "Fundamental Practices for Secure Software Development, 2/e." SAFECode, February 2011.

SING99 Singh, S. *The Code Book: The Science of Secrecy from Ancient Egypt to Quantum Cryptography.* New York: Anchor Books, 1999.

SING03 Singer, A. "Life Without Firewalls." *;login*, December 2003.

SING04 Singer, A., and Bird, T. *Building a Logging Infrastructure*. Short Topics in System Administration, Published by USENIX Association for Sage, 2004. sageweb.sage.org

SING11 Singhal, N., and Raina, J. "Comparative Analysis of AES and RC4 Algorithms for Better Utilization." *International Journal of Computer Trends and Technology*, July–August 2011.

SKAP07 Skapinetz, K. "Virtualisation as a Blackhat Tool." *Network Security*, October 2007.

SMIT12 Smith, M.; Szongott, C.; Henne, B.; and von Voigt, G. "Big Data Privacy Issues in Public Social Media." *2012 6th IEEE International Conference on Digital Ecosystems and Technologies (DEST)*, IEEE, June 2012.

SNAP91 Snapp, S., et al. "A System for Distributed Intrusion Detection." *Proceedings, COMPCON Spring '91,* 1991.

SOUR12 Sourav, K., and Mishra, D. "DDoS Detection and Defense: Client Termination Approach." *Proceedings of the CUBE International Information Technology Conference*, 2012.

SPAF89 Spafford, E. "Crisis and Aftermath." *Communications of the ACM*, June 1989.

SPAF92a Spafford, E. "Observing Reusable Password Choices." *Proceedings, UNIX Security Symposium III*, September 1992.

SPAF92b Spafford, E. "OPUS: Preventing Weak Password Choices." *Computers and Security*, No. 3, 1992.

SPAF00 Spafford, E., and Zamboni, D. "Intrusion Detection Using Autonomous Agents." *Computer Networks*, October 2000.

SPIT03 Spitzner, L. "The Honeynet Project: Trapping the Hackers." *IEEE Security and Privacy*, March/April 2003.

STAL14 Stallings, W. *Data and Computer Communications, Tenth Edition.* Upper Saddle River, NJ: Pearson, 2014.

STAL16a Stallings, W. *Foundations of Modern Networking: SDN, NFV, QoE, IoT, and Cloud.* Hoboken, NJ: Pearson, 2016.

STAL16b Stallings, W. *Computer Organization and Architecture: Designing for Performance, Tenth Edition*. Hoboken, NJ: Pearson, 2016.

STAL16c Stallings, W. *Operating Systems: Internals and Design Principles, Ninth Edition*. Hoboken, NJ: Pearson, 2016.

STAL17 Stallings, W. *Cryptography and Network Security: Principles and Practice, Seventh Edition.* Hoboken, NJ: Pearson, 2017.

STEP93 Stephenson, P. "Preventive Medicine." *LAN Magazine*, November 1993.

STEV07 Stevens, M.; Lenstra, A.; and Weger, B. "Chosen-Prefix Collisions for MD5 and Colliding X.509 Certificates for Different Identities." in *Proceedings EUROCRYPT '07*, Springer-Verlag 2007.

STEV11 Stevens, D. "Malicious PDF Documents Explained." *IEEE Security & Privacy*, January/February 2011.

STON10 Stone, P. "Next Generation Clickjacking." BlackHat Europe 2010, April 2010. http://contextis.co.uk/files/Context-Clickjacking_white_paper.pdf

SYMA01 Symantec Corp. *The Digital Immune System.* Symantec Technical Brief, 2001.

SYMA05 Symantec Corp. *Symantec™ Central Quarantine Administrator's Guide.* Symantec Documentation, 2005.

SYMA16 Symantec. "Internet Security Threat Report, Vol. 21." April 2016.

SZUB98 Szuba, T. *Safeguarding Your Technology.* National Center for Education Statistics, NCES 98-297, 1998. nces.ed.gov/pubsearch/pubsinfo.asp?pubid=98297

TARA11 Tarala, J., Implementing the 20 Critical Controls with Security Information and Event Management (SIEM) Systems, SANS Whitepaper, Apr 2011. https://www.sans.org/reading-room/whitepapers/analyst/implementing-20-critical-controls-security-information-event-management-siem-systems-34965

TAYL11 Taylor, G., and Cox, G. "Digital Randomness." *IEEE Spectrum*, September 2011.

THOM84 Thompson, K. "Reflections on Trusting Trust (Deliberate Software Bugs)." *Communications of the ACM*, August 1984.

TIRO05 Tiron, R. "Biometrics Systems Help Strengthen Border Security in Persian Gulf Nation." *National Defense*, June 2005.

TOBA07 Tobarra, L.; Cazorla, D.; Cuartero, F.; and Diaz, G. "Analysis of Security Protocol MiniSec for Wireless Sensor Networks." *The IV Congreso Iberoamericano de Seguridad Informatica (CIBSI'07)*, November 2007.

TIMM10 Timmer, J. "32 Million Passwords Show Most Users Careless about Security." *Ars Technica*, January 21, 2010.

TRUS16 Trustwave, Inc. *Global Security Report*. trustwave.com. 2016

TSUD92 Tsudik, G. "Message Authentication with One-Way Hash Functions." *Proceedings, INFOCOM '92,* May 1992.

VACC13 Vacca, J., ed. *Computer and Information Security Handbook*, Waltham, MA: Morgan Kaufmann, 2013.

VANO94 van Oorschot, P., and Wiener, M. "Parallel Collision Search with Application to Hash Functions and Discrete Logarithms." *Proceedings, Second ACM Conference on Computer and Communications Security*, 1994.

VEEN12 van der Veen, V.; dutt-Sharma, N.; Cavallaro, L.; and Bos, H. "Memory Errors: The Past, the Present, and the Future." in *Proceedings of the 15th international conference on Research in Attacks, Intrusions, and Defenses (RAID '12)*, Springer-Verlag, pp. 86–106, 2012.

VENE06 Venema, W. "Secure Programming Traps and Pitfalls—The Broken File Shredder." *Proceedings of the AusCERT2006 IT Security Conference*, Gold Coast, Australia, May 2006.

VERA16 Veracode, Inc. *State of Software Security Report.* www.veracode.com, 2016.

VERI16 Verizon. *2013 Data Breach Investigations Report.* 2016.

VIEG01 Viega, J., and McGraw, G. *Building Secure Software: How to Avoid Security Problems the Right Way*. Reading, MA: Addison-Wesley, 2001.

WAGN00 Wagner, D., and Goldberg, I. "Proofs of Security for the UNIX Password Hashing Algorithm." *Proceedings, ASIACRYPT '00*, 2000.

WALK05 Walker, J. "802.11 Security Series. Part III: AES-based Encapsulations of 802.11 Data." Platform Networking Group, Intel Corporation, 2005.

WANG05 Wang, X.; Yin, Y.; and Yu, H. "Finding Collisions in the Full SHA-1. *Proceedings, Crypto '05*, 2005; published by Springer-Verlag.

WEAV03 Weaver, N., et al. "A Taxonomy of Computer Worms." *The First ACM Workshop on Rapid Malcode (WORM)*, 2003.

WEIR09 Weir, M., et al. "Password Cracking Using Probabilistic Context-Free Grammars." *IEEE Symposium on Security and Privacy*, 2009.

WHEE03 Wheeler, D. *Secure Programming for Linux and UNIX HOWTO*, Linux Documentation Project, 2003.

WHIT99 White, S. *Anatomy of a Commercial-Grade Immune System.* IBM Research White Paper, 1999.

WHIT13 Whitham, B., "Automating the Generation of Fake Documents to Detect Network Intruders." *International Journal of Cyber-Security and Digital Forensics*, 2013.

WIEN90 Wiener, M. "Cryptanalysis of Short RSA Secret Exponents." *IEEE Transactions on Information Theory,* Vol. IT-36, 1990.

WORL04 The World Bank. *Technology Risk Checklist.* May 2004.

YANG12 Yang, K., and Jia, X. "Attributed-Based Access Control for Multi-Authority Systems in Cloud Storage." *32nd IEEE International Conference on Distributed Computing Systems*, 2012.

YUAN05 Yuan, E., and Tong, J. "Attribute Based Access Control (ABAC) for Web Services." *Proceedings of the IEEE International Conference on Web Services*, 2005.

ZHAN10 Zhang, Y.; Monrose, F.; and Reiter, M. "The Security of Modern Password Expiration: An Algorithmic Framework and Empirical Analysis." *ACM Conference on Computer and Communications Security*, 2010.

ZHOU04 Zhou, J., and Vigna, G. "Detecting Attacks that Exploit Application-Logic Errors Through Application-Level Auditing." *Proceedings of the 20th Annual Computer Security Applications Conference (ACSAC'04)*, 2004.

ZOU05 Zou, C., et al. "The Monitoring and Early Detection of Internet Worms." *IEEE/ACM Transactions on Networking*, October 2005.

CREDITS

p. 30: Table 01.01: Computer Security Terms based on Stallings, William, Computer Security: Principles and Practice, 4e., ©2019. Reprinted and electronically reproduced by permission of pearson education, inc., new york, ny.

p. 032: Table 01.02: Threat Consequences based on RFC 4949.

p. 038: Table 01.04: Security Requirements based on FIPS 200. http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.200.pdf.

p. 83: Exercise: Problem 2.7: "This problem introduces a hash function . . . " based on Mason, WK., and The American Cryptogram Association.

p. 115: Figure 03.12: Actual Biometric Measurement Operating Characteristic Curves based on [MANSO1]. Mansfield, T., Kelly, G., Chandler, D., and Kane, J. Biometric Product Testing Final Report. National Physics Laboratory, United Kingdom, March 2001. United Kingdom National Archives, Open Government Licence v3.0,

p. 116: Figure 03.13: Basic Challenge-Response Protocols for Remote User Authentication based on [OGOR03].

p. 129: Table 04.01: Access Control Security Requirements (SP 800-171) based on NIST SP 800-171 Protecting Controlled Unclassified Information in Nonfederal Information Systems and Organizations, December 2016 National Institute of Standards and Technology (NIST), United States Department of Commerce.

p. 130: Figure 04.01: Relationship Among Access Control and Other Security Functions based on [SAND94].

p. 229: "Distributed denial-of-service (DDoS) attacks . . . " based on SOURCE: From [HONE05] The Honeynet Project. Knowing Your Enemy: Tracking Botnets. Honeynet White Paper, March 2005. http://honeynet.org/papers/bots. Uses of botnets. http://honeynet.org/node/52

p. 235: Three Techniques That Used to Change System based on [LEVI06] Levine, J.; Grizzard, J.; and Owen, H. "Detecting and Categorizing Kernel-Level Rootkits to Aid Future Detection." IEEE Security and Privacy, January–February 2006, Page(s): 24–32 http://ieeexplore.ieee.org/document/1588822/

p. 276: "Performing a remote root compromise . . . " based on NIST SP 800-61 (Computer Security Incident Handling Guide, August 2012). National Institute of Standards and Technology, United States Department of Commerce.

p. 278: Definitions: security intrusion/intrusion detection based on SOURCE: From RFC 2828 Internet Security Glossary. Copyright (C) The IETF Trust (2007). Internet Society.

p. 281: "Run continually with minimal human . . . " based on [BALA98] Balasubramaniyan, J., Jose Omar Garcia-Fernandez, David Isaco, Eugene Spa ord, Diego Zamboni. "An Architecture for Intrusion Detection Using Autonomous Agents." Proceedings, 14th Annual Computer Security Applications Conference, 1998. The Institute of Electrical and Electronics Engineers, Inc. (IEEE).

p. 282: "Statistical: Analysis of the observed . . . " based on [GARC09] Garcia-Teodoro, P., et al. "Anomaly-based network intrusion detection: Techniques, systems and challenges", Computer & Security, vol. 28, 2009. Elsevier. http://www.sciencedirect.com/science/journal/01674048.

p. 291: Figure 08.04: Passive NIDS Sensor based on [CREM06].

p. 293: "Application layer reconnaissance and attacks . . . " based on NIST Special Publication SP 800-94, SP 800-94 rev 1 (draft), July 2012. National Institute of Standards and Technology, United States Department of Commerce.

p. 294: "Denial-of-service (DoS) attacks . . . " based on NIST Special Publication SP 800-94, SP 800-94 rev 1 (draft), July 2012. National Institute of Standards and Technology, United States Department of Commerce.

p. 312: "All traffic from inside to outside . . . " based on [BELL94] Bellovin, S., and Cheswick, W. "Network Firewalls." IEEE Communications Magazine, September 1994. The Institute of Electrical and Electronics Engineers, Inc. (IEEE).

p. 312: "IP Address and Protocol Values . . . " based on NIST SP 800-41 (Guidelines on Firewalls and Firewall Policy, September 2009). National Institute of Standards and Technology, United States Department of Commerce.

p. 332: Figure 09.05: Placement of Malware Monitors Security and Privacy based on [SIDI05]. Sidiroglou, S., and Keromytis, A. "Countering Network Worms Through Automatic Patch Generation.", Columbia University, Figure 1, page 3, November-December 2005. http://www1.cs.columbia.edu/~angelos/Papers/2005/j6ker3.pdfIEEE

p. 333: Figure 09.06: Unified Threat Management Appliance based on [JAME06].

p. 343: Buffer overrun based on NIST Glossary of Key Information Security Terms National Institute of Standards and Technology, United States Department of Commerce.

p. 358: C programming text from Knoppix Linux system, Pentium processor, using the GNU GCC compiler and GDB debugger. The Free Software Foundation (FSF) GCC gcc@gcc.gnu.org. CC BY-ND 3.0. Creative Commons Attribution-No Derivative Works 3.0 license.

p. 412: Script Codes from CGI script, XSS UNIX finger commands codes.

p. 423: "Install and patch the operating system . . . " based on Scarfone, K., Jansen, W., and Tracy, M. Guide to General Server Security, NIST Special Publication 800-123, July 2008. National Institute of Standards and Technology, United States Department of Commerce.

p. 446: Cloud computing based on NIST SP-800-145 (The NIST Definition of Cloud Computing, September 2011). National Institute of Standards and Technology, United States Department of Commerce.

p. 451: The NIST cloud computing reference architecture focuses . . . based on NIST SP 500-292 (NIST Cloud Computing Reference Architecture, September 2011). National Institute of Standards and Technology, United States Department of Commerce.

p. 458: Cloud-specific security threats based on The Cloud Security Alliance [CSA13]. Cloud Security Alliance. The Notorious Nine Cloud Computing Top Threats in 2013. CSA Report, February 2013. https://downloads.cloudsecurityalliance.org/initiatives/top_threats/The_Notorious_Nine_Cloud_Computing_Top_Threats_in_2013.pdf.

p. 482: Standard: IT security management based on ISO 13335 (Management of information and communications technology security). International Organization for Standardization. © William Stallings.

p. 484: Process steps for managing information security that . . . adapted from table 1 in ISO 27005 and part of figure 1 in ISO 31000.

p. 485: Organizational security policy topics . . . . adapted from the details provided in various sections of ISO 13335.

p. 530: Three elements of information system . . . based on Platt, F. "Physical Threats to the Information Infrastructure." In Bosworth, S.; Kabay, M.; and Whyne, E., eds. Computer Security Handbook. New York: Wiley, 2009.

p. 532: Table 16.01: Characteristics of Natural Disasters based on data from ComputerSite Engineering, Inc.

p. 534: Table 16.04: Temperature Thresholds for Damage to Computing Resources based on data from National Fire Protection Association.

p. 538: Fire and Smoke measures based on MARTIN, SECURITY, ACCURACY, AND PRIVACY IN COMPUTER SYSTEMS., 1st, ©1974. Printed and Electronically reproduced by permission of Pearson Education, Inc., Upper Saddle River, New Jersey.

p. 544: Figure 16.03: Convergence Example based on [FORR06].

p. 549: Table 16.07: World Bank Physical Security Checklist based on SOURCE: From "World Bank Integrator Unit and TRE Security Team Collaboration. 2004. Technology Risk Checklist 7.3, pp. 13-14. © The World Bank. https://www.cccure.org/Documents/tra/technologyriskchecklist.pdf License: Creative Commons Attribution license (CC BY 3.0 IGO)." World Bank Group. Used by permissions.

p. 552: "Security awareness is explicitly required for all . . . " adapted from NIST SP 800-16 (Information Technology Security Training Requirements: A Role- and Performance-Based Model). National Institute of Standards and Technology, United States Department of Commerce.

p. 554: "Awareness tools are used to promote information . . . " adapted from NIST SP 800-100, Information Security Handbook: A Guide for Managers. National Institute of Standards and Technology, United States Department of Commerce.

p. 555: "Goal 1: Raise staff awareness of information . . . " adapted from Szuba, T. Safeguarding Your Technology. National Center for Education Statistics, NCES 98-297, 1998, nces.ed.gov/pubsearch/pubsinfo.asp?pubid=98297 U.S. Department of Education.

p. 558: "Have an investigation agency . . . " based on Sadowsky, G. et al. Information Technology Security Handbook. Washington, DC: The World Bank, 2003 http://www.infodev-security.net/handbook. The International Bank for Reconstruction and Development.

p. 560: Policy Issues based on [KING06].

p. 562: "Responding to incidents systematically so . . . " based on Cichonski, P., et al. Computer Security Incident Handling Guide. NIST Special Publication 800-61, August 2012. National Institute of Standards and Technology, United States Department of Commerce.

p. 565: "Taking action to protect systems and networks . . . " based on Carnegie-Mellon Software Engineering Institute. Handbook for Computer Security Incident Response Teams (CSIRTs). CMU/SEI-2003-HB-002, April 2003. Carnegie Mellon University Press.

p. 571: Table 18.01: Security Audit Terminology based on [NIST95] National Institute of Standards and Technology. An Introduction to Computer Security: The NIST Handbook. Special Publication 800-12, October 1995. National Institute of Standards and Technology, United States Department of Commerce.

p. 576: Code of Practice for Information Security Management based on ISO 27002 International Organization for Standardization (ISO).

p. 580: "The date and time the access was attempted . . . " based on [NIST95] National Institute of Standards and Technology. An Introduction to Computer Security: The NIST Handbook. Special Publication 800-12, October 1995. National Institute of Standards and Technology, United States Department of Commerce.

p. 583: Figure 18.05: Windows System Log Entry Example based on Microsoft® Windows, Microsoft Corporation. Reprinted with permission Microsoft Corporation.

p. 584: "Robust filtering: Original syslog . . . " from Kent, K., and Souppaya, M. Guide to Computer Security Log Management. NIST Special Publication 800-92, September 2006. National Institute of Standards and Technology, United States Department of Commerce.

p. 599: Figures 3: Command Codes from UNIX commands codes.

p. 601: Computers as targets . . . from [DOJ00] U.S. Department of Justice.

p. 621: Figure 19.06: ACM Code of Ethics and Professional Conduct based on Excerpt reprinted courtesy of ACM, Inc.

p. 622: Figure 19.08: AITP Standard of Conduct based on From Copyright © 2006, Association of Information Technology Professionals. Association of Information Technology. Used by permissions.

p. 622: Figure 19.07: IEEE Code of Ethics based on Reprinted with permission of IEEE with the copyright notice © Copyright 2017 IEEE included.

p. 625: Table 19.03: OECD Guidelines on the Protection of Privacy based on OECD Guidelines on the Protection of Privacy and Transborder Flows of Personal Data, Annex to the Recommendation of the Council of 23rd September 1980: GUIDELINES GOVERNING THE PROTECTION OF PRIVACY AND TRANSBORDER FLOWS OF PERSONAL DATA, Part Two: Basic Principles of National Application http://www.oecd.org/sti/ieconomy/oecdguidelinesontheprotectionofprivacyandtransborderflowsofpersonaldata.htm.

p. 649: "Hardware efficiency: Unlike the three chaining modes . . . " based on Lipmaa, H., Rogaway, P., and Wagner, D. "CTR Mode Encryption." NIST First Modes of Operation Workshop, October 2000. National Institute of Standards and Technology, United States Department of Commerce.

p. 663: Design objectives for HMAC from https://www.ietf.org/rfc/rfc2104.txt.

p. 700: "A router advertisement . . . " based on Huitema, C. IPv6: The New Internet Protocol. Upper Saddle River, NJ: Prentice Hall, 1998. Pearson Education.

Page 724: "Wireless Network Threats" based on Choi, MK., Robles, RJ., Hong, CH., Kim, TH. "Wireless Network Security: Vulnerabilities, Threats and Countermeasures." International Journal of Multimedia and Ubiquitous Engineering, July 2008. Science & Engineering Research Support Society.

p. 726: "Securing Wireless Networks" based on Choi, MK., Robles, RJ., Hong, CH., Kim, TH. "Wireless Network Security: Vulnerabilities, Threats and Countermeasures." International Journal of Multimedia and Ubiquitous Engineering, July 2008. Science & Engineering Research Support Society.

p. 729: Figure 22.07: The Heartbleed Exploit based on From "Heartbleed-The Open SSL Heartbeat Exploit" Copyright © 2014 BAE Systems Applied Intelligence. Reprinted with permission.

INDEX

B

C

D

E

I

J

K

L

M

P

Q

R

S

T

U

V

W

X

Z

CHAPTER 25

LINUX SECURITY

Like other general-purpose operating systems, Linux's wide range of features presents a broad attack surface. Even so, by leveraging native Linux security controls, carefully configuring Linux applications, and deploying certain add-on security packages, you can create highly secure Linux systems.

25.1 INTRODUCTION

Since Linus Torvalds created Linux in 1991, more or less on a whim, Linux has evolved into one of the world's most popular and versatile operating systems. Linux is free, open-sourced, and available in a wide variety of "distributions" targeted at almost every usage scenario imaginable. These distributions range from conservative, commercially supported versions such as Red Hat Enterprise Linux, to cutting-edge, completely free versions such as Ubuntu, to stripped-down but hyperstable "embedded" versions (designed for use in appliances and consumer products) such as uClinux.

The study and practice of Linux security therefore has wide-ranging uses and ramifications. New exploits against popular Linux applications can affect many thousands of users around the world. New Linux security tools and techniques have just as profound of an impact, albeit a much more constructive one.

In this chapter, we'll examine the Discretionary Access Controls–based security model and architecture common to all Linux distributions and to most other UNIX-derived and UNIX-like operating systems (and also, to a surprising degree, to Microsoft Windows). We'll discuss the strengths and weaknesses of this ubiquitous model, typical vulnerabilities and exploits in Linux, best practices for mitigating those threats, and improvements to the Linux security model that are only slowly gaining popularity but that hold the promise to correct decades-old shortcomings in this platform.

25.2 LINUX'S SECURITY MODEL

Linux's traditional security model can be summed up quite succinctly: people or processes with "root" privileges can do anything; other accounts can do much less.

From the attacker's perspective, the challenge in cracking a Linux system therefore boils down to gaining root privileges. Once that happens, attackers can erase or edit logs; hide their processes, files, and directories; and basically redefine the reality of the system as experienced by its administrators and users. Thus, as it's most commonly practiced, Linux security (and UNIX security in general) is a game of "root takes all."

How can such a powerful operating system get by with such a limited security model? In fairness, many Linux system administrators fail to take full advantage of the security features available to them (features we're about to explore in depth). People can and do run robust, secure Linux systems by making careful use of native Linux security controls, plus selected add-on tools such as sudo or Tripwire. However, the crux of the problem of Linux security in general is that, like the UNIX operating systems on which it was based, Linux's security model relies on **Discretionary Access Controls** (DAC) that we introduced in Chapter 4.

In the Linux DAC system, there are users, each of which belongs to one or more groups; and there are also **objects**: files and directories. Users read, write, and execute these objects, based on the objects' **permissions**, of which each object has three sets: one each defining the permissions for the object's user-owner, group-owner, and "other" (everyone else). These permissions are enforced by the Linux kernel, the "brain" of the operating system.

Because a process/program is actually just a file that gets copied into executable memory when run, permissions come into play twice with processes. Prior to being executed, a program's file-permissions restrict who can execute, access, or change it. When running, a process normally "runs as" (with the identity of) the user and group of the person or process that executed it.

Because processes "act as" users, if a running process attempts to read, write, or execute some other object, the kernel will first evaluate that object's permissions against the process's user and group identity, just as though the process was an actual human user. This basic transaction, wherein a **subject** (user or process) attempts some action (read, write, or execute) against some object (file, directory, or special file), is illustrated in Figure 25.1.

Whoever owns an object can set or change its permissions. Herein lies the Linux DAC model's real weakness: The system **superuser** account, called "root," has the ability to both take ownership and change the permissions of all objects in the system. And as it happens, it's not uncommon for both processes and administrator-users to routinely run with root privileges, in ways that provide attackers with opportunities to hijack those privileges.

Those are the basic concepts behind the Linux DAC model. The same concepts in a different arrangement will come into play later when we examine Mandatory Access Controls such as SELinux. Now, let's take a closer look at how the Linux DAC implementation actually works.

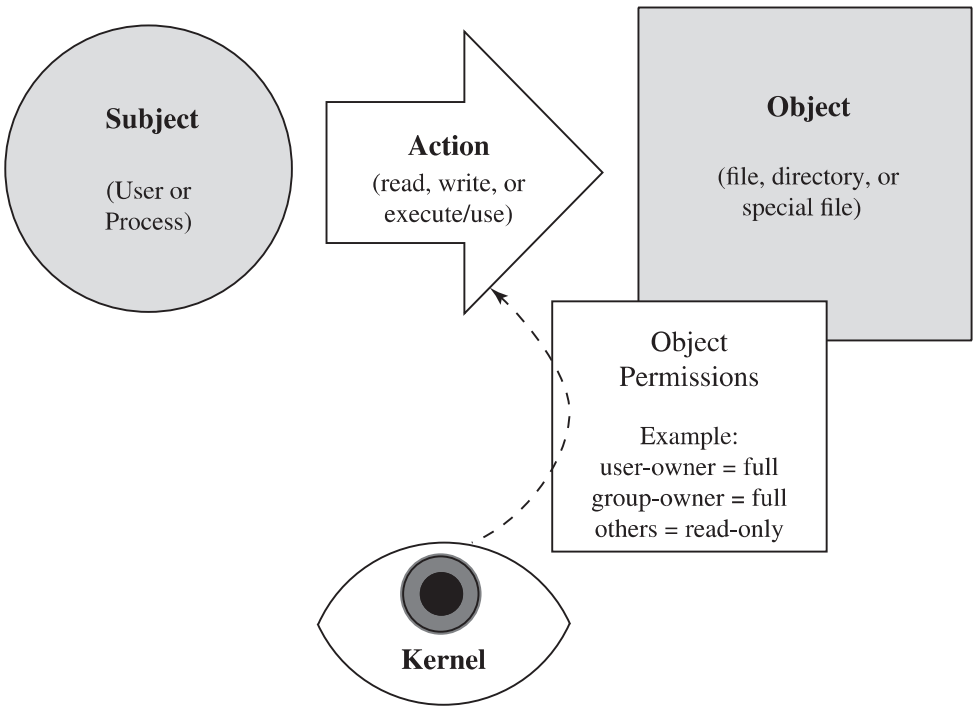


Figure 25.1 Linux Security Transactions

25.3 THE LINUX DAC IN DEPTH: FILE-SYSTEM SECURITY

So far, we haven't said anything about memory, device drivers, named pipes, and other system resources. Isn't there more to system security than users, files, and directories? Yes and no: In a sense, Linux treats *everything* as a file.

Documents, pictures, and even executable programs are very easy to conceptualize as files on your hard disk. But although we *think* of a directory as a container of files, in UNIX a directory is actually itself a file containing a list of other files.

Similarly, the CD-ROM drive attached to your system seems tangible enough, but to the Linux kernel, it is a file: the "special" device-file `/dev/cdrom`. To send data from or write data to the CD-ROM drive, the Linux kernel actually reads to and writes from this special file. (Actually, on most systems, "`/dev/cdrom`" is a symbolic link to `/dev/hdb` or some other special file, and a symbolic link is in turn nothing more than a file that contains a pointer to another file.)

Other special files, such as named pipes, act as input/output (I/O) "conduits," allowing one process or program to pass data to another. One common example of a named pipe on Linux systems is /dev/urandom: When a program reads this file, `/dev/urandom` returns random characters from the kernel's random number generator.

These examples illustrate how in Linux/UNIX, *nearly everything* is represented by a file. Once you understand this, it's much easier to understand why file-system security is so important (and how it works).

Users, Groups, and Permissions

There are two things on a UNIX system that aren't represented by files: user accounts and group accounts, which, for short, we can call **users** and **groups**. (Various files contain information about a system's users and groups, but none of those files actually represents them.)

A user account represents someone or something capable of using files. As we saw in the previous section, a user account can be associated both with actual human beings and with processes. The standard Linux user account "lp," for example, is used by the Line Printer Daemon (lpd): the lpd program runs as the user lp.

A group account is simply a list of user accounts. Each user account is defined with a **main group** membership, but may in fact belong to as many groups as you want or need it to. For example, the user "maestro" may have a main group membership in "conductors" and also belong to the group "pianists."

A user's main group membership is specified in the user account's entry in `/etc/password`; you can add that user to additional groups by editing `/etc/group` and adding the username to the end of the entry for each group the user needs to belong to, or via the **usermod** command (see the usermod(8) manpage for more information).

Listing 25-1 shows "maestro"'s entry in the file `/etc/password`, and Listing 25-2 shows part of the corresponding `/etc/group` file:

Listing 25-1: An /etc/password Entry for the User "maestro"

```
maestro:x:200:100:Maestro Edward Hizzersands:/home
/maestro:/bin/bash
```

Listing 25-2: Two /etc/group Entries

```
conductors:x:100:
pianists:x:102:maestro,volodya
```

In Listing 25-1, we see that the first field contains the name of the user account, "maestro;" the second field ("x") is a placeholder for maestro's password (which is actually stored in `/etc/shadow`); the third field shows maestro's numeric user-ID (or "uid," in this case "200"); and the fourth field shows the numeric group-ID (or "gid," in this case "100") of maestro's main group membership. The remaining fields specify a comment, maestro's home directory, and maestro's default login shell.

In Listing 25-2, from `/etc/group`, each line simply contains a group-name, a group-password (usually unused — "x" is a placeholder), and numeric group-ID (gid), and a comma-delimited list of users with "secondary" memberships in the group. Thus, we see that the group "conductors" has a gid of "100", which corresponds to the gid specified as maestro's main group in Listing 25-1; and also that the group "pianists" includes the user "maestro" (plus another named "volodya") as a secondary member.

The simplest way to modify `/etc/password` and `/etc/group` in order to create, modify, and delete user accounts is via the commands **useradd**, **usermod**, and **userdel**, respectively. All three of these commands can be used to set and modify group-memberships, and all three commands are well documented in their respective manpages. (To see a quick usage summary, you can also type the command followed by "`--help`," for example, "`useradd --help`".)

So we've got user accounts, which are associated with different group accounts. Just what is all this good for?

Simple File Permissions

Each file on a UNIX system (which, as we've seen, means practically every single thing on a UNIX system) has two owners: a user and a group, each with its own set of permissions that specify what the user or group may do with the file (read it, write to it, or delete it, and execute it). A third set of permissions pertains to **other**, that is, user accounts that don't own the file or belong to the group that owns it.

Listing 25-3 shows a "long file-listing" for the file `/home/maestro/baton_dealers.txt`:

Listing 25-3: File-Listing Showing Permissions

```
-rw-rw-r-- 1 maestro conductors 35414 Mar 25 01:38
baton_dealers.txt
```

Permissions are listed in the order "user permissions, group permissions, other permissions." Thus, we see that for the file shown in Listing 25-3, its user-owner ("maestro") may read and write/delete the file ("rw-"); its group-owner ("conductors") may also read and write/delete the file ("rw-"); but that other users (who are neither "maestro" nor members of "conductors") may only read the file.

There's a third permission besides "read" and "write": "execute," denoted by "x" (when set). If maestro writes a shell script named "`punish_bassoonists.sh`", and if he sets its permissions to "-rwxrw-r--", then maestro will be able to execute his script by entering the name of the script at the command line. If, however, he forgets to do so, he won't be able to run the script, even though he owns it. Permissions are usually set via the `chmod` command (short for "change mode").

Directory Permissions

Directory permissions work slightly differently from permissions on regular files. "Read" and "write" are similar; for directories these permissions translate to "list the directory's contents" and "create or delete files within the directory", respectively. "Execute" is less intuitive; for directories, "execute" translates to "use anything within or change working directory to this directory."

That is, if a user or group has execute permissions on a given directory, the user or group can list that directory's contents, read that directory's files (assuming those individual files' own permissions include this), and change its own working directory to that directory, as with the command "cd". If a user or group does not have execute permissions on a given directory, its permissions will be unable to list or read anything in it, regardless of the permissions set on the things inside.

(Note if you lack execute permissions on a directory, but do have read permissions on the directory, and you try to list its contents with ls, you will receive an error message that, in fact, lists the directory's contents. But this doesn't work if you have neither read nor execute permissions on the directory.)

Suppose our example system has a user named biff who belongs to the group "drummers". And suppose further his home directory contains a directory named, `extreme_casseroles` that biff wishes to share with his fellow percussionists. Listing 25-4 shows how biff might set that directory's permissions:

Listing 25-4: A Group-Readable Directory

```
bash-$ chmod g+rx extreme_casseroles
bash-$ ls -l extreme_casseroles

drwxr-x--- 8 biff drummers 288 Mar 25 01:38
extreme_casseroles
```

Per Listing 25-4, only biff has the ability to create, change, or delete files inside `extreme_casseroles`. Other members of the group "drummers" may list its contents and cd to it. Everyone else on the system, however (except root, who is always all powerful), is blocked from listing, reading, cd-ing, or doing anything else with the directory.

The Sticky Bit

Suppose our drummer friend Biff wants to allow his fellow drummers not only to read his recipes, but also to add their own. As we saw last time, all he needs to do is set the "group-write" bit for this directory, like this:

```
chmod g+w ./extreme_casseroles
```

There's only one problem with this: "write" permissions include not only the ability to create new files in this directory, but also to delete them. What's to stop one of his drummer pals from deleting other people's recipes? The "sticky bit."

In older UNIX operating systems, the sticky bit was used to write a file (program) to memory so it would load more quickly when invoked. On Linux, however, it serves a different function: When you set the sticky bit on a directory, it limits users' ability to delete things in that directory. With the sticky bit set, to delete a given file in the directory, it is not sufficient that group-write permissions are set on the directory, and you belong to the group that owns the directory. Rather, to delete a file in a directory with the sticky bit set, you must either own that file or own the directory.

To set the sticky bit, issue the command

```
chmod +t  directory_name
```

In our example, this would be "`chmod +t extreme_casseroles`". If we set the sticky bit on extreme_casseroles then do a long listing of the directory itself, using "`ls -ld extreme_casseroles`", we'll see

```
drwxrwx--T 8 biff drummers 288 Mar 25 01:38
extreme_casseroles
```

Note the "T" at the end of the permissions string. We'd normally expect to see either "x" or "-" there, depending on whether the directory is "other-writable". "T" denotes that the directory is not "other-executable" but has the sticky bit set. A lower-case "t" would denote that the directory is other-executable and has the sticky bit set.

To illustrate what effect this has, suppose a listing of the contents of `extreme_casseroles/` looks like this (see Listing 25-5):

Listing 25-5: Contents of extreme_casseroles/

```
drwxrwxr-T 3 biff drummers 192 2004-08-10 23:39 .
drwxr-xr-x 3 biff drummers 4008 2004-08-10 23:39 ..
-rw-rw-r-- 1 biff drummers   18 2004-07-08 07:40
chocolate_turkey_casserole.txt
-rw-rw-r-- 1 biff drummers   12 2004-08-08 15:10
pineapple_mushroom_suprise.txt
drwxr-xr-x 2 biff drummers   80 2004-08-10 23:28 src
```

Suppose further the user "crash" tries to delete the recipe-file `pineapple_mushroom_surprise.txt`, which crash finds offensive. crash expects this to work, because he belongs to the group "drummers" and the group-write bit is set on this file.

However, remember, biff just set the parent directory's sticky bit. crash's attempted deletion will fail, as we see in Listing 25-6 (user input in boldface):

Listing 25-6: Attempting Deletion with Sticky Bit Set

```
crash@localhost:/extreme_casseroles>
rm pineapple_mushroom_suprise.txt
rm: cannot remove 'pineapple_mushroom_suprise.txt':
Operation not permitted
```

The sticky bit only applies to the directory's first level downward. In Listing 25-5, you may have noticed that besides the two nasty recipes, `extreme_casseroles/` also contains another directory, "`src`". The contents of `src` will not be affected by `extreme_casserole's` sticky bit (though the directory `src` itself will be). If biff wants to protect `src's` contents from group deletion, he'll need to set `src's` own sticky bit.

Setuid and Setgid

Now, we come to two of the most dangerous permissions bits in the UNIX world: setuid and segid. If set on an executable binary file, the setuid bit causes that program to run as its owner, no matter who executes it. Similarly, the setgid bit, when set on an executable, causes that program to run as a member of the group that owns it, again regardless of who executes it.

By *run as*, we mean "to run with the same privileges as." For example, suppose biff writes and compiles a C program, "`killpineapple`", that behaves the same as the command "`rm /extreme_casseroles/pineapple_mushroom_surprise.txt`". Suppose further biff sets the setuid bit on killpineapple, with the command "`chmod +s ./killpineapple`", and also makes it group executable. A long-listing of `killpineapple` might look like this:

```
-rwsr-xr-- 1 biff drummers 22 2004-08-11 23:01
killpineapple
```

If crash runs this program he will finally succeed in his quest to delete the Pineapple Mushroom Surprise recipe: `killpineapple` will run as though biff had executed it. When `killpineapple` attempts to delete `pineapple_mushroom_suprise.txt`, it will succeed because the file has user-write permissions and killpineapple is acting as its user-owner, biff.

Note that setuid and setgid are *very dangerous* if set on any file owned by root or any other privileged account or group. We illustrate setuid and setgid in this discussion so you understand what they do, not because you should actually *use* them for anything important. The command "`sudo`" is a much better tool for delegating root's authority.

Also note that if you want a program to run setuid, that program must be group-executable or other-executable, for obvious reasons. Note the Linux kernel ignores the setuid and setgid bits on shell scripts; these bits only work on binary (compiled) executables.

Setgid works the same way, but with group permissions: If you set the setgid bit on an executable file via the command "`chmod g+s filename`", and if the file is also "other-executable" (-r-xr-sr-x), then when that program is executed, it will run with the group-ID of the file rather than of the user who executed it.

In the preceding example, if we change `killpineapple`'s "other" permissions to "r-x" (`chmod o+x killpineapple`) and make it setgid (`chmod g+s killpineapple`), then no matter who executes `killpineapple`, `killpineapple` will exercise the permissions of the "drummers" group, because drummers is the group-owner of `killpineapple`.

Setgid and Directories

Setuid has no effect on directories, but setgid does. Normally, when you create a file, it's automatically owned by your user-ID and your (primary) group-ID. For example,

if biff creates a file, the file will have a user-owner of "biff" and a group-owner of "drummers" (assuming that "drummers" is biff's primary group, as listed in `/etc/passwd`).

Setting a directory's setgid bit, however, causes any file created in that directory to inherit the directory's group-owner. This is useful if users on your system tend to belong to secondary groups and routinely create files that need to be shared with other members of those groups.

For example, if the user "animal" is listed in `/etc/group` as being a secondary member of "drummers" but is listed in `/etc/passwd` has having a primary group of "muppets", then animal will have no trouble creating files in the `extreme_casseroles/` directory, whose permissions are set to drwxrwx--T. However, by default, animal's files will belong to the group muppets, not to drummers, so unless animal manually reassigns his files' group-ownership (`chgrp drummers newfile`) or resets their other-permissions (`chmod o+rw newfile`), then other members of drummers won't be able to read or write animal's recipes.

If, on the other hand, biff (or root) sets the setgid bit on `extreme_casseroles/` (`chmod g+s extreme_casseroles`), then when animal creates a new file therein, the file will have a group-owner of "drummers", just like `extreme_casseroles/` itself. Note that all other permissions still apply: If the directory in question isn't group-writable, then the setgid bit will have no effect (because group members won't be able to create files inside it).

Numeric Modes

So far, we've been using mnemonics to represent permissions: "r" for read, "w" for write, and so on. But internally, Linux uses numbers to represent permissions; only user-space programs display permissions as letters. The chmod command recognizes both mnemonic permission modifiers ("u+rwx, go-w") and **numeric modes**.

A numeric mode consists of four digits: as you read left to right, these represent special permissions, user permissions, group permissions, and other permissions (where, you'll recall, "other" is short for "other users not covered by user permissions or group permissions"). For example, 0700 translates to "no special permissions set, all user permissions set, no group permissions set, no other permissions set."

Each permission has a numeric value, and the permissions in each digit-place are additive: The digit represents the sum of all permission-bits you wish to set. If, for example, user permissions are set to "7", this represents 4 (the value for "read") plus 2 (the value for "write") plus 1 (the value for "execute").

As just mentioned, the basic numeric values are 4 for read, 2 for write, and 1 for execute. (I remember these by mentally repeating the phrase, "read-write-execute, 4-2-1.") Why no "3," you might wonder? Because (a) these values represent bits in a binary stream and are therefore all powers of 2, and (b) this way, no two combinations of permissions have the same sum.

Special permissions are as follows: 4 stands for setuid, 2 stands for setgid, and 1 stands for sticky bit. For example, the numeric mode 3000 translates to "setgid set, sticky bit set, no other permissions set" (which is, actually, a useless set of permissions).

Here's one more example of a numeric mode. If I issue the command "`chmod 0644 mycoolfile`," I'll be setting the permissions of "`mycoolfile`" as shown in Figure 25.2.

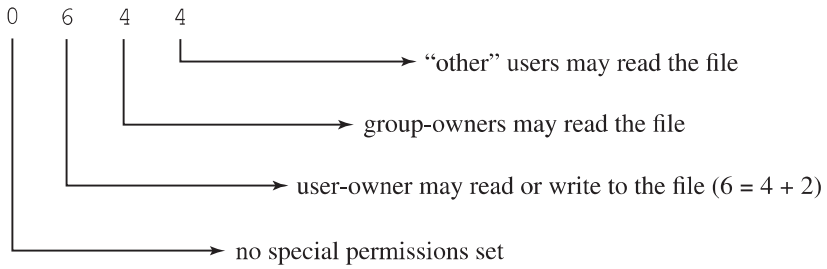


Figure 25.2 Permissions on mycoolfile

For a more complete discussion of numeric modes, see the Linux "`info`" page for "`coreutils`," node "Numeric Modes" (that is, enter the command "`info coreutils numeric`").

Kernel Space versus User Space

It is a simplification to say that users, groups, files, and directories are all that matter in the Linux DAC: Memory is important, too. Therefore, we should at least briefly discuss kernel space and user space.

Kernel space refers to memory used by the Linux kernel and its loadable modules (e.g., device drivers). **User space** refers to memory used by all other processes. Because the kernel enforces the Linux DAC and, in real terms, dictates system reality, it's extremely important to isolate kernel space from user space. For this reason, kernel space is never swapped to hard disk.

It's also the reason that only root may load and unload kernel modules. As we're about to see, one of the worst things that can happen on a compromised Linux system is for an attacker to gain the ability to load kernel modules.

25.4 LINUX VULNERABILITIES

In this section, we'll discuss the most common weaknesses in Linux systems.

First, a bit of terminology. A **vulnerability** is a specific weakness or security-related bug in an application or operating system. A **threat** is the combination of a vulnerability, an attacker, and a means for the attacker to exploit the vulnerability (called an **attack vector**).

Historically, some of the most common and far-reaching vulnerabilities in default Linux installations (unpatched and unsecured) have been:

- Buffer overflows,
- Race conditions,
- Abuse of programs run "setuid root",
- Denial of service (DoS),
- Web application vulnerabilities, and
- Rootkit attacks.

While you've already had exposure to most of these concepts earlier in this text, let's take a closer look at how several of them apply to Linux.

Abuse of Programs Run "setuid root"

As we discussed in the previous section, any program whose setuid permission bit is set will run with the privileges of the user that owns it, rather than those of the process or the user executing it. A **setuid root** program is a root-owned program with its setuid bit set; that is, a program that runs as root *no matter who executes it*.

If a setuid root program can be exploited or abused in some way (e.g., via a buffer overflow vulnerability or race condition), then otherwise unprivileged users may be able to use that program to wield unauthorized root privileges, possibly including opening a **root shell** (a command-line session running with root privileges).

Running setuid root is necessary for programs that need to be run by unprivileged users yet must provide such users with access to privileged functions (e.g., changing their password, which requires changes to protected system files). But such a program must be programmed very carefully, with impeccable user-input validation, strict memory management, and so on. That is, the program must be *designed* to be run setuid (or setgid) root. Even then, a root-owned program should only have its setuid bit set if absolutely necessary.

Due to a history of abuse against setuid root programs, major Linux distributions no longer ship with unnecessary setuid-root programs. But system attackers still scan for them.

Web Application Vulnerabilities

This is a very broad category of vulnerabilities, many of which also fall into other categories in this list. It warrants its own category because of the ubiquity of the World Wide Web: there are few attack surfaces as big and visible as an Internet-facing Website.

While Web applications written in scripting languages such as PHP, Perl, and Java may not be as prone to classic buffer overflows (thanks to the additional layers of abstraction presented by those languages' interpreters), they're nonetheless prone to similar abuses of poor input-handling, including cross-site scripting, SQL code injection, and a plethora of other vulnerabilities described in depth by the Open Web Application Security Project on the Project's website (`http://www.owasp.org`). We discussed a number of these in Chapter 11.

Nowadays, few Linux distributions ship with "enabled-by-default" Web applications (such as the default cgi scripts included with older versions of the Apache Web Server). However, many users install Web applications with known vulnerabilities, or write custom Web applications having easily identified and easily exploited flaws.

Rootkit Attacks

This attack, which allows an attacker to cover his or her tracks, typically occurs *after* root compromise: If a successful attacker is able to install a rootkit before being detected, all is very nearly lost.

Rootkits began as collections of "hacked replacements" for common UNIX commands (ls, ps, etc.) that behaved like the legitimate commands they replaced, except for hiding an attacker's files, directories, and processes. For example, if an attacker was able to replace a compromised Linux system's `ls` command with a rootkit version of `ls`, then anyone executing the ls command to view files and directories would see everything except the attacker's files and directories.

In the Linux world, since the advent of **loadable kernel modules** (LKMs), rootkits have more frequently taken the form of LKMs. This is particularly devious: An **LKM rootkit** does its business (covering the tracks of attackers) *in kernel space*, intercepting system calls pertaining to any user's attempts to view the intruder's resources.

In this way, files, directories, and processes owned by an attacker are hidden even to a compromised system's standard, untampered-with commands, including customized software. Besides operating at a lower, more global level, another advantage of the LKM rootkit over traditional rootkits is that system integrity checking tools such as Tripwire won't generate alerts from system commands being replaced.

Luckily, even LKM rootkits do not always ensure complete invisibility for attackers. Many traditional and LKM rootkits can be detected with the script **chkrootkit**, available at `www.chkrootkit.org`. In general, however, if an attacker gets far enough to install an LKM rootkit, your system can be considered to be completely compromised; if and when you detect the breach (e.g., via a defaced website, missing data, suspicious network traffic), the only way to restore your system with any confidence of completely shutting out the intruder will be to erase its hard disk (or replace it, if you have the means and inclination to analyze the old one), reinstall Linux, and apply all the latest software patches.

25.5 LINUX SYSTEM HARDENING

We've seen how Linux security is supposed to work, and how it most typically fails. The remainder of this chapter will focus on how to mitigate Linux security risks at the system and application levels. We follow the broad outline introduced in Chapter 12 for hardening an operating system and applications, but focus on applying this to Linux/UNIX systems. This section deals with the first of these: OS-level security tools and techniques that protect the entire system. The final section in this chapter, on mandatory access controls, also describes system-level controls, but because this is both an advanced topic and an emerging technology (in the Linux world), we'll consider it separately from the more fundamental controls in this section.

OS Installation: Software Selection and Initial Setup

Linux system security begins at operating system installation time: one of the most critical, system-impacting decisions a system administrator makes is what software will run on the system. Because it's hard enough for the typical, commonly overworked system administrator to find the time to secure a system's critical applications, an unused application is liable to be left in a default, unhardened and unpatched state. Therefore, it's very important that from the start, careful consideration be given to which applications should be installed, and which should not.

What software should you not install? Common sense should be your guide: for example, an SMTP (e-mail) relay shouldn't need the Apache Web Server; a database server shouldn't need an office productivity suite such as OpenOffice; and so on.

Given the plethora of roles Linux systems play (desktops, servers, laptops, firewalls, embedded systems, to name just a few), it's impossible to do much more than generalize in enumerating what software one shouldn't install. Nonetheless, here is a

list of software packages that should seldom, if ever, be installed on hardened servers, especially Internet-facing servers:

- **X Window System:** Servers are usually remotely controlled via the Secure Shell, not locally via standard desktop sessions. Even if they are, X's history of security vulnerabilities makes plaintext-console sessions a safer choice for local access.
- **RPC Services:** Remote Procedure Call is a great convenience for developers, but both difficult to track through firewalls and too reliant on the easily spoofed UDP protocol.
- **R-Services:** rsh, rlogin, and rcp use only cleartext authentication (which can be eavesdropped) or source-IP-address-based authentication (which can sometimes be spoofed). The Secure Shell (SSH), which uses strong encryption, was created specifically to replace these commands, and should be used instead.
- **inetd:** The Internet Daemon (inetd) is a poorly scaling means of starting critical network daemons, which should instead be started autonomously. inetd also tends, by default, to leave various unnecessary and potentially insecure services enabled, including RPC applications.
- **SMTP Daemons:** Traditionally, the Simple Mail Transport Protocol (SMTP) daemon Sendmail is enabled by default on many Linux distributions, despite Sendmail's history of security problems. More recent systems have replaced it with Postfix, a much more secure SMTP mail server that should be used if mail relay services are required. Such a server is unnecessary though, on any system that doesn't need to receive relayed e-mail.
- **Telnet and other cleartext-logon services:** Because it passes logon credentials (usernames and passwords) over the network unencrypted, exposing them to eavesdroppers, telnet is no longer a viable tool for remote system access (and certainly not remote administration) via untrusted networks. The Secure Shell (SSH) is almost always a better choice than telnet. FTP, POP3, and IMAP also expose user credentials in this way, though many modern FTP, POP3, and IMAP server applications now support SSL or TLS encryption.

In addition to initial software selection and installation, Linux installation utilities also perform varying amounts of initial system and software configuration, including some or all of the following:

- Setting the root password
- Creating a non-root user account
- Setting an overall system security level (usually influencing initial file-permission settings)
- Enabling a simple host-based firewall policy
- Enabling SELinux or Novell AppArmor (see Section 25.7)

Patch Management

Carefully selecting what gets installed (and what doesn't get installed) on a Linux system is an important first step in securing it. All the server applications you do

install, however, must be configured securely (the subject of Section 25.6), and they must also be kept up to date with security patches.

The bad news with patching is that you can never win the "patch rat-race": There will always be software vulnerabilities that attackers are able to exploit for some period of time before vendors issue patches for them. (As yet unpatchable vulnerabilities are known as **zero-day**, or 0-day, vulnerabilities.)

The good news is that modern Linux distributions usually include tools for automatically downloading and installing security updates, which can minimize the time your system is vulnerable to threats against which patches *are* available. For example, Red Hat, Fedora, and CentOS include **up2date** (**YUM** can be used instead); SuSE includes **YaST Online Update**; and Debian uses **apt-get**, though you must run it as a cron job for automatic updates.

Note on change-controlled systems, you should not run automatic updates, because security patches can, on rare but significant occasions, introduce instability. For systems on which availability and uptime are of paramount importance, therefore, you should stage all patches on test systems before deploying them in production.

Network-Level Access Controls

One of the most important attack-vectors in Linux threats is the network. A layered approach to security addresses not only actual vulnerabilities (e.g., patching and application-hardening), but also the means by which attackers might exploit them (e.g., the network). Network-level access controls (i.e., controls that restrict access to local resources based on the IP addresses of the networked systems attempting to access them) are, therefore, an important tool in Linux security.

LIBWRAPPERS AND TCP WRAPPERS One of the most mature network access control mechanisms in Linux is libwappers. In its original form, the software package TCP Wrappers, the daemon tcpd is used as a "wrapper" process for each service initiated by inetd.

Before allowing a connection to any given service, tcpd first evaluates access controls defined in the files `/etc/hosts.allow` and `/etc/hosts.deny`: If the transaction matches any rule in `hosts.allow` (which tcpd parses first), it's allowed. If no rule in `hosts.allow` matches, tcpd then evaluates the transaction against the rules in `hosts.deny`; if any rule in `hosts.deny` matches, the transaction is logged and denied, otherwise the transaction is permitted.

These access controls are based on the name of the local service being connected to, on the source IP address or hostname of the client attempting the connection, and on the username of the client attempting the connection (i.e., the owner of the client process). Note that client usernames are validated via the **ident** service, which unfortunately is trivially easy to forge on the client side and makes this criterion's value questionable.

The best way to configure TCP Wrappers access controls is therefore to set a "deny all" policy in `hosts.deny`, such that the only transactions permitted are those explicitly specified in `hosts.allow`.

Because, as mentioned earlier, inetd is essentially obsolete, TCP Wrappers is no longer used as commonly as libwrappers, a system library that allows applications to defend themselves by leveraging `/etc/hosts.allow` and `/etc/hosts.deny`

without requiring tcpd to act as an intermediary. In other words, libwrapper-aware applications can use the access controls in `hosts.allow` and `hosts.deny` via system calls provided by libwrappers.

Using iptables for "Local Firewall" Rules While libwrappers and TCP Wrappers are ubiquitous and easy to use, neither is nearly so powerful as the Linux kernel's native firewall mechanism, **netfilter**. Because netfilter is commonly referred to by the name of its user-space front end, **iptables**, we'll use the latter term here.

The iptables command may be used to configure both multi-interface firewall systems that protect large networks, as well as host firewall services on ordinary servers and desktop systems for local protection. Unsurprisingly, the iptables command has a steep learning curve, particularly for users who aren't network engineers. (Entire books, such as [SUEH05], are dedicated to this one command!)

Nearly all Linux distributions, however, now include utilities for automatically generating "personal" (local) firewall rules, especially at installation time. Typically, they prompt the administrator/user for local services that external hosts should be allowed to reach, if any (e.g., HTTP on TCP port 80, HTTPS on TCP port 443, and SSH on TCP port 22), and then generate rules that

- allow incoming requests to those services;
- block all other inbound (externally originating) transactions, and;
- allow all outbound (locally originating) services.

Note the last item: The assumption here is that all outbound network transactions are legitimate. However, this assumption does not hold if the system is compromised by a human attacker or by malware (e.g., a worm). On the one hand, if an attacker achieves root compromise, he or she can reconfigure iptables anyhow; on the other hand, if an attacker doesn't quite make it to root, then granular "egress rules" (allowing only selected outbound transactions) can at least limit the attacker's ability to connect back to his or her home system, scan and attack other systems, and engage in other potentially harmful network activity.

In cases in which this level of caution is justified, it may be necessary to create more complex iptables policies than your Linux installer's firewall wizard can provide. Some people manually create their own startup script for this purpose (an iptables "policy" is actually just a list of iptables commands), but a tool such as Shorewall or Firewall Builder may instead be used.

Antivirus Software

Historically, Linux hasn't been nearly so vulnerable to viruses as other operating systems (e.g., Windows). This may be due less to Linux's being inherently more secure than to its lesser popularity as a desktop platform: Virus writers wanting to maximize the return on their efforts prefer to target Windows because of its ubiquity.

To some extent, then, Linux users have tended not to worry about viruses. To the degree that they have, most Linux system administrators have tended to rely on keeping up to date with security patches for protection against malware, which is arguably a more proactive technique than relying on signature-based antivirus tools.

And indeed, prompt patching of security holes is an effective protection against worms, which have historically been a much bigger threat against Linux systems than viruses. A worm is simply an automated network attack that exploits one or more specific application vulnerabilities. If those vulnerabilities are patched, the worm won't infect the system.

Viruses, however, typically abuse the privileges of whatever user unwittingly executes them. Rather than actually exploiting a software vulnerability, the virus simply runs as the user. This may not have system-wide ramifications so long as that user isn't root, but even relatively unprivileged users can execute network client applications, create large files that could fill a disk volume, and perform any number of other problematic actions.

Unfortunately, there's no security patch to prevent users from double-clicking on e-mail attachments or loading hostile webpages. Furthermore, as Linux's popularity continues to grow, especially as a general-purpose desktop platform (versus its currently-prevalent role as a back-end server platform), we can expect Linux viruses to become much more common. Sooner or later, therefore, antivirus software will become much more important on Linux systems than it is presently. Nowadays, it's far more common for antivirus software on Linux systems to be used to scan FTP archives, mail queues, etc., for viruses that target *other* systems than to be used to protect the system the antivirus software actually runs on.

There are a variety of commercial and free antivirus software packages that run on (and protect) Linux, including products from McAfee, Symantec, and Sophos; and the free, open-source tool ClamAV.

User Management

As you'll recall from Sections 25.2 and 25.3, the guiding principles in Linux user account security are as follows:

- Be very careful when setting file and directory permissions.
- Use group memberships to differentiate between different roles on your system.
- Be extremely careful in granting and using root privileges.

Let's discuss some of the nuts and bolts of user and group account management, and delegation of root privileges. First, let's look at some commands.

You'll recall that in Section 25.3, we used the **chmod** command to set and change permissions for objects belonging to existing users and groups. To create, modify, and delete user accounts, use the **useradd**, **usermod**, and **userdel** commands, respectively. To create, modify, and delete group accounts, use the **groupadd**, **groupmod**, and **groupdel** commands, respectively. Alternatively, you can simply edit the file `/etc/passwd` directly to create, modify, or delete users, or edit `/etc/group` to create, modify, or delete groups.

Note that initial (primary) group memberships are set in each user's entry in `/etc/passwd`; supplementary (secondary) group memberships are set in `/etc/group`. (You can use the usermod command to change either primary or supplementary group memberships for any user.) To change your password, use the **passwd** command. If you're logged on as root, you can also use this command to change other users' passwords.

Password Aging

Password aging (i.e., maximum and minimum lifetime for user passwords) is set globally in the files `/etc/login.defs` and `/etc/default/useradd`, but these settings are only applied when new user accounts are created. To modify the password lifetime for an existing account, use the **chage** command.

As for the actual minimum and maximum password ages, passwords should have some minimum age to prevent users from rapidly "cycling through" password changes in attempts to reuse old passwords; seven days is a reasonable minimum password lifetime. Maximum lifetime is trickier: If this is too long, the odds of passwords being exposed before being changed will increase, but if it's too short, users frustrated with having to change their passwords frequently may feel justified in selecting easily guessed but also easily remembered passwords, writing passwords down, and otherwise mistreating their passwords in the name of convenience. Some value between two and six months is a reasonable balance for many organizations.

In any event, it's much better to disable or delete defunct user accounts promptly, and to educate users on protecting their passwords than it is to rely too much on password aging.

"Root Delegation:" su and sudo As we've seen, the fundamental problem with Linux and UNIX security is that far too often, permissions and authority on a given system boil down to "root can to anything, users can't do much of anything." Provided you know the root password, you can use the `su` command to promote yourself to root from whatever user you logged in as. Thus, the `su` command is as much a part of this problem as it is part of the solution.

Sadly, it's much easier to do a quick su to become root for a while than it is to create a granular system of group memberships and permissions that allows administrators and sub-administrators to have exactly the permissions they need. You can use the `su` command with the "-c" flag, which allows you to specify a single command to run as root rather than an entire shell session (e.g., "`su -c rm somefile.txt`"), but because this requires you to enter the root password, everyone who needs to run a particular root command via this method will need to be given the root password. But it's never good for more than a small number of people to know root's password.

Another approach to solving the "root takes all" problem is to use SELinux's Role-Based Access Controls (RBAC) (see Section 25.7), which enforce access controls that reduce root's effective authority. this is much more complicated than setting up effective groups and group permissions. (However, adding that degree of complexity may be perfectly appropriate, depending on what's at stake.)

A reasonable middle ground is to use the **sudo** command, which is a standard package on most Linux distributions. "sudo" is short for "superuser do", and it allows users to execute specified commands as root without actually needing to know the root password (unlike su). `sudo` is configured via the file `/etc/sudoers`, but you shouldn't edit this file directly; rather, you should use the command `visudo`, which opens a special `vi` (text editor) session.

As handy as it is, sudo is a very powerful tool, so use it wisely: Root privileges are never to be trifled with. It really is better to use user and group permissions judiciously than to hand out root privileges even via sudo, and it's better still to use an RBAC-based system like SELinux if feasible.

Logging

Logging isn't a proactive control; even if you use an automated "log watcher" to parse logs in real time for security events, logs can only tell you about bad things that have already happened. But effective logging helps ensure that in the event of a system breach or failure, system administrators can more quickly and accurately identify what happened and thus most effectively focus their remediation and recovery efforts.

On Linux systems, system logs are handled either by the ubiquitous **Berkeley Syslog daemon** (syslogd) in conjunction with the **kernel log daemon** (klogd), or by the much-more-feature-rich **Syslog-NG**. System log daemons receive log data from a variety of sources (the kernel via `/proc/kmsg`, named pipes such as `/dev/log`, or the network), sort by **facility** (category) and **severity**, then write the log messages to log files (or to named pipes, the network, etc.). Figure 25.3 lists the facilities and severities, both in their mnemonic and numeric forms, of Linux logging facilities, plus syslogd's actions (log targets).

Syslog-NG, the creation of Hungarian developer Balazs Scheidler, is preferable to syslogd for two reasons. First, it can use a much wider variety of log-data sources and destinations. Second, its "rules engine" (usually configured in `/etc/syslog-ng/syslog-ng.conf`) is much more flexible than syslogd's simple configuration

Facilities	Facility Codes†	Priorities (in increasing order)	Priority Codes†	Actions
auth	4	none	n/a	/some/file (log to specified file)
auth-priv	10	debug	7	-/some/file (log to spec'd file but don't sync afterwards)
cron	9	info	6	/some/pipe (log to specified pipe)
daemon	3	notice	5	dev/some/tty_or_console (log to specified console)
kern	0	warning	4	@remote.hostname.or.IP (log to specified remote host)
lpr	6	err	3	username1, username2, etc (log to these users' screens)
mail	2	crit	2	* (log to all users' screens)
mark	n/a	alert	1	
news	7	emerg	0	
syslog	5	* ("any priority")	n/a	
user	1			
uucp	8			
local {0–7}	16–23			
* {"any facility"}	n/a			

Usage of ! and = as prefixes with priorities	
*.notice (no prefix)	= "any event with priority of notice or higher"
*.!notice	= "no event with priority of notice or higher"
*.=notice	= "only events with priority of notice"
*.!=notice	= "no events with priority of notice"

†Numeric facility codes should not be used under Linux; they're here for reference only, as some other syslogd implementations (e.g., Cisco IOS) do use them

Figure 25.3 Syslogd Reference

file (`/etc/syslogd.conf`), allowing you to create a much more sophisticated set of rules for evaluating and processing log data.

Naturally, both syslogd and Syslog-NG install with default settings for what gets logged, and where. While these default settings are adequate in many cases, you should never take for granted that they are. At the very least, you should decide what combination of local and remote logging to perform. If logs remain local to the system that generates them, they may be tampered with by an attacker. If some or all log data are transmitted over the network to some central log-server, audit trails can be more effectively preserved, but log data may also be exposed to network eavesdroppers. (The risk of eavesdropping is still another reason to use Syslog-NG; whereas syslogd only supports remote logging via the connectionless UDP protocol, Syslog-NG also supports logging via TCP, which can be encrypted via a TLS "wrapper" such as Stunnel or Secure Shell.)

Local log files must be carefully managed. Logging messages from too many different log facilities to a single file may result in a log file that is difficult to cull useful information from; having too many different log files may make it difficult for administrators to remember where to look for a given audit trail. And in all cases, log files must not be allowed to fill disk volumes.

Most Linux distributions address this last problem via the **logrotate** command (typically run as a cron job), which decides how to rotate (archive or delete) system and application log files based both on global settings in the file `/etc/logrotate.conf` and on application-specific settings in the scripts contained in the directory `/etc/logrotate.d/`.

The Linux logging facility provides a local "system infrastructure" for both the kernel and applications, but it's usually also necessary to configure applications themselves to log appropriate levels of information. We will revisit the subject of application-level logging in Section 25.6.

Other System Security Tools

Other tools worth mentioning that can greatly enhance Linux system security include the following:

- **Bastille:** A comprehensive system-hardening utility that educates as it secures.
- **Tripwire:** A utility that maintains a database of characteristics of crucial system files and reports all changes made to them.
- **Snort:** A powerful free Intrusion Detection System (IDS) that detects common network-based attacks.
- **Nessus:** A modular security scanner that probes for common system and application vulnerabilities.

25.6 APPLICATION SECURITY

Application security is a large topic; entire chapters in [BAUE05] are devoted to securing particular applications. However, many security features are implemented in similar ways across different applications. In this brief but important section, we'll examine some of these common features.

Running as an Unprivileged User/Group

Remember that in Linux and other UNIX-like operating systems, every process runs as some user. For network daemons in particular, it's extremely important that this user not be root; any process running as root is never more than a single buffer overflow or race condition away from being a means for attackers to achieve remote root compromise. Therefore, one of the most important security features a daemon can have is the ability to run as a nonprivileged user or group.

Running network processes as root isn't entirely avoidable; for example, only root can bind processes to "privileged ports" (TCP and UDP ports lower than 1024). However, it's still possible for a service's *parent* process to run as root in order to bind to a privileged port, but to then spawn a new child process that runs as an unprivileged user, each time an incoming connection is made.

Ideally, the unprivileged users and groups used by a given network daemon should be dedicated for that purpose, if for no other reason than for auditability (i.e., if entries start appearing in `/var/log/messages` indicating failed attempts by the user *ftpuser* to run the command `/sbin/halt`, it will be much easier to determine precisely what's going on if the *ftpuser* account isn't shared by five different network applications).

Running in a Chroot Jail

If an FTP daemon serves files from a particular directory, say, `/srv/ftp/public`, there shouldn't be any reason for that daemon to have access to the rest of the file system. The **chroot** system call confines a process to some subset of `/`, that is, it maps a virtual "`/`" to some other directory (e.g., `/srv/ftp/public`). We call this directory to which we restrict the daemon a **chroot jail**. To the "chrooted" daemon, everything in the chroot jail appears to actually be in `/` (e.g., the "real" directory `/srv/ftp/public/etc/myconfigfile` appears as `/etc/myconfigfile` in the chroot jail). Things in directories outside the chroot jail (e.g., `/srv/www` or `/etc`) aren't visible or reachable at all.

Chrooting therefore helps contain the effects of a given daemon's being compromised or hijacked. The main disadvantage of this method is added complexity: Certain files, directories, and special files typically must be copied into the chroot jail, and determining just what needs to go into the jail for the daemon to work properly can be tricky, though detailed procedures for chrooting many different Linux applications are easy to find on the World Wide Web.

Troubleshooting a chrooted application can also be difficult: Even if an application explicitly supports this feature, it may behave in unexpected ways when run chrooted. Note also if the chrooted process runs as root, it can "break out" of the chroot jail with little difficulty. Still, the advantages usually far outweigh the disadvantages of chrooting network services.

Modularity

If an application runs in the form of a single, large, multipurpose process, it may be more difficult to run it as an unprivileged user; it may be harder to locate and fix security bugs in its source code (depending on how well documented and structured the code is); and it may be harder to disable unnecessary areas of functionality. In modern network service applications, therefore, **modularity** is a highly prized feature.

Postfix, for example, consists of a suite of daemons and commands, each dedicated to a different mail-transfer-related task. Only a couple of these processes ever run as root, and they practically never run all at the same time. Postfix, therefore, has a much smaller **attack surface** than the monolithic Sendmail. The popular Web server Apache used to be monolithic, but it now supports code modules that can be loaded at startup time as needed; this both reduces Apache's memory footprint and reduces the threat posed by vulnerabilities in unused functionality areas.

Encryption

Sending logon credentials or application data over networks in clear text (i.e., unencrypted) exposes them to network eavesdropping attacks. Most Linux network applications therefore support encryption nowadays, most commonly via the OpenSSL library. Using application-level encryption is, in fact, the most effective way to ensure end-to-end encryption of network transactions.

The SSL and TLS protocols provided by OpenSSL require the use of **X.509 digital certificates**, that we discuss in Chapter 23.2. These can be generated and signed by the user-space **openssl** command. For optimal security, either a local or commercial (third-party) **Certificate Authority** (CA) should be used to sign all server certificates, but **self-signed** (i.e., non-verifiable) certificates may also be used. [BAUE05] provides detailed instructions on how to create and use your own Certificate Authority with OpenSSL.

Logging

Most applications can be configured to log to whatever level of detail you want, ranging from "debugging" (maximum detail) to "none." Some middle setting is usually the best choice, but you should not assume that the default setting is adequate.

In addition, many applications allow you to specify either a dedicated file to write application event data to, or a syslog **facility** to use when writing log data to `/dev/log` (see Section 25.5). If you wish to handle system logs in a consistent, centralized manner, it's usually preferable for applications to send their log data to `/dev/log`. Note, however, that logrotate (also discussed in Section 25.5) can be configured to rotate *any* logs on the system, whether written by syslogd, Syslog-NG, or individual applications.

25.7 MANDATORY ACCESS CONTROLS

Linux (like most other general-purpose operating systems) uses a DAC security model, in which the owner of a given system object can set whatever access permissions on that resource he or she likes. Stringent security controls, in general, are optional.

In contrast, a computer with Mandatory Access Controls (MAC) has a global security policy that all users of the system are subject to. A user who creates a file on a MAC system generally may not set access controls on that file that are weaker than the controls dictated by the system security policy.

Compromising a system using a DAC-based security model is generally a simple matter of hijacking some process on that system that runs with root/Administrator privileges. On a MAC-based system, however, the only thing the superuser account

is used for is maintaining the global security policy. Day-to-day system administration is performed using accounts that lack the authority to change the global security policy. As a result, it's impossible to compromise the entire system by attacking any one process. (Attacks on the policy-setting account are still possible, however; for example, by booting the system into single-user mode from its physical console.)

Unfortunately, while MAC schemes have been available on various platforms over the years, they have traditionally been much more complicated to configure and maintain than DAC-based operating systems. To create an effective global security policy requires detailed knowledge of the precise (intended) behavior of every application on the system. Furthermore, the more restrictive the security controls are on a given system, the less convenient that system becomes for its users to use.

Linux packagers Novell and Red Hat have addressed MAC complexity in similar ways. Novell's SuSE Linux includes AppArmor, a partial MAC implementation that restricts specific processes but leaves everything else subject to the conventional Linux DAC. In Fedora and Red Hat Enterprise Linux, SELinux has been implemented with a policy that, like AppArmor, restricts key network daemons, but relies on the Linux DAC to secure everything else.

What about high-sensitivity, high-security, multiuser scenarios? In those cases a "pure" SELinux implementation may be deployed, in which *all* processes, system resources, and data are regulated by comprehensive, granular access controls.

Let's take a closer look at SELinux and Novell AppArmor.

SELinux

SELinux is the NSA's powerful implementation of Mandatory Access Controls for Linux. This power, however, comes at a cost: It is a complicated technology, and can be time-consuming to configure and troubleshoot. In this section, we'll discuss SELinux concepts and security models, ending with some pointers to more detailed information on managing SELinux.

The Problem As noted earlier, Linux security often seems to boil down to a cycle of researchers and attackers discovering new security vulnerabilities in Linux applications and kernels; vendors and developers scrambling to release patches, with attackers wreaking havoc against unpatched systems in the meantime; and hapless system administrators finally applying that week's or month's patches, only to repeat the entire trail of tears soon afterward. Unfortunately, there will always be zero-day (as-yet-unpatched) vulnerabilities. SELinux is a mandatory access control implementation that doesn't prevent zero-day attacks, but it's specifically designed to contain their effects.

For example, suppose we have a daemon called blinkled that is running as the user someguy, and this daemon is hijacked by an attacker. blinkled's sole function is to make a keyboard LED blink out jokes in Morse code, so you might think, well, the worst the attacker can do is blink some sort of insult, right? Wrong. The attacker can do anything the someguy account can do, which might include everything from executing the BASH shell to mounting CD-ROMs.

Under SELinux, however, the blinkled process would run in a narrowly defined domain of activity that would allow it to do its job (blinking the LED, possibly reading jokes from a particular text file, etc.). In other words, blinkled's privileges would

not be determined based on its user/owner; rather, they would be determined by much more narrow criteria. Provided blinkled's domain was sufficiently strictly defined, even a successful attack against the blinkled process would, at worst, result in naughty Morse code blinking.

That, in a nutshell, is the problem SELinux was designed to solve.

What SELinux Does By now you should understand how Linux's Discretionary Access Controls work. Even under SELinux, the Linux DACs still apply: If the ordinary Linux permissions on a given file block a particular action (e.g., user A attempting to write file B), that action will still be blocked, and SELinux won't bother evaluating that action. But if the ordinary Linux permissions allow the action, SELinux will evaluate the action against its own security policies before allowing it to occur.

So how does SELinux do this? The starting point for SELinux seems similar to the DAC paradigm: It evaluates actions attempted by **subjects** against **objects**.

In SELinux, "subjects" are always processes. This may seem counterintuitive: aren't subjects sometimes end users? Not exactly: users execute commands (processes). SELinux naturally pays close attention to who or what executes a given process, but the process itself, not the human being who executed it, is considered to be the subject.

In SELinux, we call actions "permissions," just like we do in the Linux DAC. The objects that are acted on, however, are different. Whereas in the Linux DAC model objects are always files or directories, SELinux objects include not only files and directories but also other processes and various system resources in both kernel space and userland.

SELinux differentiates among a wide variety of object "classes" (categories)—dozens, in fact. You can read the complete list in the document "An Overview of Object Classes and Permissions," in the Premium Content website for this book. Not surprisingly, "file" is the most commonly used object class. Other important object are word classes include the following:

- dir
- socket
- tcp_socket
- unix_stream_socket
- file system
- node
- xserver
- cursor

Each object class has a particular set of possible permissions (actions). This makes sense; there are things you can do to directories, for example, that simply don't apply to, say, X Servers. Each object class may have both "inherited" permissions that are common to other classes (e.g., "read"), plus "unique" permissions that apply only to it. Just a few of the unique permissions associated with the "dir" class are as follows:

- search
- rmdir

- getattr
- remove_name
- reparent

These class names or actions are not explained here. Because you don't need to understand them for their own sake, it is sufficient to know that SELinux goes much, much further than Linux DAC's simple model of users, groups, files, directories, and read/write/execute permissions.

As you might guess, SELinux would be impossible to use if you had to create an individual rule for every possible action by every possible subject against every possible object. SELinux gets around this in two ways: (1) by taking the stance "that which is not expressly permitted is denied," and (2) by grouping subjects, permissions, and objects in various ways. Both of these points have positive and negative ramifications.

The "default deny" stance allows you to only have to create rules/policies that describe the behaviors you expect and want, instead of all possible behaviors. It's also, by far, the most secure design principle any access control technology can have. However, it also requires you to anticipate all possible allowable behavior by (and interaction between) every daemon and command on your system. (This is why the "targeted" SELinux policy in Red Hat Enterprise Linux 4 and Fedora Core 3 actually implements what amounts to a "restrict only these particular services" policy, giving free rein to all processes not explicitly covered in the policy. No, this is not the most secure way to use SELinux, or even the way SELinux was originally designed to be used. But as we'll see, it's a justifiable compromise on general-purpose systems.)

The upside of SELinux's various groupings (roles, types/domains, contexts, etc.) is obviously improved efficiency over having to always specify individual subjects, permissions, and objects. The downside is still more terminology and layers of abstraction.

Security Contexts: Users, Roles, and Domains Every individual subject and object controlled by SELinux is governed by a **security context**, each consisting of a **user**, a **role**, and a **domain** (also called a **type**).

A user is what you'd expect: an individual user, whether human or daemon. However, SELinux maintains its own list of users, separately from the Linux DAC system. In security contexts for subjects, the user label indicates which SELinux user account's privileges the subject (which, again, must be a process) is running. In security contexts for objects, the user label indicates which SELinux user account owns the object.

A role is sort of like a group in the Linux DAC system, in that a role may be assumed by any of a number of preauthorized users, each of whom may be authorized to assume different roles at different times. The difference is that in SELinux, a user may only assume one role at a time, and may only switch roles if and when authorized to do so. The role specified in a security context indicates which role the specified user is operating within for that particular context.

Finally, a domain is sort of like a sandbox: a combination of subjects and objects that may interact with each other. Domains are also called types, and although

domains and types are two different things in the **Flask** security model on which the NSA based SELinux, in SELinux, "domain" and "type" are synonymous.

This model, in which each process (subject) is assigned to a domain, wherein only certain operations are permitted, is called **Type Enforcement** (TE), and it's the heart of SELinux. Type Enforcement also constitutes the bulk of the SELinux implementation in Fedora and Red Hat Enterprise Linux.

There's a bit more to it than that, but before we go into further depth, we present an example scenario to illustrate security contexts.

Suppose we're securing my LED-blinking daemon, blinkled, with SELinux. As you'll recall, it's run with the privileges of the account "someguy," and it reads the messages it blinks from a text file, which we'll call `/home/someguy/messages.txt`.

Under SELinux, we'll need an SELinux user called "someguy" (remember, this is in addition to the underlying Linux DAC's "someguy" account, that is, the one in `/etc/passwd`). We'll also need a role for someguy to assume in this context; we could call it "blink_r" (by convention, SELinux role names end with "_r").

The heart of blinkled's security context will be its domain, which we'll call "blinkled_t" (by convention, SELinux domain names end with "_t" — "t" is short for "type"). blinkled_t will specify rules that allow the blinkled process to read the file `/home/someguy/messages.txt` then write data to, say, `/dev/numlockled`.

The file `/home/someguy/messages.txt` and the special file `/dev/numlockled` will need security contexts of their own. Both of these contexts can probably use the blinkled_t domain, but because they describe objects, not subjects, they'll specify the catch-all role "object_r." Objects, which by definition are passive in nature (stuff gets done to them, not the other way around), generally don't assume meaningful roles, but every security context must include a role.

Decision-Making in SELinux There are two types of decisions SELinux must make concerning subjects, domains, and objects: **access** decisions and **transition** decisions. Access decisions involve subjects doing things to objects that already exist, or creating new things that remain in the expected domain. Access decisions are easy to understand; in our example, "may blinkled read `/home/someguy/messages.txt`?" is just such a decision.

Transition decisions, however, are a bit more subtle. They involve the invocation of processes in different domains that the one in which the subject process is running; or the creation of objects in different types than their parent directories. (Note: Even though "domain" and "type" are synonymous in SELinux, by convention we usually use "domain" when talking about processes, and "type" with files.)

That is, normally, if one process executes another, the second process will by default run within the same SELinux domain. If, for example, blinkled spawns a child process, the child process will run in the blinkled_t domain, the same as its parent. If, however, blinkled tries to spawn a process into some other domain, SELinux will need to make a domain transition decision to determine whether to allow this. Like everything else, transitions must be explicitly authorized in the SELinux policy. This is an important check against privilege-escalation attacks.

File transitions work in a similar way: If a subject creates a file in some directory (and if this file creation is allowed in the subject's domain), the new file will normally inherit the security context (user, role, and domain) of the parent directory.

For example, if blinkend's security context allows it to write a new file in `/home/someguy/`, say, `/home/someguy/error.log`, then `error.log` will inherit the security context (user, role, and type) of `/home/someguy/`. If, for some reason, blinkend tries to label `error.log` with a different security context, SELinux will need to make a type transition decision.

Transition decisions are necessary because the same file or resource may be used in multiple domains/types; process and file transitions are a normal part of system operation. But if domains can be changed arbitrarily, attackers will have a much easier time doing mischief.

Role-Based Access Control Besides Type Enforcement, SELinux includes a second model, called **Role-Based Access Control** (RBAC). RBAC builds on the concepts we've already discussed, providing controls especially useful where real human users, as opposed to daemons and other automated processes, are concerned.

RBAC is relatively straightforward. To paraphrase [MCCA05], SELinux rules specify what **roles** each user may assume; other rules specify under what circumstances each user may **transition** from one authorized role to another (unlike groups in the Linux DAC, in RBAC one user may not assume more than one role at a time); and still other rules specify in which **domains** each authorized role may operate.

Multilevel Security The third security model implemented in SELinux is **Multilevel Security** (MLS), which is based on the **Bell-LaPadula** (BLP) model. Chapter 27 describes the BLP model in detail. In SELinux, MLS is enforced via file system labeling.

Managing SELinux Policies Unfortunately, creating and maintaining SELinux policies is complicated and time-consuming; a single SELinux policy may consist of hundreds of lines of text. In Red Hat and Fedora, this complexity is mitigated by the inclusion of a default "targeted" policy that defines types for selected network applications but that allows everything else to run with only Linux DAC controls. You can use RHEL and Fedora's **system-config-securitylevel** GUI to configure the targeted policy.

SELinux policies take the form of various, lengthy text files in `/etc/security/selinux`. SELinux commands common to all SELinux implementations (besides RHEL and Fedora) are **chcon**, **checkpolicy**, **getenforce**, **newrole**, **run_init**, **setenforce**, and **setfiles**. Tresys (`http://www.tresys.com`), however, maintains a suite of free, mainly GUI-based, SELinux tools that are a bit easier to use, including **SePCuT**, **SeUser**, **Apol**, and **SeAudit**.

For more information on using RHEL's SELinux implementation, see [COKE05]. See [MCCA05] for more information on creating and maintaining custom SELinux policies.

Novell AppArmor

AppArmor, Novell's MAC implementation for SuSE, represents a major step forward in making MAC technology a feasible option for system administrators who

want strong security controls but don't have the time or patience to configure and maintain SELinux. As of this writing, AppArmor is only available for SuSE Linux and SuSE Linux Enterprise. AppArmor, like SELinux, is built on top of the Linux Security Modules.

As we've seen, SELinux implements three different types of MAC: Type Enforcement, Role-Based Access Controls, and Multilevel Security. In contrast, Novell AppArmor has a more modest objective: to restrict the behavior of selected applications in a very granular but targeted way. In focusing on applications (at the expense of roles and data classification), AppArmor is built on the assumption that the single biggest attack vector on most systems is application vulnerabilities. If the application's behavior is restricted, then the behavior of any attacker who succeeds in exploiting some vulnerability in that application will also be restricted.

For example, suppose you're running a Web application that runs as user "nobody" and uses user input to update a local text file. On a typical system, if an attacker compromised that Web application (e.g., by sending unexpected input) the attacker might succeed in gaining a remote shell with the privileges of "nobody." If that Web application were protected by AppArmor, however, all the attacker would be able to do would be to alter that single text file; it would neither be possible for the attacker to spawn a remote shell (an unexpected action) nor to read or write any other files.

Comprehensive? By no means: for non-AppArmor-protected applications, the usual (limited) user/group permissions still apply. Normally, only a subset of applications on the system even have AppArmor profiles, and AppArmor provides no controls addressing data classification. To use SELinux terminology, AppArmor provides only nonglobal Type Enforcement, no Role-Based Access Controls, and no Multilevel Security.

For the most part, root is still root, and if you use root access in a sloppy or risky fashion, AppArmor generally won't protect you from yourself. But if an AppArmor protected application runs as root and somehow, becomes compromised that application's access will be contained, root privileges notwithstanding, because those privileges are trumped by the AppArmor policy (which is enforced at the kernel level, courtesy of Linux Security Modules).

AppArmor is, therefore, only a partial implementation of Mandatory Access Controls. But on networked systems, application security is arguably the single most important area of concern, and that's what AppArmor zeroes in on. What's more, AppArmor provides application security via an easy to use graphical user interface that is fully integrated with SuSE's system administration tool, YaST.

We are stopping well short of suggesting that AppArmor is interchangeable with SELinux. If, for example, you run Linux in a true multiuser environment (in which users have shell accounts) or use a Linux system to process highly sensitive data, there really is no substitute for the comprehensive layers of access controls in SELinux.

25.8 REFERENCES

BAUE05 Bauer, M. *Linux Server Security,* Second Edition. Sebastopol, CA: O'Reilly Media, 2005.

COKE05 Coker, F., and Coker, R. "Taking Advantage of SELinux in Red Hat® Enterprise Linux®." *Red Hat Magazine*, April 2005. redhat.com/magazine/006apr05/features/selinux

MCCA05 McCarty, B. *SELinux: NSA's Open Source Security Enhanced Linux*. Sebastopol, CA: O'Reilly Media, 2005.

SUEH05 Suehring, S., and Ziegler, R. *Linux Firewalls.* Upper Saddle River, NJ: Novell Press, 2005.

CHAPTER 26

WINDOWS SECURITY

Windows is the world's most popular operating system, and as such has a number of interesting security-related advantages and challenges. The major advantage is any security advancement made to Windows can protect hundreds of millions of non-technical users, and advances in security technologies can be used by thousands of corporations to secure their assets. The challenges for Microsoft are many, including the fact that security vulnerabilities in Windows can affect millions of users. Of course, there is nothing unique about Windows having security vulnerabilities; all software products have security bugs. However, Windows is used by so many nontechnical users that Microsoft has some interesting engineering challenges.

This chapter begins with a description of the overall security architecture of Windows 2000 and later (see Section 26.1). It is important to point out that versions of Windows based on the Windows 95 code base, including Windows 98, Windows 98 SE, and Windows Me, had no security model, in contrast to the Windows NT code base, on which all current versions of Windows are based. The Windows 9x codebase is no longer supported.

The remainder of the chapter covers the security defenses built into Windows, most notably the security defenses in Windows 2000 and later.

26.1 FUNDAMENTAL WINDOWS SECURITY ARCHITECTURE

Anyone who wants to understand Windows security must have knowledge of the basic fundamental security blocks in the operating system. There are many important components in Windows that make up the fundamental security infrastructure, among them are the following:

- The Security Reference Monitor (SRM)
- The Local Security Authority (LSA)
- The Security Account Manager (SAM)
- Active Directory (AD)
- Authentication Packages
- WinLogon and NetLogon

Let's look at each in detail.

The Security Reference Monitor

This kernel-mode component performs access checks, generates audit log entries, and manipulates user rights, also called privileges. Ultimately, every permission check is performed by the SRM. Most modern operating systems include SRM type functionality that performs privileged permission checks. SRMs tend to be small in size so their correctness can be verified because no one needs a bypassable SRM!

The Local Security Authority

The LSA resides in a user-mode process named `lsass.exe` and is responsible for enforcing local security policy in Windows. It also issues security tokens to accounts as they log on to the system. Security policy includes:

- Password policy, such as complexity rules and expiration times.

- Auditing policy, specifying which operations on what objects to audit.
- Privilege settings, specifying which accounts on a computer can perform privileged operations.

The Security Account Manager

The SAM is a database that stores accounts data and relevant security information about local principals and local groups. Note the term *local*. Windows has the notion of local and domain accounts. We will explain more about this later, but for now, note that Windows users can log on to a computer using either accounts that are known only on that particular computer or accounts that are managed centrally. When a user logs on to a computer using a local account, the SAM process (SamSrv) takes the logon information and performs a lookup against the SAM database, which resides in the `\Windows\System32\Config` directory. If you're familiar with UNIX, think `/etc/passwd` (or similar). If the credentials match, then the user can log on to the system, assuming there are no other factors preventing logon, such as logon time restrictions or privilege issues, which we discuss later in this chapter. Note the SAM does not perform the logon; that is the job of the LSA. The SAM file is binary rather than text, and passwords are stored using the MD4 hash algorithm. On Windows Vista and later, the SAM stores password information using a password-based key derivation function (PBKCS), which is substantially more robust against password guessing attacks than MD4.

Note WinLogon handles local logons at the keyboard, and NetLogon handles logons across the network.

Active Directory

Active Directory (AD) is Microsoft's LDAP directory included with Windows Server 2000 and later. All currently supported client versions of Windows, including Windows 7, 8 and 10, can communicate with AD to perform security operations including account logon. A Windows client will authenticate using AD when the user logs on to the computer using a domain account rather than a local account. Like the SAM scenario, the user's credential information is sent securely across the network, verified by AD, and then, if the information is correct, the user can log on at the computer. Note we say "credential" and not "password" because a credential might take some other form, such as a public and private key pair bound to an X.509 certificate on a smart card. This is why most corporate laptops include smartcard readers.

Local versus Domain Accounts We used the terms *local* and *domain*. A networked Windows computer can be in one of two configurations: either domain joined or in a workgroup. When a computer is domain joined, users can gain access to that computer using domain accounts, which are centrally managed in Active Directory. They can, if they wish, also log on using local accounts, but local accounts may not have access to domain resources such as networked printers, Web servers, and e-mail servers. When a computer is in a workgroup, only local accounts can be used, held in the SAM. There are pros and cons to each scenario. A domain has the major advantage of being centrally managed and as such is much more secure. If an environment has 1000 Windows computers and an employee leaves, the user's account can be

disabled centrally rather than on 1000 individual computers. Security policies, such as which applications are allowed to run, or who can debug applications, are also centrally managed when using AD. This is not only more secure, it also saves time and effort as the number of ancillary computers rises.

The only advantage of using local accounts is that a computer does not need the infrastructure required to support a domain using AD.

As mentioned, Windows has the notion of a workgroup, which is simply a collection of computers connected to one another using a network; but rather than using a central database of accounts in AD, the machines use only local accounts. The difference between a workgroup and a domain is simply where accounts are authenticated. A workgroup has no domain controllers; authentication is performed on each computer, and a domain authenticates accounts at domain controllers running AD.

Using PowerShell for Security Administration Windows 7 and Windows Server 2008 and later include a flexible scripting language named PowerShell. PowerShell provides rich access to Windows computers, and that includes access to security settings. Using PowerShell it is possible to create tailored management tools for your organization. Throughout this chapter, we will give examples of using PowerShell to investigate or manipulate security-related details. In some cases, it might be necessary to run an elevated PowerShell instance, one that runs as a privileged account, such as a domain or local administrator.

If you are new to PowerShell, there are three core things you need to know. They are the following:

1. PowerShell is based on .NET. If you can do it in C# or VB.NET, you can do it in a PowerShell environment.
2. Commands in PowerShell are called cmdlets, and have a consistent verb-noun syntax.
3. Like all scripting environments, PowerShell supports piping output from one command to another. But unlike other scripting environments, PowerShell pipes objects and not text. This allows for very rich data processing, filtering, and analysis. For example, the following pipes Process objects from get-process to format-table:

   ```
   Get-Process | Format-Table
   ```

   Or, you can stop all running Google Chrome (chrome.exe) processes by running:

   ```
   Get-Process-name chrome | Stop-Process
   ```

   This only works because Process objects, one for each Chrome instance, are sent to a cmdlet that calls the Stop method on a Process object.

   You can get a list of object methods and properties by piping to the Get-Member cmdlet. For example, the following displays all the methods and properties associated with objects representing Windows:

   ```
   Get-Service | Get-Member.
   ```

   For more information about PowerShell, refer to `https://technet.microsoft.com/en-us/library/bb978526.aspx`.

Windows Security Basics—An End-to-End Domain Example

Now that you know the basic elements that make up the core Windows security infrastructure, we will give an example of what happens when a user logs on to a Windows system.

Before a user can log on to a Windows network, a domain administrator must add the user's account information to the system; this will include the user's name, account name (which must be unique within the domain), and password. Optionally, the administrator can grant group membership and privileges.

After the administrator has entered the user's account information, Windows creates an account for the user in the domain controller running AD. Each user account is uniquely represented by a Security ID (SID). SIDs are unique within a domain, and every account gets a different SID. This is an important point. If you create an account named Blake, delete the account, and "re-create" the account named Blake, they are in fact two totally different accounts because they will have different SIDs.

A user account's SID is of the following form:

- `S-1-5-21-AAA-BBB-CCC-RRR`.
- S simple means SID.
- 1 is the SID version number.
- 5 is the identifier authority; in this example, 5 is `SECURITY_NT_AUTHORITY`.
- 21 means "not unique," which just means there is no guarantee of uniqueness; however, a SID is unique within a domain, as you will see in a moment.
- `AAA-BBB-CCC` is a unique number representing the domain.
- `RRR` is called a relative ID (RID); it is a number that increase by 1 as each new account is created. RIDs are never repeated; this is what makes each SID unique.

For example, a SID might look like this:

```
S-1-5-21-123625317-425641126-188346712-2895
```

In Windows, a username can be in one of two formats. The first, named the SAM format, is supported by all versions of Windows and is of the form `DOMAIN\Username`. The second is called User Principal Name (UPN) and looks more like an RFC822 e-mail address: `username@domain.company.com`. The SAM name should be considered a legacy format.

If the user enters just a username, then the domain in which the machine resides is pre-pended to the user name. So if Blake's PC is in the Development domain, and he enters "Blake" as his logon account, he is actually logging on using Development\Blake if SAM accounts are used, or Blake@Development.Company.com if UPN names are used.

When a user logs on to Windows, he or she does so using either a username and password, or a username and a smart card. It is possible to use other authentication or identification mechanisms, such as an RSA SecureID token or biometric device, but these require third-party support.

Assuming the user logs on correctly, a Kerberos authentication token is generated by the operating system and assigned to the user, as we discuss in Chapter 23.1. A token contains the user's SID, group membership information, and privileges. Groups are also represented using SIDs. We explain privileges subsequently. The user's token is assigned to every process run by the user. It is used to perform access checks discussed subsequently.

Windows Security Basics—An End-to-End Workgroup Example

You will notice that this section is much smaller than the domain-joined scenario, because the process is much simpler.

When a user logs on to a computer using a local account, the computer must have a user account and an optional password associated with the account.

Let's say Paige has an account, and the SID for that account is:

```
S-1-5-21-251942251-425652175-1800782563-1238
```

When she enters her username and password, a token is created by the operating system, which includes Paige's SID, SIDs for all the groups of which she is a member, as well as the privileges she holds. Just like in the domain example.

On a domain-joined computer (we will use the "Marketing" domain), it is possible for a user to log on to a local account by using the "." domain. So rather than using "Marketing\Paige" or just "Paige" Paige can use ".\Paige" assuming there is a local Paige account on the computer. The "." will substitute the machine name as the workgroup name.

Important Note about Admin Accounts and Blank passwords A little earlier we used the term "optional password" which means Windows can support the use of user accounts that have no password. Hopefully, your first reaction is "isn't that insecure?" the answer is "of course, it is," but some people in a home environment want to do this. That is why setting a password is actively encouraged during setup, and never applies to domain accounts.

Your next reaction might be, "Well, does that mean I can access a computer remotely and log on using a local admin account and not be prompted for a password?" The answer is emphatically, "NO!" Remote access from one Windows computer to another using an account that is a member of the local Administrators group can only be performed if the account has a password. Access is denied when using a nonpassword admin account remotely.

Using PowerShell, you can dump information about the currently logged on user with this line:

```
[Security.Principal.WindowsIdentity]::GetCurrent()
```

Note this is not using a cmdlet; rather it is calling directly into the .NET Framework.

Privileges in Windows

Privileges are essentially systemwide permissions assigned to user accounts. Examples of Windows privileges include the ability to back up the computer, or the ability to change the system time. Performing a backup is privileged because it

bypasses all access checks so a complete backup can be performed. Likewise, setting the system time is privileged because changing the time can make Kerberos authentication fail and lead to erroneous data being written to the logging system. There are over 45 privileges in Windows 2000. Some privileges are deemed "dangerous," which means a malicious account that is granted such a privilege can cause damage. Examples of such potentially dangerous privileges include the following:

- **Act as part of operating system privilege.** This is often referred to as the Trusted Computing Base (TCB) privilege, because it allows code run by an account that granted this privilege to act as part of the most trusted code in the operating system: the security code. This is the most dangerous privilege in Windows, and is granted only the Local System account; even administrators are not granted this privilege.
- **Debug programs privilege.** This privilege allows an account to debug any process running in Windows. A user account does not need this privilege to debug an application running under the user's account. Because of the nature of debuggers, this privilege basically means a user can run any code he or she wants in any running process.
- **Backup files and directories privilege.** Any process running with this privilege will bypass all access control list (ACL) checks, because the process must be able to read all files to build a complete backup. Its sister privilege Restore files and directories is just as dangerous because it will ignore ACL checks when copying files to source media.

Some privileges are generally deemed benign. An example is the "bypass traverse checking" privilege that is used to traverse directory trees even though the user may not have permissions on the traversed directory. This privilege is assigned to all user accounts by default and is used as an NTFS file system optimization.

Access Control Lists

Windows has two forms of access control list (ACL). The first is called a discretionary access control list (DACL) and is usually what most people mean when they say ACL. A DACL grants or denies access to protected resources in Windows such as files, shared memory, and named pipes. The other kind of ACL is the system access control list (SACL), which is used for auditing to enforce mandatory integrity policy. Let's take a moment to look at the DACL.

Objects that require protection are assigned a DACL (and if possible a SACL), which includes the SID of the object owner (usually the object creator) as well as a list of access control entries (ACEs). Each ACE includes a SID and an access mask. An access mask could include the ability to read, write, create, delete, and modify. Note access masks are object-type specific; for example, services (the Windows equivalent of UNIX daemons) are protected objects and support an access mask to create a service (`SC_MANAGER_CREATE_SERVICE`) and a mask that allows service enumeration (`SC_MANAGER_ENUMERATE_SERVICE`). The data structure that includes the object owner, DACL, and SACL is referred to as the object's security descriptor (SD).

A sample SD with no SACL is as follows:

```
Owner: CORP\Blake
ACE[0]: Allow CORP\Paige Full Control
ACE[1]: Allow Administrators Full Control
ACE[2]: Allow CORP\Cheryl Read, Write, and Delete
```

The DACL in this SD allows the user named Paige (from the CORP domain) full access to the object; she can do anything to this object. Members of the Administrators can do likewise. Cheryl can read, write, and delete the object. Note the object owner is Blake; as the owner, he can do anything to the object as well. This was always the case until the release of Windows Vista. Some customers do not want owners to have such unbridled access to objects, even though they created them. In Windows Vista and later, you can include an Owner SID in the DACL, and the access mask associated with that account applies to the object owner.

There are two important things to keep in mind about access control in Windows. First, if the user accesses an object with the SD example above, and the user is not Blake, not Paige, not Cheryl, and not a member of the Administrator's group, then that user is denied to access the object. There is no implied access. Second, if Cheryl requests read access to the object, she is granted read access. If she requests read and write access, she is also granted access. If she requests create access, she is denied access unless Cheryl is also a member of the Administrators group, because the "Cheryl ACE" does not include the "create" access mask. The last point is critically important. When a Windows application accesses an object, it must request the type of access the application requires. Many developers would simply request "all access" when in fact the application may only want to read the object. If Cheryl uses an application that attempts to access the object described above and the application requests full access to the object she is denied to access the object unless she is an administrator. This is the prime reason why so many applications failed to execute correctly on Windows XP and later, unless the user is a member of the Administrator's group.

We mentioned earlier that a DACL grants or denies access; technically, this is not 100% accurate. Each ACE in the DACL determines access; an ACE can be an allow ACE or a deny ACE. Look at this variant of the previous SD:

```
Owner: CORP\Blake
ACE[0]: Deny Guests Full Control
ACE[1]: Allow CORP\Paige Full Control
ACE[2]: Allow Administrators Full Control
ACE[3]: Allow CORP\Cheryl Read, Write, and Delete
```

Note the first ACE is set to deny members of the guests account full control to the object. Basically, guests are out of luck if they attempt to access the object protected by this SD. Deny ACEs are not often used in Windows because they can be complicated to troubleshoot. Also note the first ACE is the deny ACE; it is important that deny ACEs come before allow ACEs because Windows evaluates each ACE in the ACL until access is granted or explicitly denied. If the ACL grants access, then Windows will stop ACL evaluation, and if the deny ACE is at the end of the ACL,

then it is not evaluated, so the user is granted access even if the account may be denied access. When setting an ACL from the user interface, Windows will always put deny ACEs before allow ACEs, but if you create an ACL programmatically (e.g., by using the *SetSecurityDescriptorDacl* function), you must explicitly place the deny ACEs first.

You can get an object's SD using PowerShell with the following syntax:

```
get-acl c:\folder\file.txt | format-list
```

You can also use the set-acl cmdlet to set an object's DACL or SACL.

In current versions of Windows, it is possible to set and get an SD using the Security Descriptor Definition Language (SDDL). SDDL is simply a text representation of a SD. The *ConvertStringSecurityDescriptorToSecurityDescriptor*() function can be used to convert SDDL text into a binary SD, which can then be assigned to an object.

The authorization framework in Windows also supports "conditional ACEs" which allows application-level access condition to be evaluated when an access check is performed. Examples could include business logic. For example, a conditional ACE to encapsulate the following business rule:

```
User is a Manager in Sales or Marketing
```

As:

```
(Title=="Manager" && (Division=="Sales" || Division==
"Marketing"))
```

Note there is no user interface to define these rules, these can only be set using programmatic access to SDDL.

Access Checks

It is now time to put these all together. When a user account attempts to access a protected object, the operating system performs an access check. It does this by comparing the user account and group information in the user's token and the ACEs in the object's ACL. If all the requested operations (read, write, delete, and so on) are granted, then access is granted; otherwise, the user gets an access-denied error status (error value 5).

Impersonation

There is one last thing you should understand about Windows. Windows is a multithreaded operating system, which means a single process can have more than one thread of execution at a time. This is very common for both server and client applications. For example, a word processor might have one thread accepting user input, and another performing a background spellcheck. A server application, such as a database server, might start a large number of threads to handle concurrent user requests. Let's say the database server process runs as a predefined account named DB_ACCOUNT; when it takes a user request, the application can impersonate the calling user by calling an impersonation function. For example, one networking protocol supported by Windows is called Named Pipes, and the *ImpersonateNamedPipeClient* function

will impersonate the caller. Impersonation means setting the user's token on the current thread. Normally access checks are performed against the process token, but when a thread is impersonating a user, the user's token is assigned to the thread, and the access check for that thread is performed against the token on the thread, not the process token. When the connection is done, the thread "reverts," which means the token is dropped from the thread.

So why impersonate? Imagine if the database server accesses a file named `db.txt`, and the DB_ACCOUNT account has read, write, delete, and update permission on the file. Without impersonation, any user could potentially read, write, delete, and update the file. With impersonation, it is possible to restrict who can do what to the `db.txt` file.

In older versions of Windows, a process listening on a named pipe running as any account could impersonate the connected user. But since the mid-2000s, this was changed to only allowing accounts granted the "Impersonate a client after authentication" privilege to impersonate users. By default, service accounts and administrative accounts have this privilege.

Mandatory Access Control

Windows Vista, Windows Server 2008, and later include an additional authorization technology named Integrity Control, which goes one step beyond DACLs. DACLs allow fine-grained access control, but integrity controls limit operations that might change the state of an object. The general premise behind integrity controls is simple; objects (such as files and processes) and principals (such as users) are labeled with one of the following integrity levels:

- Low integrity (S-1-16-4096)
- Medium integrity (S-1-16-8192)
- High integrity (S-1-16-12288)
- System integrity (S-1-16-16384)

Note the SIDs after the integrity levels. Microsoft implemented integrity levels using SIDs. For example, a high-integrity process will include the S-1-16-12288 SID in the process token. If a subject or object does not include an integrity label, then the subject or object is deemed medium integrity.

The screen shot of Figure 26.1 shows a normal user token in Windows Vista or Windows 7. It includes medium-integrity SID, which means this user account is medium integrity and any process run by this user can write only to objects of medium and lower integrity.

When a write operation occurs, Windows will first checks to see if the subject's integrity level dominates the object's integrity level, which means the subject's integrity level is equal to or above the object's integrity level. If it is, and the normal DACL check succeeds, then the write operation is granted. The most important component in Windows that uses integrity controls is Internet Explorer 7.0 and later. Integrity controls help create a sandbox; the main `iexplore.exe` process that renders and hosts potentially hostile markup and mobile code from the Internet runs at low integrity, but the majority of the operating system is marked medium or higher integrity, which means malicious code inside the browser has a harder time writing to the operating system.

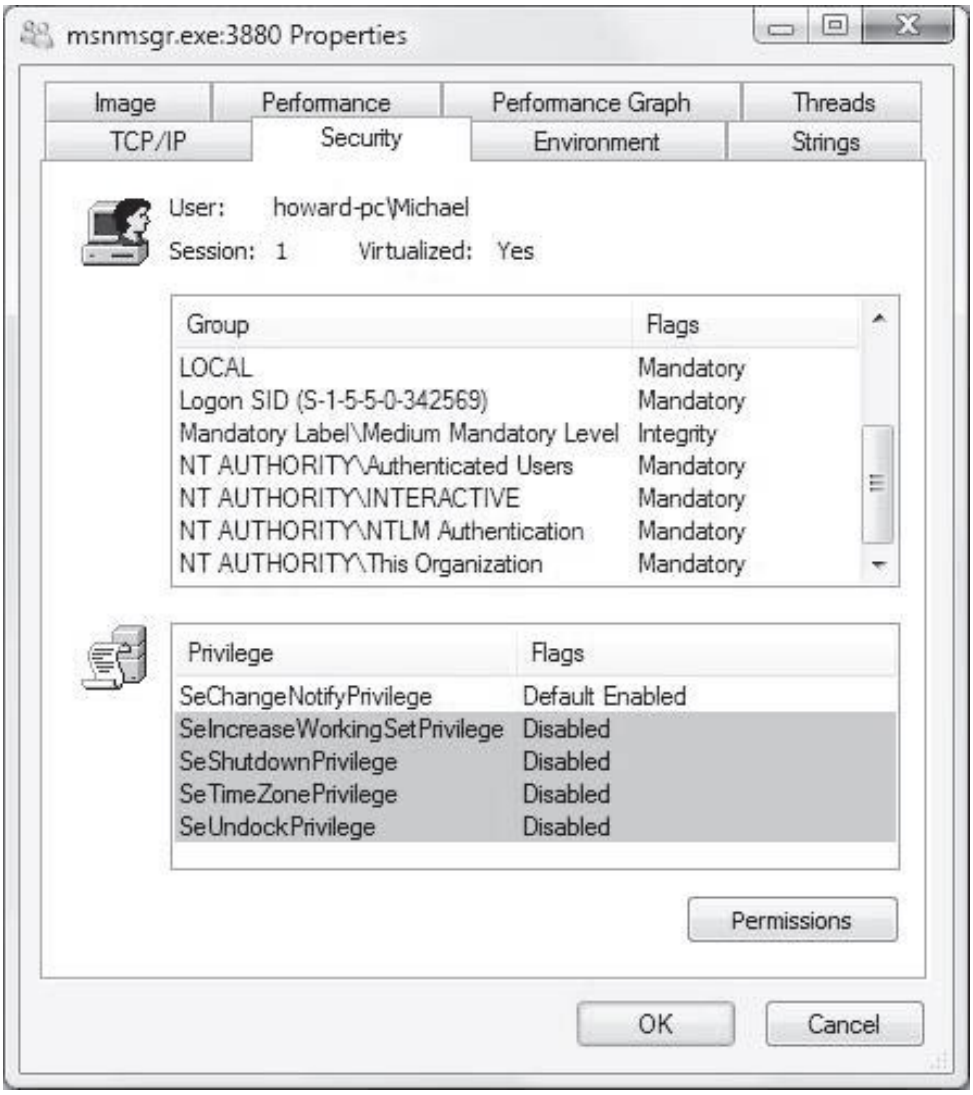


Figure 26.1 Screen Shot of User Account in Windows Vista
Source: From Microsoft® Windows Vista, Microsoft Corporation. Reprinted with permission Microsoft Corporation.

That completes this whirlwind tour of Windows security principles. Now let's shift focus to security defenses within Windows.

26.2 WINDOWS VULNERABILITIES

Windows, like all operating systems, has security bugs, and a number of these bugs have been exploited by attackers to compromise customer operating systems. After 2001, Microsoft decided to change its software development process to better accommodate secure design, coding, testing, and maintenance requirements, with one goal in mind: reduce the number of vulnerabilities in all Microsoft products. This process improvement is called the Security Development Lifecycle [HOWA06]. The core SDL requirements are as follows:

- Mandatory security education
- Secure design requirements

- Threat modeling
- Attack surface analysis and reduction
- Secure coding requirements and tools
- Secure testing requirements and tools
- Security push
- Final security review
- Security response

A full explanation of SDL is beyond the scope of this chapter, but the net effect has been an approximately 50% reduction in security bugs. Windows Vista is the first version of Windows to have undergone SDL from start to finish. Other versions of Windows had a taste of SDL, such as Windows XP SP2, but Windows XP predates the introduction of SDL at Microsoft.

SDL does not equate to "bug free" and the process is certainly not perfect, but there have been some major SDL success stories. Microsoft's Web server, Internet Information Services (IIS), has a much-maligned reputation because of serious bugs found in the product that led to worms, such as CodeRed. IIS version 6, included with Windows Server 2003, has had a stellar security track record since its release; there have been only three reported vulnerabilities in the four years since its release, none of them is critical. And this figure is an order of magnitude less bugs than IIS's main competitor, Apache [HOWA04].

Another example of SDL working is Microsoft's database server, SQL Server. In the same period, there have been less than 10 security vulnerabilities in SQL Server. When compared to SQL Server's major competitor "Unbreakable Oracle," this is a significant engineering feat.

The most visible part of any vendor's security process is patch management, and Microsoft has substantially fine-tuned the security update process over the last few years. At first, Microsoft issued security updates as soon as they were ready, but now Microsoft issues security updates the second Tuesday of each month. This day is now affectionately referred to as "Patch Tuesday." More recently, Microsoft introduced a novel idea; the Thursday before the second Tuesday of each month, Microsoft announces how many security updates will be shipped, for which products, and what the highest severity rating will be. This streamlined security update process gives system administrators to have some much-needed predictability to their busy schedules.

26.3 WINDOWS SECURITY DEFENSES

This section and the next will focus on defenses within Windows. The defenses can be grouped into four broad categories:

1. Account defenses
2. Network defenses
3. Memory Corruption defenses.
4. Browser defenses.

We discuss each in detail, most notably as each relates to Windows Vista and later.

All versions of Windows offer security defenses, but the list of defenses has grown rapidly in the last twenty years to accommodate increased Internet-based threats. The attackers today are not just kids; they are criminals who see money in compromised computers. A zombie network comprised of a few thousand computers under the control of an attacker could be trained on an e-commerce site for a few hours, effectively knocking it off the Internet, losing sales and potential customers. The attack stops when the extortion money is paid. Again, we want to stress that attacks and compromises are very real, and the attackers are highly motivated by money. Attackers are no longer just young, anarchic miscreants; they are real criminals, and in many cases, well-funded nations.

Before we discuss security defenses, we discuss system hardening, which is critical to the defensive posture of a computer system and network.

Windows System Hardening Overview

The process of hardening is shoring up defenses, reducing the amount of functionality exposed to untrusted users, and disabling less-used features. At Microsoft, this process is called Attack Surface Reduction. The concept is simple: Apply the 80/20 rule to features. If the feature is not used by 80% of the population, then the feature should be disabled by default. While this is the goal, it is not always achievable simply because disabling vast amounts of functionality makes the product unusable for nontechnical users, which leads to increased support calls and customer frustration. One of the simplest and effective ways to reduce attack surface is to replace anonymous networking protocols with authenticated networking protocols. The biggest change of this nature in Windows XP SP2 was to change all anonymous remote procedure call (RPC) access to require authentication. This was a direct result of the Blaster worm. Worms spread anonymously, and making this simple change to RPC will help prevent worms that take advantage of vulnerabilities in RPC code, and code that uses RPC. It turns out that, in practice, requiring authentication is a very good defense; the Zotob worm, which took advantage of a vulnerability in Microsoft Plug and Play (PnP) and was accessible through RPC, did not affect Windows XP SP2, even the coding bug was there, because an attacker must be authenticated first. But perhaps the beauty of using authentication to reduce attack surface is that most users don't even know it is there, yet the user is protected.

Another example of hardening Windows occurred in Windows Server 2003. Because Windows Server 2003 is a server and not a client platform, the Web browser Internet Explorer was stripped of all mobile code support by default.

In general, hardening servers is easier than hardening clients for the following reasons:

1. Servers tend to be used for very specific and controlled purposes, while client computers are used for more general purpose.
2. Whether it is true or not, the perception is that server users are administrators and have more computer configuration skills than a typical client computer user.

Account Defenses

As noted earlier, user accounts can contain highly privileged SIDs (such as the Administrators or Account operators groups) and dangerous privileges (such as Act as part of operating system), and malicious software running with these SIDs or privileges can wreak havoc. The principle of least privilege dictates that users should operate with just enough privilege to get the tasks done, and no more. Historically, Windows XP users operated by default as members of the local Administrators group; this was done simply for application compatibility reasons. Many applications that used to run on Windows 95, 98, and Me would not run correctly on Windows XP unless the user was an administrator. In other words, in some cases a Windows XP user running as a "Standard User" could run into some errors. Of course, there is nothing stopping a user from running as a "Standard User."

Windows XP and Windows Server 2003 add a new feature named "Secondary Logon," which allows a user account to right click an application, select "Run as . . . ," then enter another user account and password to run the application. Windows XP and Windows Server 2003 also include support for another way to reduce privilege on a per-thread level, called a restricted token. A restricted token is simply a thread token with privileges removed and/or SIDs marked as deny-only SIDs. You can learn more about restricted tokens and how to use them programmatically or through Windows Policy [HOWA04].

Windows Vista and later change the default; all user accounts are users and not administrators. This is referred to as User Account Control (UAC.)

When a user wants to perform a privileged operation, the user is prompted to enter an administrator's account name and password. If the user is an administrator, the user is prompted to give consent to the operation. This is often referred to as "over the shoulder logon." The reason for doing this is if malware attempts to perform a privileged task, the user is notified. Note in the case of Windows Server 2008 and later, if a user enters a command in the Run dialog box from the Start menu, the command will always run elevated if the user is normally an administrator and will not prompt the user. The great amount of user interaction required to perform these privileged operations mitigates the threat of malware performing tasks off the Run dialog box.

Low Privilege Service Accounts Windows services are long-lived processes that often start right after the computer boots. Examples include the File and Print service and the DNS service. Many such services run with elevated privileges because they perform privileged operations. It is true, however, that many services do not need such elevated requirements, and in Windows XP, Microsoft added two new service accounts: the Local Service account and the Network service account, which allow a service local or network access, respectively, but processes running with these accounts operate at a much lower privilege level. Note that unlike the system account, neither the local service nor the network service accounts are members of the local administrator's group.

In Windows XP SP2, Microsoft made an important change to the remote procedure call service (RPCS) as an outcome of the Blaster worm. In versions of Windows prior to Windows XP SP2, RPCSs ran as the System account, the most privileged account in Windows. For Windows XP SP2, a major architectural change was made;

RPCSs was split in two. The reason RPCSs ran with System identity was simply to allow it to execute Distributed Component Object Model (DCOM, which layered on top of RPC) objects on a remote computer correctly, but raw RPC traffic does not require such elevated privileges. So RPCSs was rearchitected into components, RPCSs shed its DCOM activation code, and a new service was created called the DCOM Server Process Launcher. RPCSs runs as the lower-privilege Network service account; DCOM runs as SYSTEM. This is a good example of the principle of least privilege and separation of privilege in action. Apache, OpenSSH, and Internet Information Services (IIS) 6 and later also use this model. A small amount of code runs with elevated identity, and related components run with lower identity. In the case of Apache on Linux, the initial httpd daemon runs as root because it must open port 80; once the port is open httpd spawns "worker" httpd dameons as lower-privilege accounts such as nobody or Apache. It is these worker processes that receive potentially malicious input. IIS6 follows a similar model, a process named inetinfo starts under the System identity because it must perform administrative tasks, and it starts worker processes named `w3wp.exe` to handle user requests (these processes run under the lower-privilege network service identity).

STRIPPING PRIVILEGES Another useful defense, albeit not often used in Windows, is to strip privileges from an account when the application starts. This should be performed very early in the application startup code (e.g., early in the application's *main* function). The best way to describe this is by way of example. In Windows, the Index server process runs as the system account because it needs administrative access to all disk volumes to determine if any file has changed, it can reindex the file. Only members of the local Administrators group can get a volume handle. This is the sole reason Index server must run as the system account, yet as you will remember, the system account is bristling with dangerous privileges, such as the TCB privilege and backup privilege. So when the main index server process starts (`cidaemon.exe`), it sheds any unneeded privileges as soon as possible. The function that performs this is *AdjustTokenPrivileges*.

Windows Vista and later also add a function to define the set of privileges required by a service to run correctly. The function that performs this is *ChangeServiceConfig2*.

That ends the overview of core-user account-related security defenses and technologies. Now let's switch our focus to network defenses.

Network Defenses

There is one big problem with defenses that focus on the user and user accounts: They do nothing to protect computers from low-level network attacks. Many users and industry pundits focus on "users-as-non-admins" and sometimes lose sight of attacks that do not require human interaction. No user confirmation, no user-based least-privilege defense will protect a computer from an attack that takes advantage of a vulnerability in a network facing process that has no user interaction, such as DNS server, e-mail server, or Web server. As Sun Tzu said in *The Art of War*, "So in war, the way is to avoid what is strong and to strike at what is weak." If a software product shores up its defenses in one area, it must shore them up everywhere else in the product.

Windows offers many network defenses, most notably native IPSec and IPv6 support, and a bi-directional firewall.

IPSec and IPv6 The reason why distributed denial-of-service (DDoS) attacks occur is because IPv4 is an unauthenticated protocol. UDP is one of the worst offenders because it is a connectionless protocol, and it is trivial to spoof UDP packets. But even with TCP, the initial SYN packet is unauthenticated, and a set of attack servers could easily incapacitate a vulnerable server on the Internet by sending millions of bogus TCP SYN packets, as we discuss in Chapter 7. There are many other kinds of TCP/IP-related issues, and the IETF is currently discussing the issues in depth. Two IETF documents of interest are RFC 4953 (*Defending TCP Against Spoofing Attacks*, July 2007) and RFC 4953 (*TCP SYN Flooding Attacks and Common Mitigations*, August 2007).

The problem with any potential solution that uses IPv4 is that IPv4 is fundamentally flawed. Enter IPSec and IPv6. IPSec and IPv6 both support authenticated network packets, as we discuss in Chapter 22.5. In Windows Vista and later, IPv6 is enabled by default. IPv4 is enabled by default as well, but over time, Microsoft anticipates that more of the world's networks will migrate to the much more secure protocol. A good example of this is the XBOX Live online network. The core XBOX operating system is a stripped-down version of Windows, but its core networking protocol is essentially IPSec. The XBOX Live team did not want to use IPv4 because the team knew their servers would be under constant DDoS attack. Requiring IPSec substantially raises the bar on the attackers.

Firewall All versions of Windows since Windows XP have included a built-in software firewall. The version included with Windows XP was limited in that (1) it was not enabled by default, and (2) its configuration was limited to blocking only inbound connections on specific ports. The firewall in Windows XP SP2 was substantially improved to address one core issue: Users with multiple computers in the home wanted to share files and print documents, but the old firewall would only allow this to happen if the file and print ports (TCP 139 and 445) were open to the Internet. So in Windows XP SP2, there is an option to open a port, but only on the local subnet. The other change in Windows XP SP2, and by far the most important, is that the firewall is enabled by default.

Windows Vista and later add two other functions. First the firewall is a fully integrated component of the rewritten TCP/IP networking stack. Second, the firewall supports optionally blocking outbound connections. Some analysts believe blocking outbound connections is "security theater," not real security. Here's why. Let's say a user has a browser installed (it doesn't matter which one), and the user allows the browser to make outbound connections without prompting the user for confirmation. Malware writers will simply leverage the browser to run their malicious code from within the browser, so to the firewall, it looks like the browser is making the request, which is true. The firewall in Windows Vista is intended for management and policy enforcement, not for protection against malicious code.

All firewalls that support outbound connection blocking can easily be circumvented unless the user wishes to be prompted for every single outbound connection, in which case the user will totally frustrated after 10 minutes of typical use on the Internet.

Let's now discuss another set of defensive technologies in Windows: buffer overrun defenses.

Memory Corruption Defenses

In the previous edition, this section was entitled "Buffer Overrun Defenses," but in the author's opinion, the term "buffer overrun" is much too restrictive. Any form of memory corruption, be it caused by overwriting the end of the buffer, underrunning a buffer, or writing data to arbitrary memory locations can be catastrophic.

Most operating systems today, indeed much software in use today, is written in the C and C++ programming languages. C was designed as a high-level assembly language, and because of that requirement, C gives the developer direct access to memory through pointers, as we discuss in Chapter 10. Pointers simply point to a memory location. For example, in the following code snippet, the pointer *p* points to an array of 32 characters (a character is an 8-bit value) named password.

```
char password[32];
char *p = password;
```

With this powerful functionality comes risk: the ability to corrupt memory. Because of the risks of using C and C++, most people's first reaction is, "why not just rewrite everything in [insert language dejour]?" There are two reasons. The first is the same reason that the world's cars do not run on hydrogen. It is a great idea, and it is good for the planet, but gasoline has a massive momentum behind it because people know how to get oil from the ground, refine it, ship it, store it, pump it, build engines that use it, repair engines that use it, and so on. There are also problems with hydrogen that still make it impractical today. The same reasoning applies for replacing C and C++ with Java or C#. These languages and run-time environments are not quite up to the task for building operating systems. That may change in the future, but it will be a monumental task to convert C and C++ code to Java or C#.

The other reason is that simply replacing C and C++ with another language does not solve the real problem, which is that software developers have too much trust in the data they receive.

Memory corruption vulnerabilities when the application does not constrain write operations to the correct memory locations. For example, a buffer overrun occurs because the developer expects a buffer of 32 bytes, and the attack provides a buffer that is larger. In the author's opinion, the real way to solve the buffer overrun problem is to teach new developers (and jaded developers, for that matter) the simple rule of never trusting input and to identify data as the data enter the system and to sanitize or reject the data, as we discuss in Chapter 11.

Taking the example code above, the following is a classic buffer overrun example:

```
void ParseData(char *pwd) {
   char password[32];
   strcpy(password, pwd);
   // etc.
}
```

The problem with this code is that the strcpy function continues copying pwd into password and stops only when it hits a NULL character ('\0') in the source string, pwd. If the attacker controls pwd, then he or she can determine where the trailing NULL resides, and if the attacker decided to place it after the 32nd character in pwd, strcpy overflows the password buffer. This example is a classic "stack smash," because the buffer overflow corrupts the password buffer, which resides on the function's stack.

Here is another example:

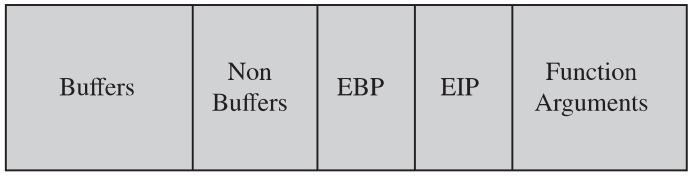
```
char t[64];
t[x] = y;
```

In this example, if the attacker controls "x", then he can write "y" to any location in memory.

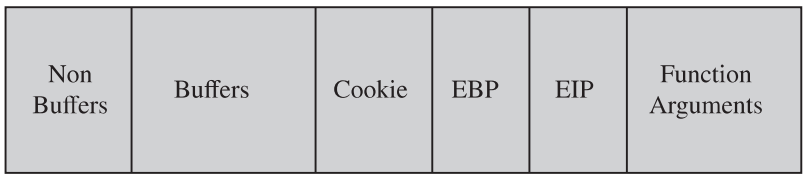
So let's look at some of the stack defenses enabled by default in Windows today.

Stack-based Buffer Overrun Detection (/GS) Normally in Windows, a function's stack looks like Figure 26.2a. You will notice two interesting items on the stack, EBP (extended base pointer) and EIP (extended instruction pointer). When the function returns, it must continue execution at the next instruction after the instruction that called this function. The CPU does this by taking the values off the stack (called popping) and populating the EBP and EIP registers. Here is where the fun starts. If the attacker can overflow the buffer on the stack, he or she can overrun the data used to populate the EBP and EIP registers with values under his or her control and hence change the application's execution flow. The source code for Windows XP SP2 is compiled with a special compiler option in Microsoft Visual C++ to add defenses to the function's stack. The compiler switch is /GS, and it is usable by anyone with access to a Visual C++ compiler. Once the code is compiled with this option, the stack is laid out as shown in Figure 26.2b.

As you can see, a cookie has been inserted between stack data and the function return address. This random value is checked when the function exits, and if the cookie is corrupted, the application is halted. You will also notice that buffers on the stack are placed in higher memory than nonbuffers, such as function pointers, C++



(a) Without /GS option



(b) With /GS option

Figure 26.2 Stack Layout in Windows Vista

objects, and scalar values. The reason for this is to make it harder for some attacks to succeed. Function pointers and C++ objects with virtual destructors (which are simply function pointers) are also subject to attack because they determine execution flow. If these constructs are placed in memory higher than buffers, then, for example, overflowing a buffer could corrupt a function pointer. By switching the order around, the attacker must take advantage of a buffer *underrun*, which is rarer, to successfully corrupt the function pointer. There are variants of the buffer overrun that will still corrupt a function pointer, such as corrupting a stack frame in higher memory, but that's beyond the scope of this chapter.

/GS does have one weakness—when the code is compiled, the compiler applies heuristics to determine which functions to protect, hence /GS does not affect every function, it affects only functions that have at least 4-bytes of contiguous stack char-data and only when the function takes a pointer or buffer as an argument. To address this potential issue, Microsoft added an option to relax the heuristics that more functions are protected. The option is named strict_gs_check; more information can be found here (`http://blogs.msdn.com/b/michael_howard/archive/2007/04/03/hardening-stack-based-buffer-overrun-detection-in-vc-2005-sp1.aspx`).

No eXecute Named NX by Advanced Micro Devices (AMD), Data Execution Prevention (DEP) by Microsoft, and eXecution Disable (XD) by Intel, this technology requires CPU support that helps prevent code from executing in data segments. Most modern Intel CPUs support this capability today, and all current AMD CPUs support NX. ARM-based CPUs also support NX. DEP support was first introduced in Windows XP SP2 and is a critically important defense in Windows, especially when used with address space layout randomization (ASLR), which we will explain later.

The goal of NX is to prevent data executing. Most buffer overrun exploits enter a computer system as data, and then those data are executed. By default, most system components in Windows and applications can use NX by linking with the /NXCOMPAT linker option.

We will discuss NX and ASLR in the context of a browser defense shortly.

Stack Randomization This defense is available in Windows Vista and later. When a thread starts in Windows, the operating system will randomize the stack base address by 0–31 pages. Normally, a page is 4k bytes in size. Once the page is chosen, a random offset is chosen within the page, and the stack starts from that spot. The purpose of randomization is to remove some of the predictability from the attacker. Attackers love predictability because it makes it more likely that an attack will be successful.

There is more to life than stack-based buffer overruns. Data can also reside in another kind of system memory, the heap.

Head-based Buffer Overrun Detection The seminal buffer overrun paper is "Smashing the Stack for Fun and Profit" by AlephOne [LEVY96]. It is a fantastic read. For quite some time, "smashing the stack" was the attack dejour, and little attention was paid to heap-based buffer overruns. Eventually, people realized that even though the heap is laid out differently than the stack, heap-based buffer overruns are exploitable, and can lead to code execution. The nature of such attacks is something you should research [LITC03].

The first heap defense, added to Windows XP SP2, is to add a random value to each heap block and detect that this cookie has not been tampered with. If the cookie has changed, then the heap has been corrupted and the application could be forced to crash. Note the application crash is not due to instability in the application caused by data corruption; rather the heap manager detects the corruption and fails the application. The process of shutting down an application in this manner is often called "failstop."

The second defense is heap integrity checking; when heap blocks are freed, metadata in the heap data structures are checked for validity, and if the data are compromised, either the heap block is leaked or the application crashes.

Other important defenses have been added including removing heap-block elements that were used by attackers.

Heap Randomization Like stack randomization, heap randomization is designed to take some of the predictability away from the attacker, but it applies to the heap. When a heap is created, the start of the heap is offset by 0–4 MB. Again, this makes things a little harder for the attacker. This feature is new to Windows Vista.

Image Randomization As far as making thinks a little less predictable for the attacker, Windows Vista also adds image randomization. When the operating system boots, it starts up in one of 256 configurations. In other words, the entire operating system is shifted up or down in memory when it is booted. The best way to think of this is to imagine that a random number is selected at boot, and every operating system component is loaded as an offset from that location, but the offset between each component is fixed. Again, this makes the operating system less predictable for attackers and makes it less likely that an exploit will succeed.

Service Restart Policy In Windows, a service can be configured to restart if the service fails. This is great for reliability but lousy for security, because if an attacker attacks the service and the attack fails but the service crashes, the service might restart and the attacker will have another chance to attack the system. In Windows Vista, Microsoft set some of the critical services to restart only twice, after which the service will not restart unless the administrator manually restarts the service. This gives the attacker only two attempts to get the attack to work, and in the face of stack, heap, and image randomization, it is much more difficult.

Note that a full description of all the defenses described in this section, and how to use them in your own code, can be found in [HOWA07].

26.4 BROWSER DEFENSES

There is no point of attack quite like a Web browser. A Web browser interprets a complex language, HTML, and renders the results. But a webpage can also contain code in the form of scripting languages such as JavaScript, or richer, more capable code such as ActiveX controls, Flash, Java applets, or .NET applications; and mixing code and data is bad for security. All of this code and data makes for a rich and productive end-user environment, but it is hard to secure. Web browsers can also render various multimedia objects such as sound, JPEG, BMP, GIF, animated GIFs, and PNG files. Many file formats are rendered by helper objects, called MIME handlers. Examples include video formats such as Quicktime, Windows Media Player, or Real Player.

A malicious webpage could take advantage of many possible attack vectors; some vectors are under the direct control of the browser, and some are not.

With this setting in mind, Microsoft decided to add many defenses to Internet Explorer, and each successive version adds more defenses. Substantially, these approaches carry over to their replacement browser, Microsoft Edge. Perhaps the most important single defense is ActiveX opt-in. An ActiveX control is a binary object that can potentially be invoked by the Web browser using the <OBJECT> HTML tag, or by calling the object directly from script. Many common Web browser extensions are implemented as ActiveX controls; probably the most well known is Adobe Flash. It is possible for ActiveX controls to be malicious, and chances are very good that a user already has one or more ActiveX controls installed on his or her computer. But does the user know which controls are installed? We would wager that for most users, the answer is a resounding, "no!" Internet Explorer adds a new feature called "ActiveX opt-in," which essentially unloads ActiveX controls by default, and when a control is used for the first time, the user is prompted to allow the control to run. At this point, the user knows that the control is on the computer. Microsoft Edge does not support ActiveX but has similar protections.

Another important defense in Internet Explorer is protected mode. When this default configuration is used, Internet Explorer runs at low-integrity level, making it more difficult for malware to manipulate the operating system which operates at a medium- or higher-integrity level. See Section 26.1 for a discussion of integrity levels in Windows.

Current versions of Internet Explorer also enable ASLR and DEP by default. In IE7, the options were available, but not enabled by default because many common components, such as Flash, Acrobat Reader, QuickTime, the Java VM, and more, broke. Microsoft worked very closely with the component vendors to make them operate correctly with ASLR and DEP.

It is important to point out that Protected Mode, DEP and ASLR only help mitigate against memory corruption vulnerabilities, they do not help protect against Phishing attacks nor common Web-specific vulnerabilities such as cross-site scripting (XSS.) Microsoft added defenses to Internet Explorer to help address these issues. First, a cross-site scripting detection logic to help detect and prevent some classes of XSS. Some would argue that adding this logic to a Web browser is a bad idea, because XSS prevention should be the goal of a Web application. Personally, I think it is a great idea, because we obviously cannot rely on Web site developers to write secure and XSS-free Web-based applications. This IE defense is simply an extra defensive layer. The second defense is a phishing filter; simply put when a user visits a Website, the site's URL is sent to a service that determines if the site is a known phishing or malware-distribution site. The user is warned if the site is suspicious.

A final defense to help prevent users being tracked is a privacy-enhancing mode name InPrivate mode, which does not persist cookies or site history.

26.5 CRYPTOGRAPHIC SERVICES

Windows includes a complete set of cryptographic functionality, from low-level cryptographic primitives for encryption, hashing, and signing to full-fledged cryptographic defenses, such as the Encrypting File System (EFS), Data Protection API, and BitLocker. Let's look at each of these features in more detail.

Encrypting File System

EFS allows files and directories to be encrypted and decrypted transparently for authorized users. All versions of Windows since Windows 2000 support EFS. On the surface, EFS is very simple; a user or administrator marks a directory to use EFS, and from that point on, any file created in that directory is encrypted. It is possible to encrypt single files, but this is problematic because it is common for applications to create temporary files while manipulating the file in question. But if the target file is marked for encryption, the temporary files are not encrypted, and if the temporary files contain sensitive data, the data are not protected. The way to fix this is to encrypt the entire directory.

At a very high level, EFS works by generating a random file encryption key (FEK) and storing that key, encrypted using the user's encryption key. This key is protected using the Data Protection API (DPAPI) in Windows, and the key used by DPAPI is derived from the user's password. The process of allowing a new user to access an EFS-encrypted file is simple too. The FEK is encrypted with the user's key, and it is stored alongside the other user keys in the file metadata.

EFS also supports the concept of a file recovery agent, a special capability to decrypt files if for some reason the user's lose their EFS keys.

The cornerstone of EFS is DPAPI, which is the next topic.

Data Protection API

The data protection API (DPAPI) allows users to encrypt and decrypt data transparently; in other words, the tasks of maintaining and protecting encryption keys are removed from the user and administered by the operating system. When DPAPI is used to encrypt user data, the encryption keys are derived in part from the user's password. A full explanation of how DPAPI works is available at [NAI01]. Again, the beauty of DPAPI lies in removing the key management problem from the user and developers. Developers need only call one of two functions, *CryptProtectData* to encrypt and *CryptUnprotectData* to decrypt. These functions also add a message authentication code to the encrypted data to help detect tampering.

BitLocker

Windows adds a much-needed defense to the operating system, BitLocker Drive Encryption. The core threat this technology helps to mitigate is data disclosure on stolen laptops. BitLocker encrypts the entire volume using AES, and the encryption key is stored either on a USB drive or within a Trusted Platform Module (TPM) chip on the computer motherboard. When booting a system that requires the USB device, the device must be present so the keys can be read by the computer, after which BitLocker decrypts the hard drive on the fly, with no perceptible performance degradation. The downside to using a USB device is that if the device is lost, the user loses the encryption keys and cannot decrypt. Thankfully, BitLocker can integrate with Active Directory to store the encryption keys, and BitLocker also supports key recovery.

Perhaps the most important aspect of BitLocker is that, like most security settings in Windows, BitLocker policy can be set as a policy for a single computer and that policy "pushed" to computers that use Active Directory.

BitLocker is the first technology in Windows to use a TPM chip, and that's the next topic.

Trusted Platform Module

The Trusted Platform Module (TPM) is the product of a specification from the Trusted Computing Group, designed to enhance system security by moving many sensitive cryptographic operations into hardware. Many software-based attacks do not affect a hardware solution, such as TPM. TPMs are discussed in Chapter 27.

Windows Vista supports TPM version 1.2.

The best-known feature that uses the TPM, if one is available, is BitLocker Drive Encryption. When a TPM is present and the system is configured appropriately, Windows will use the TPM to validate that the operating system has not been tampered with. This is known as trusted boot, or secure startup, and as the OS boots, critical portions are hashed and the hashes verified.

Microsoft expects more software vendors to make use of the TPM over time, especially as most laptops shipping today include a TPM on the motherboard, and more desktop and server computers ship with embedded TPMs.

26.6 COMMON CRITERIA

Versions of Windows since Windows 2000 have earned Common Criteria EAL4 + Flaw Remediation (ALC_FLR.3) or are in the process of being accredited. What is critically important about the work Microsoft has undertaken in getting its operating systems accredited is that the software stack (the security target) that is evaluated is useable. It is not a whittled-down configuration that is just an FTP server, for example. You can look at the Windows Server 2003 and Windows XP product validation reports at [NIAP07].

26.7 REFERENCES

HOWA07 Howard, M., and LeBlanc, D. *Writing Secure Code for Windows Vista*. Redmond, WA: Microsoft Press, 2006.

LEVY9 6 Levy, E., "Smashing the Stack for Fun and Profit." *Phrack Magazine*, file 14, Issue 49, November 1996.

LITC03 Litchfield, D."Defeating the Stack Based Buffer Overflow Prevention Mechanism of Microsoft Windows 2003 Server." *NGS Software White Paper*, 8 September 2003. ngssoftware.com/papers/defeating-w2k3-stack-protection.pdf

NAI01 NAI Labs. Windows Data Protection. Windows Developer Network, October 2001. https://msdn.microsoft.com/en-us/library/ms995355.aspx

NIAP07 National Information Assurance Partnership. Common *Criteria Evaluation and Validation Scheme Validation Report — Microsoft Windows 2003 Server and XP Workstation*. April 2007. https://msdn.microsoft.com/en-us/library/dd229319.aspx

26.8 KEY TERMS AND PROJECTS

Key Terms

Active Directory	authentication packages	NetLogon
BitLocker Drive Encryption	domain account	Security Account Manager
TPM	local account	Security Reference Monitor
Access Control Lists	Local Security Authority	WinLogon

Projects

A series of projects are contained in a document, filename `WindowsProjects.pdf`, available at this book's website. These projects were developed by Ricky Magalhaes of Fastennet Security. These are designed to help you learn about Windows security. These are not review questions but rather exercises that expose parts of Windows to you in security context, and will help you to learn parts of windows security.