part 12

Additional Database Topics: Security

This page intentionally left blank

chapter 30

Database Security

This chapter discusses techniques for securing databases against a variety of threats. It also presents schemes of providing access privileges to authorized users. Some of the security threats to databases—such as SQL injection—will be presented. At the end of the chapter, we summarize how a mainstream RDBMS—specifically, the Oracle system—provides different types of security. We start in Section 30.1 with an introduction to security issues and the threats to databases, and we give an overview of the control measures that are covered in the rest of this chapter. We also comment on the relationship between data security and privacy as it applies to personal information. Section 30.2 discusses the mechanisms used to grant and revoke privileges in relational database systems and in SQL, mechanisms that are often referred to as **discretionary access control**. In Section 30.3, we present an overview of the mechanisms for enforcing multiple levels of security—a particular concern in database system security that is known as **mandatory access control**. Section 30.3 also introduces the more recently developed strategies of **role-based access control**, and label-based and row-based security. Section 30.3 also provides a brief discussion of XML access control. Section 30.4 discusses a major threat to databases—SQL injection—and discusses some of the proposed preventive measures against it. Section 30.5 briefly discusses the security problem in statistical databases. Section 30.6 introduces the topic of flow control and mentions problems associated with covert channels. Section 30.7 provides a brief summary of encryption and symmetric key and asymmetric (public) key infrastructure schemes. It also discusses digital certificates. Section 30.8 introduces privacy-preserving techniques, and Section 30.9 presents the current challenges to database security. In Section 30.10, we discuss Oracle label-based security. Finally, Section 30.11 summarizes the chapter. Readers who are interested only in basic database security mechanisms will find it sufficient to cover the material in Sections 30.1 and 30.2.

30.1 Introduction to Database Security Issues[1]

30.1.1 Types of Security

Database security is a broad area that addresses many issues, including the following:

- Various legal and ethical issues regarding the right to access certain information—for example, some information may be deemed to be private and cannot be accessed legally by unauthorized organizations or persons. In the United States, there are numerous laws governing privacy of information.
- Policy issues at the governmental, institutional, or corporate level regarding what kinds of information should not be made publicly available—for example, credit ratings and personal medical records.
- System-related issues such as the *system levels* at which various security functions should be enforced—for example, whether a security function should be handled at the physical hardware level, the operating system level, or the DBMS level.
- The need in some organizations to identify multiple *security levels* and to categorize the data and users based on these classifications—for example, top secret, secret, confidential, and unclassified. The security policy of the organization with respect to permitting access to various classifications of data must be enforced.

Threats to Databases. Threats to databases can result in the loss or degradation of some or all of the following commonly accepted security goals: integrity, availability, and confidentiality.

- **Loss of integrity.** Database integrity refers to the requirement that information be protected from improper modification. Modification of data includes creating, inserting, and updating data; changing the status of data; and deleting data. Integrity is lost if unauthorized changes are made to the data by either intentional or accidental acts. If the loss of system or data integrity is not corrected, continued use of the contaminated system or corrupted data could result in inaccuracy, fraud, or erroneous decisions.
- **Loss of availability.** *Database availability* refers to making objects available to a human user or a program who/which has a legitimate right to those data objects. *Loss of availability* occurs when the user or program cannot access these objects.
- **Loss of confidentiality.** Database confidentiality refers to the protection of data from unauthorized disclosure. The impact of unauthorized disclosure of confidential information can range from violation of the Data Privacy Act to the jeopardization of national security. Unauthorized, unanticipated, or unintentional disclosure could result in loss of public confidence, embarrassment, or legal action against the organization.

[1]The substantial contributions of Fariborz Farahmand, Bharath Rengarajan, and Frank Rietta to this and subsequent sections of this chapter is much appreciated.

Database Security: Not an Isolated Concern. When considering the threats facing databases, it is important to remember that the database management system alone cannot be responsible for maintaining the confidentiality, integrity, and availability of the data. Rather, the database works as part of a network of services, including applications, Web servers, firewalls, SSL terminators, and security monitoring systems. Because security of an overall system is only as strong as its weakest link, a database may be compromised even if it would have been perfectly secure on its own merits.

To protect databases against the threats discussed above, it is common to implement *four kinds of control measures*: access control, inference control, flow control, and encryption. We discuss each of these in this chapter.

In a multiuser database system, the DBMS must provide techniques to enable certain users or user groups to access selected portions of a database without gaining access to the rest of the database. This is particularly important when a large integrated database is to be used by many different users within the same organization. For example, sensitive information such as employee salaries or performance reviews should be kept confidential from most of the database system's users. A DBMS typically includes a **database security and authorization subsystem** that is responsible for ensuring the security of portions of a database against unauthorized access. It is now customary to refer to two types of database security mechanisms:

- **Discretionary security mechanisms.** These are used to grant privileges to users, including the capability to access specific data files, records, or fields in a specified mode (such as read, insert, delete, or update).
- **Mandatory security mechanisms.** These are used to enforce multilevel security by classifying the data and users into various security classes (or levels) and then implementing the appropriate security policy of the organization. For example, a typical security policy is to permit users at a certain classification (or clearance) level to see only the data items classified at the user's own (or lower) classification level. An extension of this is *role-based security,* which enforces policies and privileges based on the concept of organizational roles. (See Section 30.4.2 for role based access control.)

We discuss discretionary security in Section 30.2 and mandatory and role-based security in Section 30.3.

30.1.2 Control Measures

Four main control measures are used to provide security of data in databases:

- Access control
- Inference control
- Flow control
- Data encryption

A security problem common to computer systems is that of preventing unauthorized persons from accessing the system itself, either to obtain information or to make malicious changes in a portion of the database. The security mechanism of a DBMS must include provisions for restricting access to the database system as a whole. This function, called **access control**, is handled by creating user accounts and passwords to control the login process by the DBMS. We discuss access control techniques in Section 30.1.3.

Statistical databases are used to provide statistical information or summaries of values based on various criteria. For example, a database for population statistics may provide statistics based on age groups, income levels, household size, education levels, and other criteria. Statistical database users such as government statisticians or market research firms are allowed to access the database to retrieve statistical information about a population but not to access the detailed confidential information about specific individuals. Security for statistical databases must ensure that information about individuals cannot be accessed. It is sometimes possible to deduce or infer certain facts concerning individuals from queries that involve only summary statistics on groups; consequently, this must not be permitted either. This problem, called **statistical database security**, is discussed briefly in Section 30.4. The corresponding control measures are called **inference control** measures.

Another security issue is that of **flow control**, which prevents information from flowing in such a way that it reaches unauthorized users. Flow control is discussed in Section 30.6. **Covert channels** are pathways on which information flows implicitly in ways that violate the security policy of an organization. We briefly discuss some issues related to covert channels in Section 30.6.1.

A final control measure is **data encryption**, which is used to protect sensitive data (such as credit card numbers) that is transmitted via some type of communications network. Encryption can be used to provide additional protection for sensitive portions of a database as well. The data is **encoded** using some coding algorithm. An unauthorized user who accesses encoded data will have difficulty deciphering it, but authorized users are given decoding or decrypting algorithms (or keys) to decipher the data. Encrypting techniques that are very difficult to decode without a key have been developed for military applications. However, encrypted database records are used today in both private organizations and governmental and military applications. In fact, state and federal laws prescribe encryption for any system that deals with legally protected personal information. For example, according to Georgia Law (OCGA 10-1-911):

> "Personal information" means an individual's first name or first initial and last name in combination with any one or more of the following data elements, when either the name or the data elements are not encrypted or redacted:
>
> - Social security number;
> - Driver's license number or state identification card number;

- Account number, credit card number, or debit card number, if circumstances exist wherein such a number could be used without additional identifying information, access codes, or passwords;
- Account passwords or personal identification numbers or other access codes

Because laws defining what constitutes personal information vary from state to state, systems must protect individuals' privacy and enforce privacy measures adequately. Discretionary access control (see Section 30.2) alone may not suffice. Section 30.7 briefly discusses encryption techniques, including popular techniques such as public key encryption (which is heavily used to support Web-based transactions against databases) and digital signatures (which are used in personal communications).

A comprehensive discussion of security in computer systems and databases is outside the scope of this text. We give only a brief overview of database security techniques here. Network- and communication-based security is also a vast topic that we do not cover. For a comprehensive discussion, the interested reader can refer to several of the references discussed in the Selected Bibliography at the end of this chapter.

30.1.3 Database Security and the DBA

As we discussed in Chapter 1, the database administrator (DBA) is the central authority for managing a database system. The DBA's responsibilities include granting privileges to users who need to use the system and classifying users and data in accordance with the policy of the organization. The DBA has a **DBA account** in the DBMS, sometimes called a **system** or **superuser account**, which provides powerful capabilities that are not made available to regular database accounts and users.[2] DBA-privileged commands include commands for granting and revoking privileges to individual accounts, users, or user groups and for performing the following types of actions:

1. **Account creation.** This action creates a new account and password for a user or a group of users to enable access to the DBMS.
2. **Privilege granting.** This action permits the DBA to grant certain privileges to certain accounts.
3. **Privilege revocation.** This action permits the DBA to revoke (cancel) certain privileges that were previously given to certain accounts.
4. **Security level assignment.** This action consists of assigning user accounts to the appropriate security clearance level.

The DBA is responsible for the overall security of the database system. Action 1 in the preceding list is used to control access to the DBMS as a whole, whereas actions 2 and 3 are used to control *discretionary* database authorization, and action 4 is used to control *mandatory* authorization.

[2]This account is similar to the *root* or *superuser* accounts that are given to computer system administrators and that allow access to restricted operating system commands.

30.1.4 Access Control, User Accounts, and Database Audits

Whenever a person or a group of persons needs to access a database system, the individual or group must first apply for a user account. The DBA will then create a new **account number** and **password** for the user if there is a legitimate need to access the database. The user must **log in** to the DBMS by entering the account number and password whenever database access is needed. The DBMS checks that the account number and password are valid; if they are, the user is permitted to use the DBMS and to access the database. Application programs can also be considered users and are required to log in to the database (see Chapter 10).

It is straightforward to keep track of database users and their accounts and passwords by creating an encrypted table or file with two fields: AccountNumber and Password. This table can easily be maintained by the DBMS. Whenever a new account is created, a new record is inserted into the table. When an account is canceled, the corresponding record must be deleted from the table.

The database system must also keep track of all operations on the database that are applied by a certain user throughout each **login session**, which consists of the sequence of database interactions that a user performs from the time of logging in to the time of logging off. When a user logs in, the DBMS can record the user's account number and associate it with the computer or device from which the user logged in. All operations applied from that computer or device are attributed to the user's account until the user logs off. It is particularly important to keep track of update operations that are applied to the database so that, if the database is tampered with, the DBA can determine which user did the tampering.

To keep a record of all updates applied to the database and of particular users who applied each update, we can modify the *system log*. Recall from Chapters 20 and 22 that the **system log** includes an entry for each operation applied to the database that may be required for recovery from a transaction failure or system crash. We can expand the log entries so that they also include the account number of the user and the online computer or device ID that applied each operation recorded in the log. If any tampering with the database is suspected, a **database audit** is performed, which consists of reviewing the log to examine all accesses and operations applied to the database during a certain time period. When an illegal or unauthorized operation is found, the DBA can determine the account number used to perform the operation. Database audits are particularly important for sensitive databases that are updated by many transactions and users, such as a banking database that can be updated by thousands of bank tellers. A database log that is used mainly for security purposes serves as an **audit trail**.

30.1.5 Sensitive Data and Types of Disclosures

Sensitivity of data is a measure of the importance assigned to the data by its owner for the purpose of denoting its need for protection. Some databases contain only sensitive data whereas other databases may contain no sensitive data at all. Handling databases that fall at these two extremes is relatively easy because

such databases can be covered by access control, which is explained in the next section. The situation becomes tricky when some of the data is sensitive whereas other data is not.

Several factors can cause data to be classified as sensitive:

1. **Inherently sensitive.** The value of the data itself may be so revealing or confidential that it becomes sensitive—for example, a person's salary or who a patient has HIV/AIDS.
2. **From a sensitive source.** The source of the data may indicate a need for secrecy—for example, an informer whose identity must be kept secret.
3. **Declared sensitive.** The owner of the data may have explicitly declared it as sensitive.
4. **A sensitive attribute or sensitive record.** The particular attribute or record may have been declared sensitive—for example, the salary attribute of an employee or the salary history record in a personnel database.
5. **Sensitive in relation to previously disclosed data.** Some data may not be sensitive by itself but will become sensitive in the presence of some other data—for example, the exact latitude and longitude information for a location where some previously recorded event happened that was later deemed sensitive.

It is the responsibility of the database administrator and security administrator to collectively enforce the security policies of an organization. This dictates whether access should or should not be permitted to a certain database attribute (also known as a *table column* or a *data element*) for individual users or for categories of users. Several factors must be considered before deciding whether it is safe to reveal the data. The three most important factors are data availability, access acceptability, and authenticity assurance.

1. **Data availability.** If a user is updating a field, then this field becomes inaccessible and other users should not be able to view this data. This blocking is only temporary and only to ensure that no user sees any inaccurate data. This is typically handled by the concurrency control mechanism (see Chapter 21).
2. **Access acceptability.** Data should only be revealed to authorized users. A database administrator may also deny access to a user request even if the request does not directly access a sensitive data item, on the grounds that the requested data may reveal information about the sensitive data that the user is not authorized to have.
3. **Authenticity assurance.** Before granting access, certain external characteristics about the user may also be considered. For example, a user may only be permitted access during working hours. The system may track previous queries to ensure that a combination of queries does not reveal sensitive data. The latter is particularly relevant to statistical database queries (see Section 30.5).

The term *precision*, when used in the security area, refers to allowing as much as possible of the data to be available, subject to protecting exactly the subset of data that is sensitive. The definitions of *security* versus *precision* are as follows:

- **Security:** Means of ensuring that data is kept safe from corruption and that access to it is suitably controlled. To provide security means to disclose only nonsensitive data and to reject any query that references a sensitive field.
- **Precision:** To protect all sensitive data while disclosing or making available as much nonsensitive data as possible. Note that this definition of *precision* is not related to the precision of information retrieval defined in Section 27.6.1.

The ideal combination is to maintain perfect security with maximum precision. If we want to maintain security, precision must be sacrificed to some degree. Hence there is typically a tradeoff between security and precision.

30.1.6 Relationship between Information Security and Information Privacy

The rapid advancement of the use of information technology (IT) in industry, government, and academia raises challenging questions and problems regarding the protection and use of personal information. Questions of *who* has *what* rights to information about individuals for *which* purposes become more important as we move toward a world in which it is technically possible to know just about anything about anyone.

Deciding how to design privacy considerations in technology for the future includes philosophical, legal, and practical dimensions. There is a considerable overlap between issues related to access to resources (security) and issues related to appropriate use of information (privacy). We now define the difference between *security* and *privacy*.

Security in information technology refers to many aspects of protecting a system from unauthorized use, including authentication of users, information encryption, access control, firewall policies, and intrusion detection. For our purposes here, we will limit our treatment of security to the concepts associated with how well a system can protect access to information it contains. The concept of **privacy** goes beyond security. Privacy examines how well the use of personal information that the system acquires about a user conforms to the explicit or implicit assumptions regarding that use. From an end user perspective, privacy can be considered from two different perspectives: *preventing storage* of personal information versus *ensuring appropriate use* of personal information.

For the purposes of this chapter, a simple but useful definition of **privacy** is *the ability of individuals to control the terms under which their personal information is acquired and used*. In summary, security involves technology to ensure that information is appropriately protected. Security is a required building block for privacy. Privacy involves mechanisms to support compliance with some basic principles and other explicitly stated policies. One basic principle is that people should be informed

about information collection, told in advance what will be done with their information, and given a reasonable opportunity to approve or disapprove of such use of the information. A related concept, **trust**, relates to both security and privacy and is seen as increasing when it is perceived that both security and privacy are provided for.

30.2 Discretionary Access Control Based on Granting and Revoking Privileges

The typical method of enforcing **discretionary access control** in a database system is based on the granting and revoking of **privileges**. Let us consider privileges in the context of a relational DBMS. In particular, we will discuss a system of privileges somewhat similar to the one originally developed for the SQL language (see Chapters 7 and 8). Many current relational DBMSs use some variation of this technique. The main idea is to include statements in the query language that allow the DBA and selected users to grant and revoke privileges.

30.2.1 Types of Discretionary Privileges

In SQL2 and later versions,[3] the concept of an **authorization identifier** is used to refer, roughly speaking, to a user account (or group of user accounts). For simplicity, we will use the words *user* or *account* interchangeably in place of *authorization identifier*. The DBMS must provide selective access to each relation in the database based on specific accounts. Operations may also be controlled; thus, having an account does not necessarily entitle the account holder to all the functionality provided by the DBMS. Informally, there are two levels for assigning privileges to use the database system:

- **The account level.** At this level, the DBA specifies the particular privileges that each account holds independently of the relations in the database.
- **The relation (or table) level.** At this level, the DBA can control the privilege to access each individual relation or view in the database.

The privileges at the **account level** apply to the capabilities provided to the account itself and can include the CREATE SCHEMA or CREATE TABLE privilege, to create a schema or base relation; the CREATE VIEW privilege; the ALTER privilege, to apply schema changes such as adding or removing attributes from relations; the DROP privilege, to delete relations or views; the MODIFY privilege, to insert, delete, or update tuples; and the SELECT privilege, to retrieve information from the database by using a SELECT query. Notice that these account privileges apply to the account in general. If a certain account does not have the CREATE TABLE privilege, no relations can be created from that account. Account-level privileges *are not* defined as part of SQL2; they are left to the DBMS implementers to define. In earlier versions of SQL, a CREATETAB privilege existed to give an account the privilege to create tables (relations).

[3]Discretionary privileges were incorporated into SQL2 and are applicable to later versions of SQL.

The second level of privileges applies to the **relation level**, which includes base relations and virtual (view) relations. These privileges *are* defined for SQL2. In the following discussion, the term *relation* may refer either to a base relation or to a view, unless we explicitly specify one or the other. Privileges at the relation level specify for each user the individual relations on which each type of command can be applied. Some privileges also refer to individual columns (attributes) of relations. SQL2 commands provide privileges at the *relation and attribute level only*. Although this distinction is general, it makes it difficult to create accounts with limited privileges. The granting and revoking of privileges generally follow an authorization model for discretionary privileges known as the **access matrix model**, where the rows of a matrix M represent *subjects* (users, accounts, programs) and the columns represent *objects* (relations, records, columns, views, operations). Each position $M(i, j)$ in the matrix represents the types of privileges (read, write, update) that subject i holds on object j.

To control the granting and revoking of relation privileges, each relation R in a database is assigned an **owner account**, which is typically the account that was used when the relation was created in the first place. The owner of a relation is given *all* privileges on that relation. In SQL2, the DBA can assign an owner to a whole schema by creating the schema and associating the appropriate authorization identifier with that schema, using the CREATE SCHEMA command (see Section 7.1.1). The owner account holder can pass privileges on any of the owned relations to other users by **granting** privileges to their accounts. In SQL, the following types of privileges can be granted on each individual relation R:

- **SELECT (retrieval or read) privilege on** R. Gives the account retrieval privilege. In SQL, this gives the account the privilege to use the SELECT statement to retrieve tuples from R.
- **Modification privileges on** R. This gives the account the capability to modify the tuples of R. In SQL, this includes three privileges: UPDATE, DELETE, and INSERT. These correspond to the three SQL commands (see Section 7.4) for modifying a table R. Additionally, both the INSERT and UPDATE privileges can specify that only certain attributes of R can be modified by the account.
- **References privilege on** R. This gives the account the capability to *reference* (or refer to) a relation R when specifying integrity constraints. This privilege can also be restricted to specific attributes of R.

Notice that to create a view, the account must have the SELECT privilege on *all relations* involved in the view definition in order to specify the query that corresponds to the view.

30.2.2 Specifying Privileges through the Use of Views

The mechanism of **views** is an important *discretionary authorization mechanism* in its own right. For example, if the owner A of a relation R wants another account B to be able to retrieve only some fields of R, then A can create a view V of R that

includes only those attributes and then grant SELECT on V to B. The same applies to limiting B to retrieving only certain tuples of R; a view V' can be created by defining the view by means of a query that selects only those tuples from R that A wants to allow B to access. We will illustrate this discussion with the example given in Section 30.2.5.

30.2.3 Revoking of Privileges

In some cases, it is desirable to grant a privilege to a user temporarily. For example, the owner of a relation may want to grant the SELECT privilege to a user for a specific task and then revoke that privilege once the task is completed. Hence, a mechanism for **revoking** privileges is needed. In SQL, a REVOKE command is included for the purpose of canceling privileges. We will see how the REVOKE command is used in the example in Section 30.2.5.

30.2.4 Propagation of Privileges Using the GRANT OPTION

Whenever the owner A of a relation R grants a privilege on R to another account B, the privilege can be given to B *with* or *without* the **GRANT OPTION**. If the GRANT OPTION is given, this means that B can also grant that privilege on R to other accounts. Suppose that B is given the GRANT OPTION by A and that B then grants the privilege on R to a third account C, also with the GRANT OPTION. In this way, privileges on R can **propagate** to other accounts without the knowledge of the owner of R. If the owner account A now revokes the privilege granted to B, all the privileges that B propagated based on that privilege *should automatically be revoked* by the system.

It is possible for a user to receive a certain privilege from two or more sources. For example, A4 may receive a certain UPDATE R privilege from *both* A2 and A3. In such a case, if A2 revokes this privilege from A4, A4 will still continue to have the privilege by virtue of having been granted it from A3. If A3 later revokes the privilege from A4, A4 totally loses the privilege. Hence, a DBMS that allows propagation of privileges must keep track of how all the privileges were granted in the form of some internal log so that revoking of privileges can be done correctly and completely.

30.2.5 An Example to Illustrate Granting and Revoking of Privileges

Suppose that the DBA creates four accounts—A1, A2, A3, and A4—and wants only A1 to be able to create base relations. To do this, the DBA must issue the following GRANT command in SQL:

```
GRANT CREATETAB TO A1;
```

The CREATETAB (create table) privilege gives account A1 the capability to create new database tables (base relations) and is hence an *account privilege.* This privilege was part of earlier versions of SQL but is now left to each individual system

implementation to define. Note that A1 , A2, and so forth may be individuals, like John in IT department or Mary in marketing; but they may also be applications or programs that want to access a database.

In SQL2, the same effect can be accomplished by having the DBA issue a CREATE SCHEMA command, as follows:

CREATE SCHEMA EXAMPLE **AUTHORIZATION** A1;

User account A1 can now create tables under the schema called EXAMPLE. To continue our example, suppose that A1 creates the two base relations EMPLOYEE and DEPARTMENT shown in Figure 30.1; A1 is then the **owner** of these two relations and hence has *all the relation privileges* on each of them.

Next, suppose that account A1 wants to grant to account A2 the privilege to insert and delete tuples in both of these relations. However, A1 does not want A2 to be able to propagate these privileges to additional accounts. A1 can issue the following command:

GRANT INSERT, DELETE **ON** EMPLOYEE, DEPARTMENT **TO** A2;

Notice that the owner account A1 of a relation automatically has the GRANT OPTION, allowing it to grant privileges on the relation to other accounts. However, account A2 cannot grant INSERT and DELETE privileges on the EMPLOYEE and DEPARTMENT tables because A2 was not given the GRANT OPTION in the preceding command.

Next, suppose that A1 wants to allow account A3 to retrieve information from either of the two tables and also to be able to propagate the SELECT privilege to other accounts. A1 can issue the following command:

GRANT SELECT **ON** EMPLOYEE, DEPARTMENT **TO** A3 **WITH GRANT OPTION**;

The clause WITH GRANT OPTION means that A3 can now propagate the privilege to other accounts by using GRANT. For example, A3 can grant the SELECT privilege on the EMPLOYEE relation to A4 by issuing the following command:

GRANT SELECT **ON** EMPLOYEE **TO** A4;

Notice that A4 cannot propagate the SELECT privilege to other accounts because the GRANT OPTION was not given to A4.

Now suppose that A1 decides to revoke the SELECT privilege on the EMPLOYEE relation from A3; A1 then can issue this command:

REVOKE SELECT **ON** EMPLOYEE **FROM** A3;

Figure 30.1
Schemas for the two relations EMPLOYEE and DEPARTMENT.

EMPLOYEE

Name	Ssn	Bdate	Address	Sex	Salary	Dno

DEPARTMENT

Dnumber	Dname	Mgr_ssn

The DBMS must now revoke the SELECT privilege on EMPLOYEE from A3, and it must also *automatically revoke* the SELECT privilege on EMPLOYEE from A4. This is because A3 granted that privilege to A4, but A3 does not have the privilege any more.

Next, suppose that A1 wants to give back to A3 a limited capability to SELECT from the EMPLOYEE relation and wants to allow A3 to be able to propagate the privilege. The limitation is to retrieve only the Name, Bdate, and Address attributes and only for the tuples with Dno = 5. A1 then can create the following view:

```
CREATE VIEW A3EMPLOYEE AS
SELECT Name, Bdate, Address
FROM EMPLOYEE
WHERE Dno = 5;
```

After the view is created, A1 can grant SELECT on the view A3EMPLOYEE to A3 as follows:

```
GRANT SELECT ON A3EMPLOYEE TO A3 WITH GRANT OPTION;
```

Finally, suppose that A1 wants to allow A4 to update only the Salary attribute of EMPLOYEE; A1 can then issue the following command:

```
GRANT UPDATE ON EMPLOYEE (Salary) TO A4;
```

The UPDATE and INSERT privileges can specify particular attributes that may be updated or inserted in a relation. Other privileges (SELECT, DELETE) are not attribute specific, because this specificity can easily be controlled by creating the appropriate views that include only the desired attributes and granting the corresponding privileges on the views. However, because updating views is not always possible (see Chapter 8), the UPDATE and INSERT privileges are given the option to specify the particular attributes of a base relation that may be updated.

30.2.6 Specifying Limits on Propagation of Privileges

Techniques to limit the propagation of privileges have been developed, although they have not yet been implemented in most DBMSs and *are not a part* of SQL. Limiting **horizontal propagation** to an integer number i means that an account B given the GRANT OPTION can grant the privilege to at most i other accounts. **Vertical propagation** is more complicated; it limits the depth of the granting of privileges. Granting a privilege with a vertical propagation of zero is equivalent to granting the privilege with *no* GRANT OPTION. If account A grants a privilege to account B with the vertical propagation set to an integer number $j > 0$, this means that the account B has the GRANT OPTION on that privilege, but B can grant the privilege to other accounts only with a vertical propagation *less than j*. In effect, vertical propagation limits the sequence of GRANT OPTIONS that can be given from one account to the next based on a single original grant of the privilege.

We briefly illustrate horizontal and vertical propagation limits—which are *not available* currently in SQL or other relational systems—with an example. Suppose

that A1 grants SELECT to A2 on the EMPLOYEE relation with horizontal propagation equal to 1 and vertical propagation equal to 2. A2 can then grant SELECT to at most one account because the horizontal propagation limitation is set to 1. Additionally, A2 cannot grant the privilege to another account except with vertical propagation set to 0 (no GRANT OPTION) or 1; this is because A2 must reduce the vertical propagation by at least 1 when passing the privilege to others. In addition, the horizontal propagation must be less than or equal to the originally granted horizontal propagation. For example, if account *A* grants a privilege to account *B* with the horizontal propagation set to an integer number $j > 0$, this means that *B* can grant the privilege to other accounts only with a horizontal propagation *less than or equal to j*. As this example shows, horizontal and vertical propagation techniques are designed to limit the depth and breadth of propagation of privileges.

30.3 Mandatory Access Control and Role-Based Access Control for Multilevel Security

The discretionary access control technique of granting and revoking privileges on relations has traditionally been the main security mechanism for relational database systems. This is an all-or-nothing method: A user either has or does not have a certain privilege. In many applications, an *additional security policy* is needed that classifies data and users based on security classes. This approach, known as **mandatory access control (MAC)**, would typically be *combined* with the discretionary access control mechanisms described in Section 30.2. It is important to note that most mainstream RDBMSs currently provide mechanisms only for discretionary access control. However, the need for multilevel security exists in government, military, and intelligence applications, as well as in many industrial and corporate applications. Because of the overriding concerns for privacy, in many systems the levels are determined by who has what access to what private information (also called personally identifiable information). Some DBMS vendors—for example, Oracle—have released special versions of their RDBMSs that incorporate mandatory access control for government use.

Typical **security classes** are top secret (TS), secret (S), confidential (C), and unclassified (U), where TS is the highest level and U the lowest. Other more complex security classification schemes exist, in which the security classes are organized in a lattice. For simplicity, we will use the system with four security classification levels, where $TS \geq S \geq C \geq U$, to illustrate our discussion. The commonly used model for multilevel security, known as the *Bell-LaPadula model*,[4] classifies each **subject** (user, account, program) and **object** (relation, tuple, column, view, operation) into one of the security classifications TS, S, C, or U. We will refer to the **clearance** (classification) of a subject *S* as **class(*S*)** and to the **classification** of an object *O* as **class(*O*)**. Two restrictions are enforced on data access based on the subject/object classifications:

[4]Bell and La Padulla (1976) was a MITRE technical report on secure computer systems in Multics.

1. A subject S is not allowed read access to an object O unless class(S) ≥ class(O). This is known as the **simple security property**.
2. A subject S is not allowed to write an object O unless class(S) ≤ class(O). This is known as the **star property** (or *-property).

The first restriction is intuitive and enforces the obvious rule that no subject can read an object whose security classification is higher than the subject's security clearance. The second restriction is less intuitive. It prohibits a subject from writing an object at a lower security classification than the subject's security clearance. Violation of this rule would allow information to flow from higher to lower classifications, which violates a basic tenet of multilevel security. For example, a user (subject) with TS clearance may make a copy of an object with classification TS and then write it back as a new object with classification U, thus making it visible throughout the system.

To incorporate multilevel security notions into the relational database model, it is common to consider attribute values and tuples as data objects. Hence, each attribute A is associated with a **classification attribute** C in the schema, and each attribute value in a tuple is associated with a corresponding security classification. In addition, in some models, a **tuple classification** attribute TC is added to the relation attributes to provide a classification for each tuple as a whole. The model we describe here is known as the *multilevel model*, because it allows classifications at multiple security levels. A **multilevel relation** schema R with n attributes would be represented as:

$$R(A_1, C_1, A_2, C_2, \ldots, A_n, C_n, TC)$$

where each C_i represents the *classification attribute* associated with attribute A_i.

The value of the tuple classification attribute TC in each tuple t—which is the *highest* of all attribute classification values within t—provides a general classification for the tuple itself. Each attribute classification C_i provides a finer security classification for each attribute value within the tuple. The value of TC in each tuple t is the *highest* of all attribute classification values C_i within t.

The **apparent key** of a multilevel relation is the set of attributes that would have formed the primary key in a regular (single-level) relation. A multilevel relation will appear to contain different data to subjects (users) with different clearance levels. In some cases, it is possible to store a single tuple in the relation at a higher classification level and produce the corresponding tuples at a lower-level classification through a process known as **filtering**. In other cases, it is necessary to store two or more tuples at different classification levels with the same value for the *apparent key*. This leads to the concept of **polyinstantiation**,[5] where several tuples can have the same apparent key value but have different attribute values for users at different clearance levels.

We illustrate these concepts with the simple example of a multilevel relation shown in Figure 30.2(a), where we display the classification attribute values next to each

[5]This is similar to the notion of having multiple versions in the database that represent the same real-world object.

(a) EMPLOYEE

Name	Salary	JobPerformance	TC
Smith U	40000 C	Fair S	S
Brown C	80000 S	Good C	S

(b) EMPLOYEE

Name	Salary	JobPerformance	TC
Smith U	40000 C	NULL C	C
Brown C	NULL C	Good C	C

(c) EMPLOYEE

Name	Salary	JobPerformance	TC
Smith U	NULL U	NULL U	U

(d) EMPLOYEE

Name	Salary	JobPerformance	TC
Smith U	40000 C	Fair S	S
Smith U	40000 C	Excellent C	C
Brown C	80000 S	Good C	S

Figure 30.2
A multilevel relation to illustrate multilevel security. (a) The original EMPLOYEE tuples. (b) Appearance of EMPLOYEE after filtering for classification C users. (c) Appearance of EMPLOYEE after filtering for classification U users. (d) Polyinstantiation of the Smith tuple.

attribute's value. Assume that the Name attribute is the apparent key, and consider the query **SELECT** * **FROM** EMPLOYEE. A user with security clearance *S* would see the same relation shown in Figure 30.2(a), since all tuple classifications are less than or equal to *S*. However, a user with security clearance *C* would not be allowed to see the values for Salary of 'Brown' and Job_performance of 'Smith', since they have higher classification. The tuples would be **filtered** to appear as shown in Figure 30.2(b), with Salary and Job_performance *appearing as null*. For a user with security clearance *U*, the filtering allows only the Name attribute of 'Smith' to appear, with all the other attributes appearing as null (Figure 30.2(c)). Thus, filtering introduces null values for attribute values whose security classification is higher than the user's security clearance.

In general, the **entity integrity** rule for multilevel relations states that all attributes that are members of the apparent key must not be null and must have the *same* security classification within each individual tuple. Additionally, all other attribute values in the tuple must have a security classification greater than or equal to that of the apparent key. This constraint ensures that a user can see the key if the user is permitted to see any part of the tuple. Other integrity rules, called **null integrity** and **interinstance integrity**, informally ensure that if a tuple value at some security level can be filtered (derived) from a higher-classified tuple, then it is sufficient to store the higher-classified tuple in the multilevel relation.

To illustrate polyinstantiation further, suppose that a user with security clearance *C* tries to update the value of Job_performance of 'Smith' in Figure 30.2 to 'Excellent'; this corresponds to the following SQL update being submitted by that user:

```
UPDATE  EMPLOYEE
SET     Job_performance = 'Excellent'
WHERE   Name = 'Smith';
```

Since the view provided to users with security clearance *C* (see Figure 30.2(b)) permits such an update, the system should not reject it; otherwise, the user could *infer* that some nonnull value exists for the Job_performance attribute of 'Smith' rather than the null value that appears. This is an example of inferring information through what is known as a **covert channel**, which should not be permitted in highly secure systems (see Section 30.6.1). However, the user should not be allowed to overwrite the existing value of Job_performance at the higher classification level. The solution is to create a **polyinstantiation** for the 'Smith' tuple at the lower classification level *C*, as shown in Figure 30.2(d). This is necessary since the new tuple cannot be filtered from the existing tuple at classification *S*.

The basic update operations of the relational model (INSERT, DELETE, UPDATE) must be modified to handle this and similar situations, but this aspect of the problem is outside the scope of our presentation. We refer the interested reader to the Selected Bibliography at the end of this chapter for further details.

30.3.1 Comparing Discretionary Access Control and Mandatory Access Control

Discretionary access control (DAC) policies are characterized by a high degree of flexibility, which makes them suitable for a large variety of application domains. The main drawback of DAC models is their vulnerability to malicious attacks, such as Trojan horses embedded in application programs. The reason for this vulnerability is that discretionary authorization models do not impose any control on how information is propagated and used once it has been accessed by users authorized to do so. By contrast, mandatory policies ensure a high degree of protection—in a way, they prevent any illegal flow of information. Therefore, they are suitable for military and high-security types of applications, which require a higher degree of protection. However, mandatory policies have the drawback of being too rigid in that they require a strict classification of subjects and objects into security levels, and therefore they are applicable to few environments and place an additional burden of labeling every object with its security classification. In many practical situations, discretionary policies are preferred because they offer a better tradeoff between security and applicability than mandatory policies.

30.3.2 Role-Based Access Control

Role-based access control (RBAC) emerged rapidly in the 1990s as a proven technology for managing and enforcing security in large-scale enterprise-wide systems.

Its basic notion is that privileges and other permissions are associated with organizational **roles** rather than with individual users. Individual users are then assigned to appropriate roles. Roles can be created using the CREATE ROLE and DESTROY ROLE commands. The GRANT and REVOKE commands discussed in Section 30.2 can then be used to assign and revoke privileges from roles, as well as for individual users when needed. For example, a company may have roles such as sales account manager, purchasing agent, mailroom clerk, customer service manager, and so on. Multiple individuals can be assigned to each role. Security privileges that are common to a role are granted to the role name, and any individual assigned to this role would automatically have those privileges granted.

RBAC can be used with traditional discretionary and mandatory access controls; it ensures that only authorized users in their specified roles are given access to certain data or resources. Users create sessions during which they may activate a subset of roles to which they belong. Each session can be assigned to several roles, but it maps to one user or a single subject only. Many DBMSs have allowed the concept of roles, where privileges can be assigned to roles.

Separation of duties is another important requirement in various mainstream DBMSs. It is needed to prevent one user from doing work that requires the involvement of two or more people, thus preventing collusion. One method in which separation of duties can be successfully implemented is with mutual exclusion of roles. Two roles are said to be **mutually exclusive** if both the roles cannot be used simultaneously by the user. **Mutual exclusion of roles** can be categorized into two types, namely *authorization time exclusion (static)* and *runtime exclusion (dynamic)*. In authorization time exclusion, two roles that have been specified as mutually exclusive cannot be part of a user's authorization at the same time. In runtime exclusion, both these roles can be authorized to one user but cannot be activated by the user at the same time. Another variation in mutual exclusion of roles is that of complete and partial exclusion.

The **role hierarchy** in RBAC is a natural way to organize roles to reflect the organization's lines of authority and responsibility. By convention, junior roles at the bottom are connected to progressively senior roles as one moves up the hierarchy. The hierarchic diagrams are partial orders, so they are reflexive, transitive, and antisymmetric. In other words, if a user has one role, the user automatically has roles lower in the hierarchy. Defining a role hierarchy involves choosing the type of hierarchy and the roles, and then implementing the hierarchy by granting roles to other roles. Role hierarchy can be implemented in the following manner:

```
GRANT ROLE full_time TO employee_type1
GRANT ROLE intern TO employee_type2
```

The above are examples of granting the roles *full_time* and *intern* to two types of employees.

Another issue related to security is *identity management*. **Identity** refers to a unique name of an individual person. Since the legal names of persons are not necessarily unique, the identity of a person must include sufficient additional information to

make the complete name unique. Authorizing this identity and managing the schema of these identities is called **identity management.** Identity management addresses how organizations can effectively authenticate people and manage their access to confidential information. It has become more visible as a business requirement across all industries affecting organizations of all sizes. Identity management administrators constantly need to satisfy application owners while keeping expenditures under control and increasing IT efficiency.

Another important consideration in RBAC systems is the possible temporal constraints that may exist on roles, such as the time and duration of role activations and the timed triggering of a role by an activation of another role. Using an RBAC model is a highly desirable goal for addressing the key security requirements of Web-based applications. Roles can be assigned to workflow tasks so that a user with any of the roles related to a task may be authorized to execute it and may play a certain role only for a certain duration.

RBAC models have several desirable features, such as flexibility, policy neutrality, better support for security management and administration, and a natural enforcement of the hierarchical organization structure within organizations. They also have other aspects that make them attractive candidates for developing secure Web-based applications. These features are lacking in DAC and MAC models. RBAC models do include the capabilities available in traditional DAC and MAC policies. Furthermore, an RBAC model provides mechanisms for addressing the security issues related to the execution of tasks and workflows, and for specifying user-defined and organization-specific policies. Easier deployment over the Internet has been another reason for the success of RBAC models.

30.3.3 Label-Based Security and Row-Level Access Control

Many mainstream RDBMSs currently use the concept of row-level access control, where sophisticated access control rules can be implemented by considering the data row by row. In row-level access control, each data row is given a label, which is used to store information about data sensitivity. Row-level access control provides finer granularity of data security by allowing the permissions to be set for each row and not just for the table or column. Initially the user is given a default session label by the database administrator. Levels correspond to a hierarchy of data-sensitivity levels to exposure or corruption, with the goal of maintaining privacy or security. Labels are used to prevent unauthorized users from viewing or altering certain data. A user having a low authorization level, usually represented by a low number, is denied access to data having a higher-level number. If no such label is given to a row, a row label is automatically assigned to it depending upon the user's session label.

A policy defined by an administrator is called a **label security policy**. Whenever data affected by the policy is accessed or queried through an application, the policy is automatically invoked. When a policy is implemented, a new column is added to each row in the schema. The added column contains the label for each row that

reflects the sensitivity of the row as per the policy. Similar to MAC (mandatory access control), where each user has a security clearance, each user has an identity in label-based security. This user's identity is compared to the label assigned to each row to determine whether the user has access to view the contents of that row. However, the user can write the label value himself, within certain restrictions and guidelines for that specific row. This label can be set to a value that is between the user's current session label and the user's minimum level. The DBA has the privilege to set an initial default row label.

The label security requirements are applied on top of the DAC requirements for each user. Hence, the user must satisfy the DAC requirements and then the label security requirements to access a row. The DAC requirements make sure that the user is legally authorized to carry on that operation on the schema. In most applications, only some of the tables need label-based security. For the majority of the application tables, the protection provided by DAC is sufficient.

Security policies are generally created by managers and human resources personnel. The policies are high-level, technology neutral, and relate to risks. Policies are a result of management instructions to specify organizational procedures, guiding principles, and courses of action that are considered to be expedient, prudent, or advantageous. Policies are typically accompanied by a definition of penalties and countermeasures if the policy is transgressed. These policies are then interpreted and converted to a set of label-oriented policies by the **label security administrator**, who defines the security labels for data and authorizations for users; these labels and authorizations govern access to specified protected objects.

Suppose a user has SELECT privileges on a table. When the user executes a SELECT statement on that table, label security will automatically evaluate each row returned by the query to determine whether the user has rights to view the data. For example, if the user has a sensitivity of 20, then the user can view all rows having a security level of 20 or lower. The level determines the sensitivity of the information contained in a row; the more sensitive the row, the higher its security label value. Such label security can be configured to perform security checks on UPDATE, DELETE, and INSERT statements as well.

30.3.4 XML Access Control

With the worldwide use of XML in commercial and scientific applications, efforts are under way to develop security standards. Among these efforts are digital signatures and encryption standards for XML. The XML Signature Syntax and Processing specification describes an XML syntax for representing the associations between cryptographic signatures and XML documents or other electronic resources. The specification also includes procedures for computing and verifying XML signatures. An XML digital signature differs from other protocols for message signing, such as **OpenPGP** (**Pretty Good Privacy**—a confidentiality and authentication service that can be used for electronic mail and file storage application), in its support for signing only specific portions of the XML tree (see Chapter 13) rather than the

complete document. Additionally, the XML signature specification defines mechanisms for countersigning and transformations—so-called *canonicalization*—to ensure that two instances of the same text produce the same digest for signing even if their representations differ slightly; for example, in typographic white space.

The XML Encryption Syntax and Processing specification defines XML vocabulary and processing rules for protecting confidentiality of XML documents in whole or in part and of non-XML data as well. The encrypted content and additional processing information for the recipient are represented in well-formed XML so that the result can be further processed using XML tools. In contrast to other commonly used technologies for confidentiality, such as SSL (Secure Sockets Layer—a leading Internet security protocol) and virtual private networks, XML encryption also applies to parts of documents and to documents in persistent storage. Database systems such as PostgreSQL or Oracle support JSON (JavaScript Object Notation) objects as a data format and have similar facilities for JSON objects like those defined above for XML.

30.3.5 Access Control Policies for the Web and Mobile Appplications

Publicly accessible Web application environments present a unique challenge to database security. These systems include those responsible for handling sensitive or private information and include social networks, mobile application API servers, and e-commerce transaction platforms.

Electronic commerce (**e-commerce**) environments are characterized by any transactions that are done electronically. They require elaborate access control policies that go beyond traditional DBMSs. In conventional database environments, access control is usually performed using a set of authorizations stated by security officers or users according to some security policies. Such a simple paradigm is not well suited for a dynamic environment like e-commerce. Furthermore, in an e-commerce environment the resources to be protected are not only traditional data but also knowledge and experience. Such peculiarities call for more flexibility in specifying access control policies. The access control mechanism must be flexible enough to support a wide spectrum of heterogeneous protection objects.

Because many reservation, ticketing, payment, and online shopping systems process information that is protected by law, the security architecture that goes beyond simple database access control must be put in place to protect the information. When an unauthorized party inappropriately accesses protected information, it amounts to a data breach, which has significant legal and financial consequences. This unauthorized party could be an adversary that actively seeks to steal protected information or may be an employee who overstepped his or her role or incorrectly distributed protected information to others. Inappropriate handling of credit card data, for instance, has led to significant data breaches at major retailers.

In conventional database environments, access control is usually performed using a set of authorizations stated by security officers. But in Web applications, it is all too

common that the Web application itself is the user rather than a duly authorized individual. This gives rise to a situation where the DBMS's access control mechanisms are bypassed and the database becomes just a relational data store to the system. In such environments, vulnerabilities like SQL injection (which we cover in depth in Section 30.4) become significantly more dangerous bacause it may lead to a total data breach rather than being limited to data that a particular account is authorized to access.

To protect against data breaches in these systems, a first requirement is a comprehensive information security policy that goes beyond the technical access control mechanisms found in mainstream DBMSs. Such a policy must protect not only traditional data, but also processes, knowledge, and experience.

A second related requirement is the support for content-based access control. **Content-based access control** allows one to express access control policies that take the protection object content into account. In order to support content-based access control, access control policies must allow inclusion of conditions based on the object content.

A third requirement is related to the heterogeneity of subjects, which requires access control policies based on user characteristics and qualifications rather than on specific and individual characteristics (for example, user Ids). A possible solution that will allow better accounting of user profiles in the formulation of access control policies, is to support the notion of credentials. A **credential** is a set of properties concerning a user that are relevant for security purposes (for example, age or position or role within an organization). For instance, by using credentials, one can simply formulate policies such as *Only permanent staff with five or more years of service can access documents related to the internals of the system.*

XML is expected to play a key role in access control for e-commerce applications[6] because XML is becoming the common representation language for document interchange over the Web, and is also becoming the language for e-commerce. Thus, on the one hand, there is the need to make XML representations secure by providing access control mechanisms specifically tailored to the protection of XML documents. On the other hand, access control information (that is, access control policies and user credentials) can be expressed using XML itself. The **Directory Services Markup Language (DSML)** is a representation of directory service information in XML syntax. It provides a foundation for a standard for communicating with the directory services that will be responsible for providing and authenticating user credentials. The uniform presentation of both protection objects and access control policies can be applied to policies and credentials themselves. For instance, some credential properties (such as the user name) may be accessible to everyone, whereas other properties may be visible only to a restricted class of users. Additionally, the use of an XML-based language for specifying credentials and access control policies facilitates secure credential submission and export of access control policies.

[6]See Thuraisingham et al. (2001).

30.4 SQL Injection

SQL injection is one of the most common threats to a database system. We will discuss it in detail later in this section. Some of the other frequent attacks on databases are:

- **Unauthorized privilege escalation.** This attack is characterized by an individual attempting to elevate his or her privilege by attacking vulnerable points in the database systems.
- **Privilege abuse.** Whereas unauthorized privilege escalation is done by an unauthorized user, this attack is performed by a privileged user. For example, an administrator who is allowed to change student information can use this privilege to update student grades without the instructor's permission.
- **Denial of service.** A **denial of service (DOS) attack** is an attempt to make resources unavailable to its intended users. It is a general attack category in which access to network applications or data is denied to intended users by overflowing the buffer or consuming resources.
- **Weak authentication.** If the user authentication scheme is weak, an attacker can impersonate the identity of a legitimate user by obtaining her login credentials.

30.4.1 SQL Injection Methods

As we discussed in Chapter 11, Web programs and applications that access a database can send commands and data to the database, as well as display data retrieved from the database through the Web browser. In an **SQL injection attack**, the attacker injects a string input through the application, which changes or manipulates the SQL statement to the attacker's advantage. An SQL injection attack can harm the database in various ways, such as unauthorized manipulation of the database or retrieval of sensitive data. It can also be used to execute system-level commands that may cause the system to deny service to the application. This section describes types of injection attacks.

SQL Manipulation. A manipulation attack, which is the most common type of injection attack, changes an SQL command in the application—for example, by adding conditions to the WHERE-clause of a query, or by expanding a query with additional query components using set operations such as UNION, INTERSECT, or MINUS. Other types of manipulation attacks are also possible. A typical manipulation attack occurs during database login. For example, suppose that a simplistic authentication procedure issues the following query and checks to see if any rows were returned:

```
SELECT * FROM users WHERE username = 'jake' and PASSWORD =
    'jakespasswd' ;
```

The attacker can try to change (or manipulate) the SQL statement by changing it as follows:

```
SELECT * FROM users WHERE username = 'jake' and (PASSWORD =
    'jakespasswd' or 'x' = 'x') ;
```

As a result, the attacker who knows that 'jake' is a valid login of some user is able to log into the database system as 'jake' without knowing his password and is able to do everything that 'jake' may be authorized to do to the database system.

Code Injection. This type of attack attempts to add additional SQL statements or commands to the existing SQL statement by exploiting a computer bug, which is caused by processing invalid data. The attacker can inject or introduce code into a computer program to change the course of execution. Code injection is a popular technique for system hacking or cracking to gain information.

Function Call Injection. In this kind of attack, a database function or operating system function call is inserted into a vulnerable SQL statement to manipulate the data or make a privileged system call. For example, it is possible to exploit a function that performs some aspect related to network communication. In addition, functions that are contained in a customized database package, or any custom database function, can be executed as part of an SQL query. In particular, dynamically created SQL queries (see Chapter 10) can be exploited since they are constructed at runtime.

For example, the *dual* table is used in the FROM clause of SQL in Oracle when a user needs to run SQL that does not logically have a table name. To get today's date, we can use:

```
SELECT SYSDATE FROM dual;
```

The following example demonstrates that even the simplest SQL statements can be vulnerable.

```
SELECT TRANSLATE ('user input', 'from_string', 'to_string') FROM dual;
```

Here, TRANSLATE is used to replace a string of characters with another string of characters. The TRANSLATE function above will replace the characters of the 'from_string' with the characters in the 'to_string' one by one. This means that the *f* will be replaced with the *t*, the *r* with the *o*, the *o* with the _, and so on.

This type of SQL statement can be subjected to a function injection attack. Consider the following example:

```
SELECT TRANSLATE (" || UTL_HTTP.REQUEST ('http://129.107.2.1/') || ",
    '98765432', '9876') FROM dual;
```

The user can input the string (" || UTL_HTTP.REQUEST ('http://129.107.2.1/') ||"), where || is the concatenate operator, thus requesting a page from a Web server. UTL_HTTP makes Hypertext Transfer Protocol (HTTP) callouts from SQL. The REQUEST object takes a URL ('http://129.107.2.1/' in this example) as a parameter, contacts that site, and returns the data (typically HTML) obtained from that site. The attacker could manipulate the string he inputs, as well as the URL, to include other functions and do other illegal operations. We just used a dummy example to show conversion of '98765432' to '9876', but the user's intent would be to access the URL and get sensitive information. The attacker can then retrieve

useful information from the database server—located at the URL that is passed as a parameter—and send it to the Web server (that calls the TRANSLATE function).

30.4.2 Risks Associated with SQL Injection

SQL injection is harmful and the risks associated with it provide motivation for attackers. Some of the risks associated with SQL injection attacks are explained below.

- **Database fingerprinting.** The attacker can determine the type of database being used in the backend so that he can use database-specific attacks that correspond to weaknesses in a particular DBMS.
- **Denial of service.** The attacker can flood the server with requests, thus denying service to valid users, or the attacker can delete some data.
- **Bypassing authentication.** This is one of the most common risks, in which the attacker can gain access to the database as an authorized user and perform all the desired tasks.
- **Identifying injectable parameters.** In this type of attack, the attacker gathers important information about the type and structure of the back-end database of a Web application. This attack is made possible by the fact that the default error page returned by application servers is often overly descriptive.
- **Executing remote commands.** This provides attackers with a tool to execute arbitrary commands on the database. For example, a remote user can execute stored database procedures and functions from a remote SQL interactive interface.
- **Performing privilege escalation.** This type of attack takes advantage of logical flaws within the database to upgrade the access level.

30.4.3 Protection Techniques against SQL Injection

Protection against SQL injection attacks can be achieved by applying certain programming rules to all Web-accessible procedures and functions. This section describes some of these techniques.

Bind Variables (Using Parameterized Statements). The use of bind variables (also known as *parameters*; see Chapter 10) protects against injection attacks and also improves performance.

Consider the following example using Java and JDBC:

```
PreparedStatement stmt = conn.prepareStatement( "SELECT * FROM
    EMPLOYEE WHERE EMPLOYEE_ID=? AND PASSWORD=?");
stmt.setString(1, employee_id);
stmt.setString(2, password);
```

Instead of embedding the user input into the statement, the input should be bound to a parameter. In this example, the input '1' is assigned (bound) to a bind variable

'employee_id' and input '2' to the bind variable 'password' instead of directly passing string parameters.

Filtering Input (Input Validation). This technique can be used to remove escape characters from input strings by using the SQL `Replace` function. For example, the delimiter single quote (') can be replaced by two single quotes (''). Some SQL manipulation attacks can be prevented by using this technique, since escape characters can be used to inject manipulation attacks. However, because there can be a large number of escape characters, this technique is not reliable.

Function Security. Database functions, both standard and custom, should be restricted, as they can be exploited in the SQL function injection attacks.

30.5 Introduction to Statistical Database Security

Statistical databases are used mainly to produce statistics about various populations. The database may contain confidential data about individuals; this information should be protected from user access. However, users are permitted to retrieve statistical information about the populations, such as averages, sums, counts, maximums, minimums, and standard deviations. The techniques that have been developed to protect the privacy of individual information are beyond the scope of this text. We will illustrate the problem with a very simple example, which refers to the relation shown in Figure 30.3. This is a PERSON relation with the attributes Name, Ssn, Income, Address, City, State, Zip, Sex, and Last_degree.

A **population** is a set of tuples of a relation (table) that satisfy some selection condition. Hence, each selection condition on the PERSON relation will specify a particular population of PERSON tuples. For example, the condition Sex = 'M' specifies the male population; the condition ((Sex = 'F') AND (Last_degree = 'M.S.' OR Last_degree = 'Ph.D.')) specifies the female population that has an M.S. or Ph.D. degree as their highest degree; and the condition City = 'Houston' specifies the population that lives in Houston.

Statistical queries involve applying statistical functions to a population of tuples. For example, we may want to retrieve the number of individuals in a population or the average income in the population. However, statistical users are not allowed to retrieve individual data, such as the income of a specific person. **Statistical database security** techniques must prohibit the retrieval of individual data. This can be achieved by prohibiting queries that retrieve attribute values and by allowing

Figure 30.3
The PERSON relation schema for illustrating statistical database security.

PERSON

Name	Ssn	Income	Address	City	State	Zip	Sex	Last_degree

only queries that involve statistical aggregate functions such as COUNT, SUM, MIN, MAX, AVERAGE, and STANDARD DEVIATION. Such queries are sometimes called **statistical queries**.

It is the responsibility of a database management system to ensure the confidentiality of information about individuals while still providing useful statistical summaries of data about those individuals to users. Provision of **privacy protection** of users in a statistical database is paramount; its violation is illustrated in the following example.

In some cases it is possible to **infer** the values of individual tuples from a sequence of statistical queries. This is particularly true when the conditions result in a population consisting of a small number of tuples. As an illustration, consider the following statistical queries:

Q1: **SELECT COUNT** (*)**FROM** PERSON
WHERE <condition>;

Q2: **SELECT AVG** (Income) **FROM** PERSON
WHERE <condition>;

Now suppose that we are interested in finding the Salary of Jane Smith, and we know that she has a Ph.D. degree and that she lives in the city of Bellaire, Texas. We issue the statistical query Q1 with the following condition:

(Last_degree='Ph.D.' AND Sex='F' AND City='Bellaire' AND State='Texas')

If we get a result of 1 for this query, we can issue Q2 with the same condition and find the Salary of Jane Smith. Even if the result of Q1 on the preceding condition is not 1 but is a small number—say 2 or 3—we can issue statistical queries using the functions MAX, MIN, and AVERAGE to identify the possible range of values for the Salary of Jane Smith.

The possibility of inferring individual information from statistical queries is reduced if no statistical queries are permitted whenever the number of tuples in the population specified by the selection condition falls below some threshold. Another technique for prohibiting retrieval of individual information is to prohibit sequences of queries that refer repeatedly to the same population of tuples. It is also possible to introduce slight inaccuracies or *noise* into the results of statistical queries deliberately, to make it difficult to deduce individual information from the results. Another technique is partitioning of the database. Partitioning implies that records are stored in groups of some minimum size; queries can refer to any complete group or set of groups, but never to subsets of records within a group. The interested reader is referred to the bibliography at the end of this chapter for a discussion of these techniques.

30.6 Introduction to Flow Control

Flow control regulates the distribution or flow of information among accessible objects. A flow between object *X* and object *Y* occurs when a program reads values from *X* and writes values into *Y*. **Flow controls** check that information contained in some objects does not flow explicitly or implicitly into less protected objects. Thus, a

user cannot get indirectly in Y what he or she cannot get directly in X. Active flow control began in the early 1970s. Most flow controls employ some concept of security class; the transfer of information from a sender to a receiver is allowed only if the receiver's security class is at least as privileged as the sender's. Examples of a flow control include preventing a service program from leaking a customer's confidential data, and blocking the transmission of secret military data to an unknown classified user.

A **flow policy** specifies the channels along which information is allowed to move. The simplest flow policy specifies just two classes of information—confidential (C) and nonconfidential (N)—and allows all flows except those from class C to class N. This policy can solve the confinement problem that arises when a service program handles data such as customer information, some of which may be confidential. For example, an income-tax-computing service might be allowed to retain a customer's address and the bill for services rendered, but not a customer's income or deductions.

Access control mechanisms are responsible for checking users' authorizations for resource access: Only granted operations are executed. Flow controls can be enforced by an extended access control mechanism, which involves assigning a security class (usually called the *clearance*) to each running program. The program is allowed to read a particular memory segment only if its security class is as high as that of the segment. It is allowed to write in a segment only if its class is as low as that of the segment. This automatically ensures that no information transmitted by the person can move from a higher to a lower class. For example, a military program with a secret clearance can only read from objects that are unclassified and confidential and can only write into objects that are secret or top secret.

Two types of flow can be distinguished: *explicit flows*, which occur as a consequence of assignment instructions, such as $Y := f(X_1, X_n)$; and *implicit flows*, which are generated by conditional instructions, such as if $f(X_{m+1}, \ldots, X_n)$ then $Y := f(X_1, X_m)$.

Flow control mechanisms must verify that only authorized flows, both explicit and implicit, are executed. A set of rules must be satisfied to ensure secure information flows. Rules can be expressed using flow relations among classes and assigned to information, stating the authorized flows within a system. (An information flow from A to B occurs when information associated with A affects the value of information associated with B. The flow results from operations that cause information transfer from one object to another.) These relations can define, for a class, the set of classes where information (classified in that class) can flow, or can state the specific relations to be verified between two classes to allow information to flow from one to the other. In general, flow control mechanisms implement the controls by assigning a label to each object and by specifying the security class of the object. Labels are then used to verify the flow relations defined in the model.

30.6.1 Covert Channels

A covert channel allows a transfer of information that violates the security or the policy. Specifically, a **covert channel** allows information to pass from a higher classification level to a lower classification level through improper means. Covert

channels can be classified into two broad categories: timing channels and storage. The distinguishing feature between the two is that in a **timing channel** the information is conveyed by the timing of events or processes, whereas **storage channels** do not require any temporal synchronization, in that information is conveyed by accessing system information or what is otherwise inaccessible to the user.

In a simple example of a covert channel, consider a distributed database system in which two nodes have user security levels of secret (S) and unclassified (U). In order for a transaction to commit, both nodes must agree to commit. They mutually can only do operations that are consistent with the *-property, which states that in any transaction, the *S* site cannot write or pass information to the *U* site. However, if these two sites collude to set up a covert channel between them, a transaction involving secret data may be committed unconditionally by the *U* site, but the *S* site may do so in some predefined agreed-upon way so that certain information may be passed from the *S* site to the *U* site, violating the *-property. This may be achieved where the transaction runs repeatedly, but the actions taken by the *S* site implicitly convey information to the *U* site. Measures such as locking, which we discussed in Chapters 21 and 22, prevent concurrent writing of the information by users with different security levels into the same objects, preventing the storage-type covert channels. Operating systems and distributed databases provide control over the multiprogramming of operations, which allows a sharing of resources without the possibility of encroachment of one program or process into another's memory or other resources in the system, thus preventing timing-oriented covert channels. In general, covert channels are not a major problem in well-implemented robust database implementations. However, certain schemes may be contrived by clever users that implicitly transfer information.

Some security experts believe that one way to avoid covert channels is to disallow programmers to actually gain access to sensitive data that a program will process after the program has been put into operation. For example, a programmer for a bank has no need to access the names or balances in depositors' accounts. Programmers for brokerage firms do not need to know what buy and sell orders exist for clients. During program testing, access to a form of real data or some sample test data may be justifiable, but not after the program has been accepted for regular use.

30.7 Encryption and Public Key Infrastructures

The previous methods of access and flow control, despite being strong control measures, may not be able to protect databases from some threats. Suppose we communicate data, but our data falls into the hands of a nonlegitimate user. In this situation, by using encryption we can disguise the message so that even if the transmission is diverted, the message will not be revealed. **Encryption** is the conversion of data into a form, called a **ciphertext**, that cannot be easily understood by unauthorized persons. It enhances security and privacy when access controls are bypassed, because in cases of data loss or theft, encrypted data cannot be easily understood by unauthorized persons.

With this background, we adhere to following standard definitions:[7]

- *Ciphertext:* Encrypted (enciphered) data
- *Plaintext (or cleartext):* Intelligible data that has meaning and can be read or acted upon without the application of decryption
- *Encryption:* The process of transforming plaintext into ciphertext
- *Decryption:* The process of transforming ciphertext back into plaintext

Encryption consists of applying an **encryption algorithm** to data using some prespecified **encryption key**. The resulting data must be **decrypted** using a **decryption key** to recover the original data.

30.7.1 The Data Encryption and Advanced Encryption Standards

The **Data Encryption Standard (DES)** is a system developed by the U.S. government for use by the general public. It has been widely accepted as a cryptographic standard both in the United States and abroad. DES can provide end-to-end encryption on the channel between sender *A* and receiver *B*. The DES algorithm is a careful and complex combination of two of the fundamental building blocks of encryption: substitution and permutation (transposition). The algorithm derives its strength from repeated application of these two techniques for a total of 16 cycles. Plaintext (the original form of the message) is encrypted as blocks of 64 bits. Although the key is 64 bits long, in effect the key can be any 56-bit number. After questioning the adequacy of DES, the NIST introduced the **Advanced Encryption Standard (AES)**. This algorithm has a block size of 128 bits, compared with DES's 56-block size, and can use keys of 128, 192, or 256 bits, compared with DES's 56-bit key. AES introduces more possible keys, compared with DES, and thus takes a much longer time to crack. In present systems, AES is the default with large key lengths. It is also the standard for full drive encryption products, with both Apple FileVault and Microsoft BitLocker using 256-bit or 128-bit keys. TripleDES is a fallback option if a legacy system cannot use a modern encryption standard.

30.7.2 Symmetric Key Algorithms

A symmetric key is one key that is used for both encryption and decryption. By using a symmetric key, fast encryption and decryption is possible for routine use with sensitive data in the database. A message encrypted with a secret key can be decrypted only with the same secret key. Algorithms used for symmetric key encryption are called **secret key algorithms**. Since secret-key algorithms are mostly used for encrypting the content of a message, they are also called **content-encryption algorithms**.

[7]U.S. Department of Commerce.

The major liability associated with secret-key algorithms is the need for sharing the secret key. A possible method is to derive the secret key from a user-supplied password string by applying the same function to the string at both the sender and receiver; this is known as a *password-based encryption algorithm*. The strength of the symmetric key encryption depends on the size of the key used. For the same algorithm, encrypting using a longer key is tougher to break than the one using a shorter key.

30.7.3 Public (Asymmetric) Key Encryption

In 1976, Diffie and Hellman proposed a new kind of cryptosystem, which they called **public key encryption**. Public key algorithms are based on mathematical functions rather than operations on bit patterns. They address one drawback of symmetric key encryption, namely that both sender and recipient must exchange the common key in a secure manner. In public key systems, two keys are used for encryption/decryption. The *public key* can be transmitted in a nonsecure way, whereas the *private key* is not transmitted at all. These algorithms—which use two related keys, a public key and a private key, to perform complementary operations (encryption and decryption)—are known as **asymmetric key encryption algorithms**. The use of two keys can have profound consequences in the areas of confidentiality, key distribution, and authentication. The two keys used for public key encryption are referred to as the **public key** and the **private key**. The private key is kept secret, but it is referred to as a *private key* rather than a *secret key* (the key used in conventional encryption) to avoid confusion with conventional encryption. The two keys are mathematically related, since one of the keys is used to perform encryption and the other to perform decryption. However, it is very difficult to derive the private key from the public key.

A public key encryption scheme, or *infrastructure*, has six ingredients:

1. **Plaintext.** This is the data or readable message that is fed into the algorithm as input.
2. **Encryption algorithm.** This algorithm performs various transformations on the plaintext.
3. and 4. **Public and private keys.** These are a pair of keys that have been selected so that if one is used for encryption, the other is used for decryption. The exact transformations performed by the encryption algorithm depend on the public or private key that is provided as input. For example, if a message is encrypted using the public key, it can only be decrypted using the private key.
5. **Ciphertext.** This is the scrambled message produced as output. It depends on the plaintext and the key. For a given message, two different keys will produce two different ciphertexts.
6. **Decryption algorithm.** This algorithm accepts the ciphertext and the matching key and produces the original plaintext.

As the name suggests, the public key of the pair is made public for others to use, whereas the private key is known only to its owner. A general-purpose public key

cryptographic algorithm relies on one key for encryption and a different but related key for decryption. The essential steps are as follows:

1. Each user generates a pair of keys to be used for the encryption and decryption of messages.
2. Each user places one of the two keys in a public register or other accessible file. This is the public key. The companion key is kept private.
3. If a sender wishes to send a private message to a receiver, the sender encrypts the message using the receiver's public key.
4. When the receiver receives the message, he or she decrypts it using the receiver's private key. No other recipient can decrypt the message because only the receiver knows his or her private key.

The RSA Public Key Encryption Algorithm. One of the first public key schemes was introduced in 1978 by Ron Rivest, Adi Shamir, and Len Adleman at MIT[8] and is named after them as the **RSA scheme**. The RSA scheme has since then reigned supreme as the most widely accepted and implemented approach to public key encryption. The RSA encryption algorithm incorporates results from number theory, combined with the difficulty of determining the prime factors of a target. The RSA algorithm also operates with modular arithmetic—mod n.

Two keys, d and e, are used for decryption and encryption. An important property is that they can be interchanged. n is chosen as a large integer that is a product of two large distinct prime numbers, a and b, $n = a \times b$. The encryption key e is a randomly chosen number between 1 and n that is relatively prime to $(a - 1) \times (b - 1)$. The plaintext block P is encrypted as P^e where $P^e = P \bmod n$. Because the exponentiation is performed mod n, factoring P^e to uncover the encrypted plaintext is difficult. However, the decrypting key d is carefully chosen so that $(P^e)d \bmod n = P$. The decryption key d can be computed from the condition that $d \times e = 1 \bmod ((a - 1) \times (b - 1))$. Thus, the legitimate receiver who knows d simply computes $(P^e)d \bmod n = P$ and recovers P without having to factor P^e.

30.7.4 Digital Signatures

A digital signature is an example of using encryption techniques to provide authentication services in electronic commerce applications. Like a handwritten signature, a **digital signature** is a means of associating a mark unique to an individual with a body of text. The mark should be unforgettable, meaning that others should be able to check that the signature comes from the originator.

A digital signature consists of a string of symbols. If a person's digital signature were always the same for each message, then one could easily counterfeit it by simply copying the string of symbols. Thus, signatures must be different for each use. This can be achieved by making each digital signature a function of the message

[8]Rivest et al. (1978).

that it is signing, together with a timestamp. To be unique to each signer and counterfeitproof, each digital signature must also depend on some secret number that is unique to the signer. Thus, in general, a counterfeitproof digital signature must depend on the message and a unique secret number of the signer. The verifier of the signature, however, should not need to know any secret number. Public key techniques are the best means of creating digital signatures with these properties.

30.7.5 Digital Certificates

A digital certificate is used to combine the value of a public key with the identity of the person or service that holds the corresponding private key into a digitally signed statement. Certificates are issued and signed by a certification authority (CA). The entity receiving this certificate from a CA is the subject of that certificate. Instead of requiring each participant in an application to authenticate every user, third-party authentication relies on the use of digital certificates.

The digital certificate itself contains various types of information. For example, both the certification authority and the certificate owner information are included. The following list describes all the information included in the certificate:

1. The certificate owner information, which is represented by a unique identifier known as the distinguished name (DN) of the owner. This includes the owner's name, as well as the owner's organization and other information about the owner.
2. The certificate also includes the public key of the owner.
3. The date of issue of the certificate is also included.
4. The validity period is specified by 'Valid From' and 'Valid To' dates, which are included in each certificate.
5. Issuer identifier information is included in the certificate.
6. Finally, the digital signature of the issuing CA for the certificate is included. All the information listed is encoded through a message-digest function, which creates the digital signature. The digital signature basically certifies that the association between the certificate owner and public key is valid.

30.8 Privacy Issues and Preservation

Preserving data privacy is a growing challenge for database security and privacy experts. In some perspectives, to preserve data privacy we should even limit performing large-scale data mining and analysis. The most commonly used techniques to address this concern are to avoid building mammoth central warehouses as a single repository of vital information. This is one of the stumbling blocks for creating nationwide registries of patients for many important diseases. Another possible measure is to intentionally modify or perturb data.

If all data were available at a single warehouse, violating only a single repository's security could expose all data. Avoiding central warehouses and using distributed

data mining algorithms minimizes the exchange of data needed to develop globally valid models. By modifying, perturbing, and anonymizing data, we can also mitigate privacy risks associated with data mining. This can be done by removing identity information from the released data and injecting noise into the data. However, by using these techniques, we should pay attention to the quality of the resulting data in the database, which may undergo too many modifications. We must be able to estimate the errors that may be introduced by these modifications.

Privacy is an important area of ongoing research in database management. It is complicated due to its multidisciplinary nature and the issues related to the subjectivity in the interpretation of privacy, trust, and so on. As an example, consider medical and legal records and transactions, which must maintain certain privacy requirements. Providing access control and privacy for mobile devices is also receiving increased attention. DBMSs need robust techniques for efficient storage of security-relevant information on small devices, as well as trust negotiation techniques. Where to keep information related to user identities, profiles, credentials, and permissions and how to use it for reliable user identification remains an important problem. Because large-sized streams of data are generated in such environments, efficient techniques for access control must be devised and integrated with processing techniques for continuous queries. Finally, the privacy of user location data, acquired from sensors and communication networks, must be ensured.

30.9 Challenges to Maintaining Database Security

Considering the vast growth in volume and speed of threats to databases and information assets, research efforts need to be devoted to a number of issues: data quality, intellectual property rights, and database survivability, to name a few. We briefly outline the work required in a few important areas that researchers in database security are trying to address.

30.9.1 Data Quality

The database community needs techniques and organizational solutions to assess and attest to the quality of data. These techniques may include simple mechanisms such as quality stamps that are posted on Web sites. We also need techniques that provide more effective integrity semantics verification and tools for the assessment of data quality, based on techniques such as record linkage. Application-level recovery techniques are also needed for automatically repairing incorrect data. The ETL (extract, transform, load) tools widely used to load data in data warehouses (see Section 29.4) are presently grappling with these issues.

30.9.2 Intellectual Property Rights

With the widespread use of the Internet and intranets, legal and informational aspects of data are becoming major concerns for organizations. To address these

concerns, watermarking techniques for relational data have been proposed. The main purpose of digital watermarking is to protect content from unauthorized duplication and distribution by enabling provable ownership of the content. Digital watermarking has traditionally relied upon the availability of a large noise domain within which the object can be altered while retaining its essential properties. However, research is needed to assess the robustness of such techniques and to investigate different approaches aimed at preventing intellectual property rights violations.

30.9.3 Database Survivability

Database systems need to operate and continue their functions, even with reduced capabilities, despite disruptive events such as information warfare attacks. A DBMS, in addition to making every effort to prevent an attack and detecting one in the event of occurrence, should be able to do the following:

- **Confinement.** Take immediate action to eliminate the attacker's access to the system and to isolate or contain the problem to prevent further spread.
- **Damage assessment.** Determine the extent of the problem, including failed functions and corrupted data.
- **Reconfiguration.** Reconfigure to allow operation to continue in a degraded mode while recovery proceeds.
- **Repair.** Recover corrupted or lost data and repair or reinstall failed system functions to reestablish a normal level of operation.
- **Fault treatment.** To the extent possible, identify the weaknesses exploited in the attack and take steps to prevent a recurrence.

The goal of the information warfare attacker is to damage the organization's operation and fulfillment of its mission through disruption of its information systems. The specific target of an attack may be the system itself or its data. Although attacks that bring the system down outright are severe and dramatic, they must also be well timed to achieve the attacker's goal, since attacks will receive immediate and concentrated attention in order to bring the system back to operational condition, diagnose how the attack took place, and install preventive measures.

To date, issues related to database survivability have not been sufficiently investigated. Much more research needs to be devoted to techniques and methodologies that ensure database system survivability.

30.10 Oracle Label-Based Security

Restricting access to entire tables or isolating sensitive data into separate databases is a costly operation to administer. **Oracle label security** overcomes the need for such measures by enabling row-level access control. It is available starting with Oracle Database 11g Release 1 (11.1) Enterprise Edition. Each database table or view has a security policy associated with it. This policy executes every time the table or view is queried or altered. Developers can readily add label-based access

control to their Oracle Database applications. Label-based security provides an adaptable way of controlling access to sensitive data. Both users and data have labels associated with them. Oracle label security uses these labels to provide security.

30.10.1 Virtual Private Database (VPD) Technology

Virtual private databases (VPDs) are a feature of the Oracle Enterprise Edition that add predicates to user statements to limit their access in a transparent manner to the user and the application. The VPD concept allows server-enforced, fine-grained access control for a secure application.

VPD provides access control based on policies. These VPD policies enforce object-level access control or row-level security. VPD provides an application programming interface (API) that allows security policies to be attached to database tables or views. Using PL/SQL, a host programming language used in Oracle applications, developers and security administrators can implement security policies with the help of stored procedures.[9] VPD policies allow developers to remove access security mechanisms from applications and centralize them within the Oracle Database.

VPD is enabled by associating a security "policy" with a table, view, or synonym. An administrator uses the supplied PL/SQL package, DBMS_RLS, to bind a policy function with a database object. When an object having a security policy associated with it is accessed, the function implementing this policy is consulted. The policy function returns a predicate (a WHERE clause) that is then appended to the user's SQL statement, thus *transparently* and *dynamically* modifying the user's data access. Oracle label security is a technique of enforcing row-level security in the form of a security policy.

30.10.2 Label Security Architecture

Oracle label security is built on the VPD technology delivered in the Oracle Database 11.1 Enterprise Edition. Figure 30.4 illustrates how data is accessed under Oracle label security, showing the sequence of DAC and label security checks.

Figure 30.4 shows the sequence of discretionary access control (DAC) and label security checks. The left part of the figure shows an application user in an Oracle Database 11g Release 1 (11.1) session sending out an SQL request. The Oracle DBMS checks the DAC privileges of the user, making sure that he or she has SELECT privileges on the table. Then it checks whether the table has a virtual private database (VPD) policy associated with it to determine if the table is protected using Oracle label security. If it is, the VPD SQL modification (WHERE clause) is added to the original SQL statement to find the set of accessible rows for the user to view. Then Oracle label security checks the labels on each row to determine the subset of rows to which the user has access (as explained in the next section). This modified query is processed, optimized, and executed.

[9]Stored procedures are discussed in Section 8.2.2.

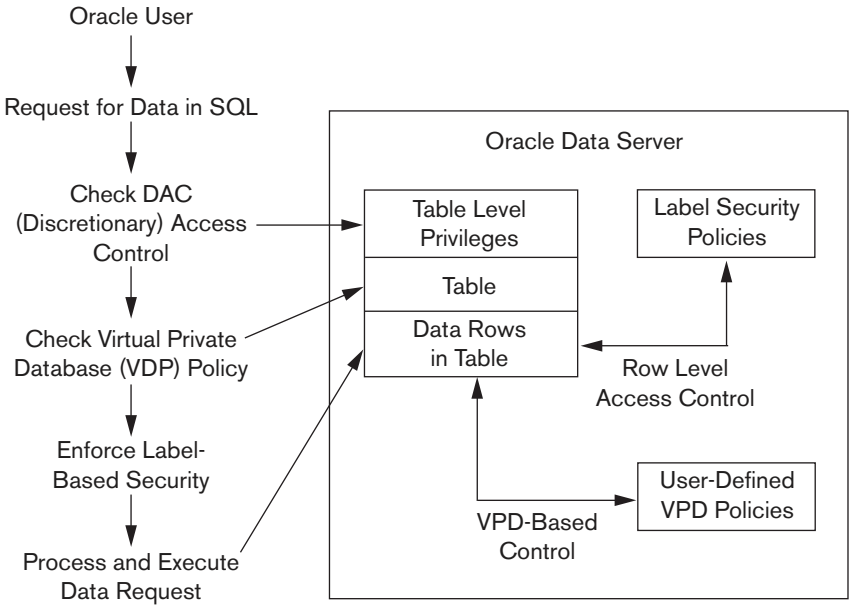

Figure 30.4
Oracle label security architecture.
Data from: Oracle (2007)

30.10.3 How Data Labels and User Labels Work Together

A user's label indicates the information the user is permitted to access. It also determines the type of access (read or write) that the user has on that information. A row's label shows the sensitivity of the information that the row contains as well as the ownership of the information. When a table in the database has a label-based access associated with it, a row can be accessed only if the user's label meets certain criteria defined in the policy definitions. Access is granted or denied based on the result of comparing the data label and the session label of the user.

Compartments allow a finer classification of sensitivity of the labeled data. All data related to the same project can be labeled with the same compartment. Compartments are optional; a label can contain zero or more compartments.

Groups are used to identify organizations as owners of the data with corresponding group labels. Groups are hierarchical; for example, a group can be associated with a parent group.

If a user has a maximum level of SENSITIVE, then the user potentially has access to all data having levels SENSITIVE, CONFIDENTIAL, and UNCLASSIFIED. This user has no access to HIGHLY_SENSITIVE data. Figure 30.5 shows how data labels and user labels work together to provide access control in Oracle label security.

As shown in Figure 30.5, User 1 can access the rows 2, 3, and 4 because his maximum level is HS (Highly_Sensitive). He has access to the FIN (Finance) compartment, and his access to group WR (Western Region) hierarchically includes group

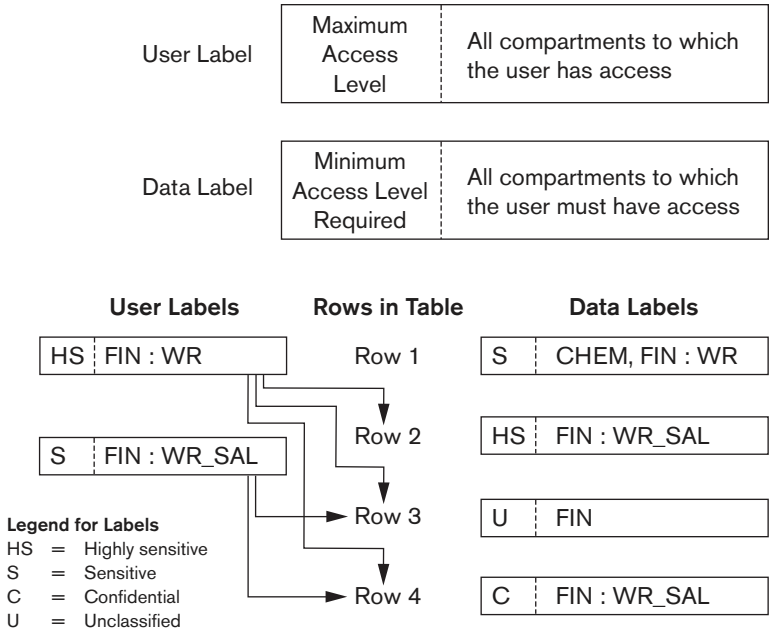

Figure 30.5
Data labels and user labels in Oracle.
Data from: Oracle (2007)

WR_SAL (WR Sales). He cannot access row 1 because he does not have the CHEM (Chemical) compartment. It is important that a user has authorization for all compartments in a row's data label so the user can access that row. Based on this example, user 2 can access both rows 3 and 4 and has a maximum level of S, which is less than the HS in row 2. So, although user 2 has access to the FIN compartment, he can only access the group WR_SAL and thus cannot access row 1.

30.11 Summary

In this chapter, we discussed several techniques for enforcing database system security. Section 30.1 is an introduction to database security. We presented in Section 30.1.1 different threats to databases in terms of loss of integrity, availability, and confidentiality. We discussed in Section 30.1.2 the types of control measures to deal with these problems: access control, inference control, flow control, and encryption. In the rest of Section 30.1, we covered various issues related to security, including data sensitivity and type of disclosures; security versus precision of results when a user requests information; and the relationship between information security and privacy.

Security enforcement deals with controlling access to the database system as a whole and controlling authorization to access specific portions of a database. The former is usually done by assigning accounts with passwords to users. The latter can be accomplished by using a system of granting and revoking privileges to individual accounts for accessing specific parts of the database. This approach,

presented in Section 30.2, is generally referred to as discretionary access control (DAC). We presented some SQL commands for granting and revoking privileges, and we illustrated their use with examples. Then in Section 30.3 we gave an overview of mandatory access control (MAC) mechanisms that enforce multilevel security. These require the classifications of users and data values into security classes and enforce the rules that prohibit flow of information from higher to lower security levels. Some of the key concepts underlying the multilevel relational model, including filtering and polyinstantiation, were presented. Role-based access control (RBAC) was introduced in Section 30.3.2, which assigns privileges based on roles that users play. We introduced the notion of role hierarchies, mutual exclusion of roles, and row- and label-based security. We explained the main ideas behind the threat of SQL injection in Section 30.4, the methods in which it can be induced, and the various types of risks associated with it. Then we gave an idea of the various ways SQL injection can be prevented.

We briefly discussed in Section 30.5 the problem of controlling access to statistical databases to protect the privacy of individual information while concurrently providing statistical access to populations of records. The issues related to flow control and the problems associated with covert channels were discussed next in Section 30.6, as well as encryption and public-versus-private key-based infrastructures in Section 30.7. The idea of symmetric key algorithms and the use of the popular asymmetric key-based public key infrastructure (PKI) scheme was explained in Section 30.7.3. We also covered in Sections 30.7.4 and 30.7.5 the concepts of digital signatures and digital certificates. We highlighted in Section 30.8 the importance of privacy issues and hinted at some privacy preservation techniques. We discussed in Section 30.9 a variety of challenges to security, including data quality, intellectual property rights, and data survivability. We ended the chapter in Section 30.10 by introducing the implementation of security policies by using a combination of label-based security and virtual private databases in Oracle 11g.

Review Questions

30.1. Discuss what is meant by each of the following terms: *database authorization*, *access control*, *data encryption*, *privileged (system) account*, *database audit*, *audit trail*.

30.2. Which account is designated as the owner of a relation? What privileges does the owner of a relation have?

30.3. How is the view mechanism used as an authorization mechanism?

30.4. Discuss the types of privileges at the account level and those at the relation level.

30.5. What is meant by granting a privilege? What is meant by revoking a privilege?

30.6. Discuss the system of propagation of privileges and the restraints imposed by horizontal and vertical propagation limits.

30.7. List the types of privileges available in SQL.

30.8. What is the difference between *discretionary* and *mandatory* access control?

30.9. What are the typical security classifications? Discuss the simple security property and the *-property, and explain the justification behind these rules for enforcing multilevel security.

30.10. Describe the multilevel relational data model. Define the following terms: *apparent key*, *polyinstantiation*, *filtering*.

30.11. What are the relative merits of using DAC or MAC?

30.12. What is role-based access control? In what ways is it superior to DAC and MAC?

30.13. What are the two types of mutual exclusion in role-based access control?

30.14. What is meant by row-level access control?

30.15. What is label security? How does an administrator enforce it?

30.16. What are the different types of SQL injection attacks?

30.17. What risks are associated with SQL injection attacks?

30.18. What preventive measures are possible against SQL injection attacks?

30.19. What is a statistical database? Discuss the problem of statistical database security.

30.20. How is privacy related to statistical database security? What measures can be taken to ensure some degree of privacy in statistical databases?

30.21. What is flow control as a security measure? What types of flow control exist?

30.22. What are covert channels? Give an example of a covert channel.

30.23. What is the goal of encryption? What process is involved in encrypting data and then recovering it at the other end?

30.24. Give an example of an encryption algorithm and explain how it works.

30.25. Repeat the previous question for the popular RSA algorithm.

30.26. What is a symmetric key algorithm for key-based security?

30.27. What is the public key infrastructure scheme? How does it provide security?

30.28. What are digital signatures? How do they work?

30.29. What type of information does a digital certificate include?

Exercises

30.30. How can privacy of data be preserved in a database?

30.31. What are some of the current outstanding challenges for database security?

30.32. Consider the relational database schema in Figure 5.5. Suppose that all the relations were created by (and hence are owned by) user *X*, who wants to grant the following privileges to user accounts *A*, *B*, *C*, *D*, and *E*:

a. Account *A* can retrieve or modify any relation except DEPENDENT and can grant any of these privileges to other users.

b. Account *B* can retrieve all the attributes of EMPLOYEE and DEPARTMENT except for Salary, Mgr_ssn, and Mgr_start_date.

c. Account *C* can retrieve or modify WORKS_ON but can only retrieve the Fname, Minit, Lname, and Ssn attributes of EMPLOYEE and the Pname and Pnumber attributes of PROJECT.

d. Account *D* can retrieve any attribute of EMPLOYEE or DEPENDENT and can modify DEPENDENT.

e. Account *E* can retrieve any attribute of EMPLOYEE but only for EMPLOYEE tuples that have Dno = 3.

f. Write SQL statements to grant these privileges. Use views where appropriate.

30.33. Suppose that privilege (a) of Exercise 30.32 is to be given with GRANT OPTION but only so that account *A* can grant it to at most five accounts, and each of these accounts can propagate the privilege to other accounts but *without* the GRANT OPTION privilege. What would the horizontal and vertical propagation limits be in this case?

30.34. Consider the relation shown in Figure 30.2(d). How would it appear to a user with classification *U*? Suppose that a classification *U* user tries to update the salary of 'Smith' to $50,000; what would be the result of this action?

Selected Bibliography

Authorization based on granting and revoking privileges was proposed for the SYSTEM R experimental DBMS and is presented in Griffiths and Wade (1976). Several books discuss security in databases and computer systems in general, including the books by Leiss (1982a), Fernandez et al. (1981), and Fugini et al. (1995). Natan (2005) is a practical book on security and auditing implementation issues in all major RDBMSs.

Many papers discuss different techniques for the design and protection of statistical databases. They include McLeish (1989), Chin and Ozsoyoglu (1981), Leiss (1982), Wong (1984), and Denning (1980). Ghosh (1984) discusses the use of statistical databases for quality control. There are also many papers discussing cryptography and data encryption, including Diffie and Hellman (1979), Rivest et al. (1978), Akl (1983), Pfleeger and Pfleeger (2007), Omura et al. (1990), Stallings (2000), and Iyer at al. (2004).

Halfond et al. (2006) helps us understand the concepts of SQL injection attacks and the various threats imposed by them. The white paper Oracle (2007a) explains how Oracle is less prone to SQL injection attack as compared to SQL Server. Oracle

(2007a) also gives a brief explanation of how these attacks can be prevented from occurring. Further proposed frameworks are discussed in Boyd and Keromytis (2004), Halfond and Orso (2005), and McClure and Krüger (2005).

Multilevel security is discussed in Jajodia and Sandhu (1991), Denning et al. (1987), Smith and Winslett (1992), Stachour and Thuraisingham (1990), Lunt et al. (1990), and Bertino et al. (2001). Overviews of research issues in database security are given by Lunt and Fernandez (1990), Jajodia and Sandhu (1991), Bertino (1998), Castano et al. (1995), and Thuraisingham et al. (2001). The effects of multilevel security on concurrency control are discussed in Atluri et al. (1997). Security in next-generation, semantic, and object-oriented databases is discussed in Rabbiti et al. (1991), Jajodia and Kogan (1990), and Smith (1990). Oh (1999) presents a model for both discretionary and mandatory security. Security models for Web-based applications and role-based access control are discussed in Joshi et al. (2001). Security issues for managers in the context of e-commerce applications and the need for risk assessment models for selection of appropriate security control measures are discussed in Farahmand et al. (2005). Row-level access control is explained in detail in Oracle (2007b) and Sybase (2005). The latter also provides details on role hierarchy and mutual exclusion. Oracle (2009) explains how Oracle uses the concept of identity management.

Recent advances as well as future challenges for security and privacy of databases are discussed in Bertino and Sandhu (2005). U.S. Govt. (1978), OECD (1980), and NRC (2003) are good references on the view of privacy by important government bodies. Karat et al. (2009) discusses a policy framework for security and privacy. XML and access control are discussed in Naedele (2003). More details are presented on privacy-preserving techniques in Vaidya and Clifton (2004), intellectual property rights in Sion et al. (2004), and database survivability in Jajodia et al. (1999). Oracle's VPD technology and label-based security is discussed in more detail in Oracle (2007b).

Agrawal et al. (2002) defined the concept of Hippocratic Databases for preserving privacy in healthcare information. K-anonymity as a privacy preserving technique is discussed in Bayardo and Agrawal (2005) and in Ciriani et al. (2007). Privacy-preserving data mining techniques based on k-anonymity are surveyed by Ciriani et al. (2008). Vimercati et al. (2014) discuss encryption and fragmentation as potential protection techniques for data confidentiality in the cloud.