Application Security

This chapter presents the following:

- Various types of software controls and implementation
- Database concepts and security issues
- Data warehousing and data mining
- Software life-cycle development processes
- Change control concepts
- Object-oriented programming components
- Expert systems and artificial intelligence

Applications and computer systems are usually developed for functionality first, not security first. To get the best of both worlds, security and functionality would have to be designed and developed at the same time. Security should be interwoven into the core of a product and provide protection at different layers. This is a better approach than trying to develop a front end or wrapper that may reduce the overall functionality and leave security holes when the product has to be integrated with other applications.

Software’s Importance

Application system controls come in various flavors with many different goals. They can control input, processing, number-crunching methods, interprocess communication, access, output, and interfacing to the system and other programs. They should be developed with potential risks in mind, and many types of threat models and risk analyses should be invoked at different stages of development. The goal is to prevent security compromises and to reduce vulnerabilities and the possibility of data corruption. The controls can be preventive, detective, or corrective. They can come in the form of administrative and physical controls, but are usually more technical in this context.

The specific application controls depend upon the application itself, its objectives, the security goals of the application security policy, the type of data and processing it is to carry out, and the environment the application will be placed in. If an application is purely proprietary and will run only in closed trusted environments, fewer security controls may be needed than those required for applications that will connect businesses over the Internet and provide financial transactions. The trick is to understand the security needs of an application, implement the right controls and mechanisms, thoroughly test the mechanisms and how they integrate into the application, follow
structured development methodologies, and provide secure and reliable distribution methods. Seems easy as 1-2-3, right? Nope, the development of a secure application or operating system is very complex and should only be attempted if you have a never-ending supply of coffee, are mentally and physically stable, and have no social life. (This is why we don’t have many secure applications.)

### Where Do We Place the Security?

*“I put mine in my shoe.”*

Today, many security efforts look to solve security problems through controls such as firewalls, intrusion detection systems (IDSs), sensors, content filtering, antivirus software, vulnerability scanners, and much more. This reliance on a long laundry list of controls occurs mainly because our software contains many vulnerabilities. Our environments are commonly referred to as hard and crunchy on the outside and soft and chewy on the inside. This means our perimeter security is fortified and solid, but our internal environment and software are easy to exploit once access has been obtained.

In reality, the flaws within the software cause a majority of the vulnerabilities in the first place. Several reasons explain why perimeter devices are more often considered than software development for security:

- In the past, it was not crucial to implement security during the software development stages; thus, many programmers today do not practice these procedures.
- Most security professionals are usually not software developers.
- Many software developers do not have security as a main focus.
- Software vendors are trying to rush their products to market with their eyes set on functionality, not security.
- The computing community is used to receiving software with bugs and then applying patches.
- Customers cannot control the flaws in the software they purchase, so they must depend upon perimeter protection.

Finger-pointing and quick judgments are neither useful nor necessarily fair at this stage of our computing evolution. Twenty years ago, mainframes did not require much security because only a handful of people knew how to run them, users worked on computers (dumb terminals) that could not introduce malicious code to the mainframe, and environments were closed. The core protocols and framework were developed at a time when threats and attacks were not prevalent. Such stringent security wasn’t needed. Then, computer and software evolution took off, and the possibilities splintered into a thousand different directions. The high demand for computer technology and different types of software increased the demand for programmers, system designers, administrators, and engineers. This demand brought in a wave of people who had little experience. Thus, the lack of experience, the high change rate of technology, and the race to market added problems to security measures that are not always clearly understood.
Although it is easy to blame the big software vendors in the sky for producing flawed or buggy software, this is driven by customer demand. For at least a decade, and even today, we have been demanding more and more functionality from software vendors. The software vendors have done a wonderful job in providing these perceived necessities. It has only been in the last five years or so that customers started to also demand security. Our programmers were not properly educated in secure coding, operating systems and applications were not built on secure architectures from the beginning, our software development procedures have not been security-oriented, and integrating security as an afterthought makes the process all the clumsier. So although software vendors should be doing a better job providing us with secure products, we should also understand that this is a relatively new requirement and there is much more complexity when you peek under the covers than most consumers can even comprehend.

This chapter is an attempt to show how to address security at its source, which is at the software and development level. This requires a shift from reactive to proactive actions toward security problems to ensure they do not happen in the first place, or at least happen to a smaller extent. Figure 11-1 illustrates our current way of dealing with security issues.

![Usual Trend of Dealing with Security](image-url)
Different Environments Demand Different Security

*I demand total and complete security in each and every one of my applications!*

*Response: Well, don't hold your breath on that one.*

Today, network and security administrators are in an overwhelming position of having to integrate different applications and computer systems to keep up with their company's demand for expandable functionality and the new gee-whiz components that executives buy into and demand quick implementation of. This integration is further frustrated by the company's race to provide a well-known presence on the Internet by implementing web sites with the capabilities of taking online orders, storing credit card information, and setting up extranets with partners. This can quickly turn into a confusing ball of protocols, devices, interfaces, incompatibility issues, routing and switching techniques, telecommunications routines, and management procedures—all in all, a big enough headache to make an administrator buy some land in Montana and go raise goats instead.

On top of this, security is expected, required, and depended upon. When security compromises creep in, the finger-pointing starts, liability issues are tossed like hot potatoes, and people might even lose their jobs. An understanding of the environment, what is currently in it, and how it works is required so these new technologies can be implemented in a more controlled and comprehensible fashion.

The days of developing a simple web page and posting it on the Internet to illustrate your products and services are long gone. Today, the customer front-end, complex middleware, and three-tiered architectures must be developed and work seamlessly. As the complexity of this type of environment grows, tracking down errors and security compromises becomes an awesome task.

**The Client/Server Model**

Basically, the client/server architecture enables an application system to be divided across multiple platforms that vary in operating systems and hardware. The client requests services and the server fulfills these requests. The server handles the data-processing services and provides the processed result to the client. The client performs the front-end portion of an application, and the server performs the back-end portion, which is usually more labor intensive.

The front end usually includes the user interface and local data-manipulation capabilities, and provides the communications mechanisms that can request services from the server portion of the application.

**Environment vs. Application**

Software controls can be implemented by the operating system, by the application, or through database management controls—and usually a combination of all three is used. Each has its strengths and weaknesses, but if they are all understood and programmed to work in a concerted effort, then many different scenarios and types of compromises can be thwarted. One downside to relying mainly on operating system controls is that although they can control a subject's access to different objects and restrict the actions of that subject within the system, they do not necessarily restrict the
subject’s actions within an application. If an application has a security compromise within its own programming code, it is hard for the operating system to predict and control this vulnerability. An operating system is a broad environment for many applications to work within. It is unfair to expect the operating system to understand all the nuances of different programs and their internal mechanisms.

On the other hand, application controls and database management controls are very specific to their needs and the security compromises they understand. Although an application might be able to protect data by allowing only certain types of input and not permitting certain users to view data kept in sensitive database fields, it cannot prevent the user from inserting bogus data into the Address Resolution Protocol (ARP) table—this is the responsibility of the operating system and its network stack. Operating system and application controls have their place and limitations. The trick is to find out where one type of control stops so the next type of control can be configured to kick into action.

Security has been mainly provided by security products and perimeter devices rather than controls built into applications. The security products can cover a wide range of applications, can be controlled by a centralized management console, and are further away from application control. However, this approach does not always provide the necessary level of granularity, and does not approach compromises that can take place because of problematic coding and programming routines. Firewalls and access control mechanisms can provide a level of protection by preventing attackers from gaining access to be able to exploit buffer overflows, but the real protection happens at the core of the problem—proper software development and coding practices must be in place.

Complexity of Functionality

Programming is a complex trade—the code itself, routine interaction, global and local variables, input received from other programs, output fed to different applications, attempts to envision future user inputs, calculations, and restrictions form a long list of possible negative security consequences. Many times, trying to account for all the what-ifs and programming on the side of caution can reduce the overall functionality of the application. As you limit the functionality and scope of an application, the market share and potential profitability of that program could be reduced. A balancing act always exists between functionality and security, and in the development world, functionality is usually deemed the most important.

So, programmers and application architects need to find a happy medium between the necessary functionality of the program, the security requirements, and the mechanisms that should be implemented to provide this security. This can add more complexity to an already complex task.

More than one road may lead to enlightenment, but as these roads increase in number, it is hard to know if a path will eventually lead you to bliss or to fiery doom in the underworld. Many programs accept data from different parts of the program, other programs, the system itself, and user input. Each of these paths must be followed in a methodical way, and each possible scenario and input must be thought through and tested to provide a deep level of assurance. It is important that each module be capable of being tested individually and in concert with other modules. This level of understanding and testing will make the product more secure by catching flaws that could be exploited.
Data Types, Format, and Length

I would like my data to be in a small pink rectangle that I can fit in my pocket.

Response: You didn’t take your medication today, did you?

We have all heard about the vulnerabilities pertaining to buffer overflows, as if they were new to the programming world. They are not new, but they are being exploited nowadays on a recurring basis.

Buffer overflows were discussed in Chapter 5, which explained that attacks are carried out when the software code does not check the length of input that is actually being accepted. Extra instructions could be executed in a privileged mode that would enable an attacker to take control of the system. If a programmer wrote a program that expected the input length to be 5KB, then this needs to be part of the code so the right amount of buffer space is available to hold these data when they actually come in. However, if that program does not make sure the 5KB is accepted—and only that 5KB is accepted—an evildoer can input the first 5KB for the expected data to process, and then another 50KB containing malicious instructions can also be processed by the CPU.

Length is not the only thing programmers need to be worried about when it comes to accepting input data. Data also needs to be in the right format and data type. If the program is expecting alpha ASCII characters, it should not accept hexadecimal or UNICODE values.

The accepted value also needs to be reasonable. This means that if an application asks Stacy to enter the amount she would like to transfer from her checking account to her savings account, she should not be able to enter “Bob.” This means the data accepted by the program must be in the correct format (numbers versus alphabet characters), but procedures also need to be in place to watch for bogus entries so errors can be stopped at their origin instead of being passed to calculations and logic procedures.

These examples are extremely simplistic compared with what programmers have to face in the real programming world. However, they are presented to show that software needs to be developed to accept the correct data types, format, and length of input data for security and functionality purposes.

Implementation and Default Issues

If I have not said “yes,” then the answer is “no.”

As many people in the computer field know, out-of-the-box implementations are usually far from secure. Most security has to be configured and turned on after installation—not being aware of this can be dangerous for the inexperienced security person. Windows NT has received its share of criticism for lack of security, but the platform can be secured in many ways. It just comes out of the box in an insecure state, because settings have to be configured to properly integrate into different environments, and this is a friendlier way of installing the product for users. For example, if Mike is installing a new software package that continually throws messages of “Access Denied” when he is attempting to configure it to interoperate with other applications and systems, his patience might wear thin, and he might decide to hate that vendor for years to come because of the stress and confusion inflicted upon him.
Yet again, we are at a hard place for developers and architects. When a security application or device is installed, it should default to “No Access.” This means that when Laurel installs a packet-filter firewall, it should not allow any packets to pass into the network that were not specifically granted access. However, this requires Laurel to know how to configure the firewall for it to ever be useful. A fine balance exists between security, functionality, and user-friendliness. If an application is extremely user-friendly, it is probably not as secure. For an application to be user-friendly, it usually requires a lot of extra coding for potential user errors, dialog boxes, wizards, and step-by-step instructions. This extra coding can result in bloated code that can create unforeseeable compromises. So vendors have a hard time winning, but they usually keep making money while trying.

NOTE Later versions of Windows have services turned off and require the user to turn them on as needed. This is a step closer to “default with no access,” but we still have a ways to go.

Implementation errors and misconfigurations are common issues that cause a majority of the security issues in networked environments. Many people do not realize that various services are enabled when a system is installed. These services can provide evildoers with information that can be used during an attack. Many services provide an actual way into the environment itself. NetBIOS services can be enabled to permit sharing resources in Windows environments, and Telnet services, which let remote users run command shells, and other services can be enabled with no restrictions. Many systems have File Transfer Protocol (FTP), SNMP, and Internet Relay Chat (IRC) services enabled that are not being used and have no real safety measures in place. Some of these services are enabled by default, so when an administrator installs an operating system and does not check these services to properly restrict or disable them, they are available for attackers to uncover and use.

Because vendors have user-friendliness and user functionality in mind, the product will usually be installed with defaults that provide no, or very little, security protection. It would be very hard for vendors to know the security levels required in all the environments the product will be installed in, so they usually do not attempt it. It is up to the person installing the product to learn how to properly configure the settings to achieve the necessary level of protection.

Another problem in implementation and security is the number of unpatched systems. Once security issues are identified, vendors develop patches or updates to address and fix these security holes. However, these often do not get installed on the systems that are vulnerable. The reasons for this vary: administrators may not keep up-to-date on the recent security vulnerabilities and patches, they may not fully understand the importance of these patches, or they may be afraid the patches will cause other problems. All of these reasons are quite common, but they all have the same result—insecure systems. Many vulnerabilities that are exploited today have had patches developed and released months or years ago.

It is unfortunate that adding security (or service) patches can adversely affect other mechanisms within the system. The patches should be tested for these types of activities
before they are applied to production servers and workstations, to help prevent service disruptions that can affect network and employee productivity.

**Failure States**

Many circumstances are unpredictable and are therefore hard to plan for. However, unpredictable situations can be planned for in a general sense, instead of trying to plan and code for every situation. If an application fails for any reason, it should return to a safe and more secure state. This could require the operating system to restart and present the user with a logon screen to start the operating system from its initialization state. This is why some systems “blue-screen” and/or restart. When this occurs, something is going on within the system that is unrecognized or unsafe, so the system dumps its memory contents and starts all over.

Different system states were discussed in Chapter 5, which described how processes can be executed in a privileged or user mode. If an application fails and is executing in a privileged state, these processes should be shut down properly and released to ensure that disrupting a system does not provide compromises that could be exploited. If a privileged process does not shut down properly and instead stays active, an attacker can figure out how to access the system, using this process, in a privileged state. This means the attacker could have administrative or root access to a system, which opens the door for more severe destruction.

**Database Management**

*From now on I am going to manage the database with ESP.*

Response: Well, your crystals, triangles, and tarot cards aren’t working.

Databases have a long history of storing important intellectual property and items that are considered valuable and proprietary to companies. Because of this, they usually live in an environment of mystery to all but the database and network administrators. The less anyone knows about the databases, the better. Users usually access databases indirectly through a client interface, and their actions are restricted to ensure the confidentiality, integrity, and availability of the data held within the database and the structure of the database itself.

**NOTE**

A database management system (DBMS) is a suite of programs used to manage large sets of structured data with ad hoc query capabilities for many types of users. These can also control the security parameters of the database.

The risks are increasing as companies run to connect their networks to the Internet, allow remote user access, and provide more and more access to external entities. A large risk to understand is that these activities can allow indirect access to a back-end database. In the past, employees accessed customer information held in databases instead of customers accessing it themselves. Today, many companies allow their customers to access data in their databases through a browser. The browser makes a connection to the company’s middleware, which then connects them to the back-end database. This adds levels of complexity, and the database will be accessed in new and unprecedented ways.
One example is in the banking world, where online banking is all the rage. Many financial institutions want to keep up with the times and add the services they think their customers will want. But online banking is not just another service like being able to order checks. Most banks work in closed (or semiclosed) environments, and opening their environments to the Internet is a huge undertaking. The perimeter network needs to be secured, middleware software has to be developed or purchased, and the database should be behind one, preferably two, firewalls. Many times, components in the business application tier are used to extract data from the databases and process the customer requests.

Access control can be restricted by only allowing roles to interact with the database. The database administrator can define specific roles that are allowed to access the database. Each role will have assigned rights and permissions, and customers and employees are then ported into these roles. Any user who is not within one of these roles is denied access. This means that if an attacker compromises the firewall and other perimeter network protection mechanisms, and then is able to make requests to the database, if he is not in one of the predefined roles, the database is still safe. This process streamlines access control and ensures that no users or evildoers can access the database directly, but must access it indirectly through a role account. Figure 11-2 illustrates these concepts.

**Database Management Software**

A *database* is a collection of data stored in a meaningful way that enables multiple users and applications to access, view, and modify data as needed. Databases are managed with software that provides these types of capabilities. It also enforces access control restrictions, provides data integrity and redundancy, and sets up different procedures for data manipulation. This software is referred to as a *database management system (DBMS)* and is usually controlled by a database administrator. Databases not only

![Figure 11-2](image-url)
store data, but may also process data and represent it in a more usable and logical form. DBMSs interface with programs, users, and data within the database. They help us store, organize, and retrieve information effectively and efficiently.

A database is the mechanism that provides structure for the data collected. The actual specifications of the structure may be different per database implementation, because different organizations or departments work with different types of data and need to perform diverse functions upon that information. There may be different workloads, relationships between the data, platforms, performance requirements, and security goals. Any type of database should have the following characteristics:

- It centralizes by not having data held on several different servers throughout the network.
- It allows for easier backup procedures.
- It provides transaction persistence.
- It allows for more consistency since all the data are held and maintained in one central location.
- It provides recovery and fault tolerance.
- It allows the sharing of data with multiple users.
- It provides security controls that implement integrity checking, access control, and the necessary level of confidentiality.

**NOTE** Transaction persistence means the database procedures carrying out transactions are durable and reliable. The state of the database’s security should be the same after a transaction has occurred and the integrity of the transaction needs to be ensured.

Because the needs and requirements for databases vary, different data models can be implemented that align with different business and organizational needs.

**Database Models**

*Ohhh, that database model is very pretty, indeed.*
*Response: You have problems.*

The database model defines the relationships between different data elements, dictates how data can be accessed, and defines acceptable operations, the type of integrity offered, and how the data is organized. A model provides a formal method of representing data in a conceptual form and provides the necessary means of manipulating the data held within the database. Databases come in several types of models, as listed next:

- Relational
- Hierarchical
- Network
A relational database model uses attributes (columns) and tuples (rows) to contain and organize information (see Figure 11-3). The relational database model is the most widely used model today. It presents information in the form of tables. A relational database is composed of two-dimensional tables, and each table contains unique rows, columns, and cells (the intersection of a row and a column). Each cell contains only one data value that represents a specific attribute value within a given tuple. These data entities are linked by relationships. The relationships between the data entities provide the framework for organizing data. A primary key is a field that links all the data within a record to a unique value. For example, in the table in Figure 11-3, the primary keys are Product G345 and Product G978. When an application or another record refers to this primary key, it is actually referring to all the data within that given row.

A hierarchical data model (see Figure 11-4) combines records and fields that are related in a logical tree structure. The structure and relationship between the data elements are different from those in a relational database. In the hierarchical database the parents can have one child, many children, or no children. The tree structure contains branches, and each branch has a number of leaves, or data fields. These databases have well-defined, prespecified access paths, but are not as flexible in creating relationships between data elements as a relational database. Hierarchical databases are useful for mapping one-to-many relationships.

The hierarchical structured database is one of the first types of database model created, but is not as common as relational databases. To be able to access a certain data entity within a hierarchical database requires the knowledge of which branch to start with and which route to take through each layer until the data are reached. It does not use indexes as relational databases do for searching procedures. Also links (relationships) cannot be created between different branches and leaves on different layers.

The most commonly used implementation of the hierarchical model is in the Lightweight Directory Access Protocol (LDAP) model. You can find this model also used in

![Figure 11-3](https://example.com/figure11_3.png)

**Figure 11-3** Relational databases hold data in table structures.
the Windows registry structure and different file systems, but it is not commonly used in newer database products.

The network database model is built upon the hierarchical data model. Instead of being constrained by having to know how to go from one branch to another and then from one parent to a child to find a data element, the network database model allows each data element to have multiple parent and child records. This forms a redundant network-like structure instead of a strict tree structure. (The name does not indicate it is on or distributed throughout a network, it just describes the data element relationships.) If you look at Figure 11-5, you can see how a network model sets up a structure that is similar to a mesh network topology for the sake of redundancy and allows for quick retrieval of data compared to the hierarchical model.

**NOTE** In Figure 11-5 you will also see a comparison of different database models.

This model uses the constructs of records and sets. A record contains fields, which may lay out in a hierarchical structure. Sets define the one-to-many relationships between the different records. One record can be the “owner” of any number of sets and the same “owner” can be a member of different sets. This means that one record can be the “top dog” and have many data elements underneath it, or that record can be lower on the totem pole and be beneath a different field that is its “top dog.” This allows for a lot of flexibility in the development of relationships between data elements.
An object-oriented database is designed to handle a variety of data (images, audio, documents, video). An object-oriented database management system (ODBMS) is more dynamic in nature than a relational database, because objects can be created when needed and the data and procedure (called method) go with the object when it is requested. In a relational database, the application has to use its own procedures to obtain data from the database and then process the data for its needs. The relational database does not actually provide procedures, as object-oriented databases do. The object-oriented database has classes to define the attributes and procedures of its objects.

As an analogy, let’s say two different companies provide the same data to their customer bases. If you go to Company A (relational), the person behind the counter will just give you a piece of paper that contains information. Now you have to figure out what to do with that information and how to properly use it for your needs. If you go to Company B (object-oriented), the person behind the counter will give you a box. Within this box is a piece of paper with information on it, but you will also be given a couple of tools to process the data for your needs instead of you having to do it yourself. So in object-oriented databases, when your application queries for some data, what is returned is not only the data but the code to carry out procedures on this data. (When we get to object-oriented programming, you will understand objects, classes and methods more fully.)
The goal of creating this type of model was to address the limitations that relational databases encountered when large amounts of data must be stored and processed. An object-oriented database also does not depend upon SQL for interactions, so applications that are not SQL clients can work with these types of databases.

**NOTE** Structured Query Language (SQL) is a standard programming language used to allow clients to interact with a database. Many database products support SQL. It allows clients to carry out operations such as inserting, updating, searching, and committing data. When a client interacts with a database, it is most likely using SQL to carry out requests.

ODBMSs are not as common as relational databases, but are used in niche areas such as engineering and biology, and for some financial sector needs.

Now let’s look at object-relational databases, just for the fun of it. An object-relational database (ORD) or object-relational database management system (ORDBMS) is a relational database with a software front end that is written in an object-oriented programming language. Why would we create such a silly combination? Well, a rela-
tional database just holds data in static two-dimensional tables. When the data are accessed, some type of processing needs to be carried out on it—otherwise, there is really no reason to obtain the data. If we have a front end that provides the procedures (methods) that can be carried out on the data, then each and every application that accesses this database does not need to have the necessary procedures. This means that each and every application does not need to contain the procedures necessary to gain what it really wants from this database.

Different companies will have different business logic that needs to be carried out on the stored data. Allowing programmers to develop this front-end software piece allows the business logic procedures to be used by requesting applications and the data within the database. For example, if we had a relational database that contains inventory data for our company, we might want to be able to use this data for different business purposes. One application can access that database and just check the quantity of widget A products we have in stock. So a front-end object that can carry out that procedure will be created, the data will be grabbed from the database by this object, and the answer will be provided to the requesting application. We also have a need to carry out a trend analysis, which will indicate which products were moved the most from inventory to production. A different object that can carry out this type of calculation will gather the necessary data and present it to our requesting application. We have many different ways we need to view the data in that database: how many products were damaged during transportation, how fast did each vendor fulfill our supply requests, how much does it cost to ship the different products based on their weights, and so on. The data objects in Figure 11-6 contain these different business logic instructions.

**Database Programming Interfaces**

Data are useless if you can’t get to them and use them. Applications need to be able to obtain and interact with the information stored in databases. They also need some type
of interface and communication mechanism. The following sections address some of these interface languages:

- **Open Database Connectivity (ODBC)** An application programming interface (API) that allows an application to communicate with a database either locally or remotely. The application sends requests to the ODBC API. ODBC tracks down the necessary database-specific driver for the database to carry out the translation, which in turn translates the requests into the database commands that a specific database will understand.

- **Object Linking and Embedding Database (OLE DB)** Separates data into components that run as middleware on a client or server. It provides a low-level interface to link information across different databases and provides access to data no matter where it is located or how it is formatted. The following are some characteristics of OLE DB:
  - A replacement for ODBC, extending its feature set to support a wider variety of nonrelational databases, such as object databases and spreadsheets that do not necessarily implement SQL.
  - A set of COM-based interfaces that provide applications with uniform access to data stored in diverse data sources (see Figure 11-7).
  - Because it is COM-based, OLE DB is limited to use by Microsoft Windows–based client tools. (Unrelated to OLE.)

![Figure 11-7](image-url) OLE DB provides an interface to allow applications to communicate with different data sources.
A developer accesses OLE DB services through ActiveX data objects (ADO).

- **ActiveX Data Objects (ADO)**  
  An API that allows applications to access back-end database systems. It is a set of ODBC interfaces that exposes the functionality of a database through accessible objects. ADO uses the OLE DB interface to connect with the database and can be developed with many different scripting languages. The following are some characteristics of ADO:
  - It's a high-level data access programming interface to an underlying data access technology (such as OLE DB).
  - It's a set of COM objects for accessing data sources, not just database access.
  - It allows a developer to write programs that access data, without knowing how the database is implemented.
  - SQL commands are not required to access a database when using ADO.

- **Java Database Connectivity (JDBC)**  
  An API that allows a Java application to communicate with a database. The application can bridge through ODBC or directly to the database. The following are some characteristics of JDBC:
  - It is an API that provides the same functionality as ODBC but is specifically designed for use by Java database applications.
  - Has database-independent connectivity between the Java platform and a wide range of databases.
  - JDBC is a Java API that enables Java programs to execute SQL statements.

- **Extensible Markup Language (XML)**  
  A standard for structuring data so it can be easily shared by applications using web technologies. It is a markup standard that is self-defining and provides a lot of flexibility in how data within the database is presented. The web browser interprets the XML tags to illustrate to the user how the developer wanted the data to be presented.

### Relational Database Components

Like all software, databases are built with programming languages. Most database languages include a **data definition language (DDL)**, which defines the schema; a **data manipulation language (DML)**, which examines data and defines how the data can be manipulated within the database; a **data control language (DCL)**, which defines the internal organization of the database, and an ad hoc **query language (QL)**, which defines queries that enable users to access the data within the database.

Each type of database model may have many other differences, which vary from vendor to vendor. Most, however, contain the following basic core functionalities:

- **Data definition language (DDL)**  
  Defines the structure and schema of the database. The structure could mean the table size, key placement, views, and data element relationship. The **schema** describes the type of data that will be held and manipulated, and its properties. It defines the structure of the database, access operations, and integrity procedures.
• Data manipulation language (DML) Contains all the commands that enable a user to view, manipulate, and use the database (view, add, modify, sort, and delete commands).

• Query language (QL) Enables users to make requests of the database.

• Report generator Produces printouts of data in a user-defined manner.

Data Dictionary
Will the data dictionary explain all the definitions of database jargon to me?
Response: Wrong dictionary.

A data dictionary is a central collection of data element definitions, schema objects, and reference keys. The schema objects can contain tables, views, indexes, procedures, functions, and triggers. A data dictionary can contain the default values for columns, integrity information, the names of users, the privileges and roles for users, and auditing information. It is a tool used to centrally manage parts of a database by controlling data about the data (referred to as metadata) within the database. It provides a cross-reference between groups of data elements and the databases.

A data dictionary is a central collection of data element definitions, schema objects, and reference keys. The schema objects can contain tables, views, indexes, procedures, functions, and triggers. A data dictionary can contain the default values for columns, integrity information, the names of users, the privileges and roles for users, and auditing information.

The database management software creates and reads the data dictionary to ascertain what schema objects exist and checks to see if specific users have the proper access rights to view them (see Figure 11-8). When users look at the database, they can be restricted by specific views. The different view settings for each user are held within the data dictionary. When new tables, new rows, or new schema are added, the data dictionary is updated to reflect this.

Primary vs. Foreign Key
Hey, my primary key is stuck to my foreign key.
Response: That is the whole idea of their existence.

The primary key is an identifier of a row and is used for indexing in relational databases. Each row must have a unique primary key to properly represent the row as one entity. When a user makes a request to view a record, the database tracks this record by its unique primary key. If the primary key were not unique, the database would not know which record to present to the user. In the following illustration, the primary keys for Table A are the dogs’ names. Each row (tuple) provides characteristics for each dog (primary key). So when a user searches for Cricket, the characteristics of the type, weight, owner, and color will be provided.
A primary key is different from a foreign key, although they are closely related. If an attribute in one table has a value matching the primary key in another table and there is a relationship set up between the two of them, this attribute is considered a foreign key. This foreign key is not necessarily the primary key in its current table. It only has to

**Figure 11-8** The data dictionary is a centralized program that contains information about a database.
contain the same information that is held in another table's primary key and be mapped to the primary key in this other table. In the following illustration, a primary key for Table A is Dallas. Because Table B has an attribute that contains the same data as this primary key and there is a relationship set up between these two keys, it is referred to as a foreign key. This is another way for the database to track relationships between data that it houses.

We can think of being presented with a web page that contains the data on Table B. If we want to know more about this dog named Dallas, we double-click that value and the browser presents the characteristics about Dallas that are in Table A.

This allows us to set up our databases with the relationship between the different data elements as we see fit.

**Integrity**

*You just wrote over my table!*

*Response: Well, my information is more important than yours.*

Like other resources within a network, a database can run into *concurrency* problems. Concurrency issues come up when there is a piece of software that will be accessed at the same time by different users and/or applications. As an example of a concurrency problem, suppose that two groups use one price sheet to know how many supplies to order for the next week and also to calculate the expected profit. If Dan and Elizabeth copy this price sheet from the file server to their workstations, they each have...
a copy of the original file. Suppose that Dan changes the stock level of computer books from 120 to 5, because they sold 115 in the last three days. He also uses the current prices listed in the price sheet to estimate his expected profits for the next week. Elizabeth reduces the price on several software packages on her copy of the price sheet and sees that the stock level of computer books is still over 100, so she chooses not to order any more for next week for her group. Dan and Elizabeth do not communicate this different information to each other, but instead upload their copies of the price sheet to the server for everyone to view and use.

Dan copies his changes back to the file server, and then 30 seconds later Elizabeth copies her changes over Dan’s changes. So, the file only reflects Elizabeth’s changes. Because they did not synchronize their changes, they are both now using incorrect data. Dan’s profit estimates are off because he does not know that Elizabeth reduced the prices, and next week Elizabeth will have no computer books because she did not know that the stock level had dropped to five.

The same thing happens in databases. If controls are not in place, two users can access and modify the same data at the same time, which can be detrimental to a dynamic environment. To ensure that concurrency problems do not cause problems, processes can lock tables within a database, make changes, and then release the software lock. The next process that accesses the table will then have the updated information. Locking ensures that two processes do not access the same table at the same time. Pages, tables, rows, and fields can be locked to ensure that updates to data happen one at a time, which enables each process and subject to work with correct and accurate information.

Database software performs three main types of integrity services: semantic, referential, and entity. A semantic integrity mechanism makes sure structural and semantic rules are enforced. These rules pertain to data types, logical values, uniqueness constraints, and operations that could adversely affect the structure of the database. A database has referential integrity if all foreign keys reference existing primary keys. There should be a mechanism in place that ensures no foreign key contains a reference to a primary key of a nonexisting record, or a null value. Entity integrity guarantees that the tuples are uniquely identified by primary key values. In the previous illustration, the primary keys are the names of the dogs, in which case, no two dogs could have the same name. For the sake of entity integrity, every tuple must contain one primary key. If it does not have a primary key, it cannot be referenced by the database.

The database must not contain unmatched foreign key values. Every foreign key refers to an existing primary key. In the previous illustration, if the foreign key in Table B is Dallas, then Table A must contain a record for a dog named Dallas. If these values do not match, then their relationship is broken, and again the database cannot reference the information properly.

Other configurable operations are available to help protect the integrity of the data within a database. These operations are rollbacks, commits, savepoints, and checkpoints.

The rollback is an operation that ends a current transaction and cancels the current changes to the database. These changes could have taken place with the data itself or with schema changes that were typed in. When a rollback operation is executed, the changes are cancelled, and the database returns to its previous state. A rollback can take place if the database has some type of unexpected glitch or if outside entities disrupt its
processing sequence. Instead of transmitting and posting partial or corrupt information, the database will roll back to its original state and log these errors and actions so they can be reviewed later.

The commit operation completes a transaction and executes all changes just made by the user. As its name indicates, once the commit command is executed, the changes are committed and reflected in the database. These changes can be made to data or schema information. By committing these changes, they are then available to all other applications and users. If a user attempts to commit a change and it cannot complete correctly, a rollback is performed. This ensures that partial changes do not take place and that data are not corrupted.

Savepoints are used to make sure that if a system failure occurs, or if an error is detected, the database can attempt to return to a point before the system crashed or hiccupped. For a conceptual example, say Dave typed, “Jeremiah was a bullfrog. He was <savepoint> a good friend of mine.” (The system inserted a savepoint.) Then a freak storm came through and rebooted the system. When Dave got back into the database client application, he might see “Jeremiah was a bullfrog. He was,” but the rest was lost. Therefore, the savepoint saved some of his work. Databases and other applications will use this technique to attempt to restore the user’s work and the state of the database after a glitch, but some glitches are just too large and invasive to overcome.

Savepoints are easy to implement within databases and applications, but a balance must be struck between too many and not enough savepoints. Having too many savepoints can degrade the performance, whereas not having enough savepoints runs the risk of losing data and decreasing user productivity because the lost data would have to be reentered. Savepoints can be initiated by a time interval, a specific action by the user, or the number of transactions or changes made to the database. For example, a database can set a savepoint for every 15 minutes, every 20 transactions completed, each time a user gets to the end of a record, or every 12 changes made to the databases.

So a savepoint restores data by enabling the user to go back in time before the system crashed or hiccupped. This can reduce frustration and help us all live in harmony.

**NOTE** Checkpoints are very similar to savepoints. When the database software fills up a certain amount of memory, a checkpoint is initiated, which saves the data from the memory segment to a temporary file. If a glitch is experienced, the software will try to use this information to restore the user’s working environment to its previous state.

A two-phase commit mechanism is yet another control that is used in databases to ensure the integrity of the data held within the database. Databases commonly carry out transaction processes, which means the user and the database interact at the same time. The opposite is batch processing, which means that requests for database changes are put into a queue and activated all at once—not at the exact time the user makes the request. In transactional processes, many times a transaction will require that more than one database be updated during the process. The databases need to make sure each database is properly modified, or no modification takes place at all. When a database change is submitted by the user, the different databases initially store these changes temporarily. A transaction monitor will then send out a “pre-commit” command to
each database. If all the right databases respond with an acknowledgment, then the
monitor sends out a “commit” command to each database. This ensures that all of the
necessary information is stored in all the right places at the right time.

**Reference**

- Databases 1 & 2  [http://stein.cshl.org/genome_informatics/Intro_to_DB/](http://stein.cshl.org/genome_informatics/Intro_to_DB/)

**Database Security Issues**

*Oh, I know this and I know that. Now I know the big secret!*  
*Response: Then I am changing the big secret—hold on.*

The two main database security issues this section addresses are aggregation and
inference. **Aggregation** happens when a user does not have the clearance or permission
to access specific information, but she does have the permission to access components
of this information. She can then figure out the rest and obtain restricted information.
She can learn of information from different sources and combine it to learn something
she does not have the clearance to know.

**NOTE**  
Aggregation is the act of combining information from separate
sources. The combination of the data forms new information, which the
subject does not have the necessary rights to access. The combined
information has a sensitivity that is greater than that of the individual parts.

The following is a silly conceptual example. Let’s say a database administrator does not
want anyone in the Users group to be able to figure out a specific sentence, so he segregates
the sentence into components and restricts the Users group from accessing it, as represented
in Figure 11-9. However, Emily can access components A, C, and F. Because she is particularly
bright, she figures out the sentence and now knows the restricted secret.

To prevent aggregation, the subject, and any application or process acting on the
subject’s behalf, needs to be prevented from gaining access to the whole collection,
including the independent components. The objects can be placed into containers,
which are classified at a higher level to prevent access from subjects with lower-level
permissions or clearances. A subject’s queries can also be tracked, and context-dependent
access control can be enforced. This would keep a history of the objects that a
subject has accessed and restrict an access attempt if there is an indication that an ag-
gregation attack is under way.

The other security issue is **inference**, which is the intended result of aggregation. The
inference problem happens when a subject deduces the full story from the pieces he
learned of through aggregation. This is seen when data at a lower security level indi-
directly portrays data at a higher level.

**NOTE**  
Inference is the ability to derive information not explicitly available.
For example, if a clerk were restricted from knowing the planned movements of troops based in a specific country, but did have access to food shipment requirements forms and tent allocation documents, he could figure out that the troops were moving to a specific place because that is where the food and tents are being shipped. The food shipment and tent allocation documents were classified as confidential, and the troop movement was classified as top secret. Because of the varying classifications, the clerk could access and ascertain top-secret information he was not supposed to know.

The trick is to prevent the subject, or any application or process acting on behalf of that subject, from indirectly gaining access to the inferable information. This problem is usually dealt with in the development of the database by implementing content- and context-dependent access control rules. Content-dependent access control is based on the sensitivity of the data. The more sensitive the data, the smaller the subset of individuals who can gain access to the data.

Context-dependent access control means that the software “understands” what actions should be allowed based upon the state and sequence of the request. So what does that mean? It means the software must keep track of previous access attempts by the user and understand what sequences of access steps are allowed. Where content-dependent access control can go like this, “Does Julio have access to File A?” and the system reviews the ACL on File A and returns with a response of “Yes, Julio can access the file, but can only read it.” In a context-dependent access control situation, it would be more like, “Does Julio have access to File A?” The system then reviews several pieces of data: What other access attempts has Julio made? Is this request out of sequence of how a safe series of requests takes place? Does this request fall within the allowed time period of system access (8 A.M. to 5 P.M.)? If the answers to all of these questions are within a set of preconfigured parameters, Julio can access the file. If not, he needs to go find something else to do.

Figure 11-9  Because Emily has access to components A, C, and F, she can figure out the secret sentence through aggregation.
Obviously, content-dependent access control is not as complex as context-dependent control because of the amount of items that needs to be processed by the system.

Common attempts to prevent inference attacks are cell suppression, partitioning the database, and noise and perturbation. Cell suppression is a technique used to hide specific cells that contain information that could be used in inference attacks. Partitioning a database involves dividing the database into different parts, which makes it much harder for an unauthorized individual to find connecting pieces of data that can be brought together and other information that can be deduced or uncovered. Noise and perturbation is a technique of inserting bogus information in the hopes of misdirecting an attacker or confusing the matter enough that the actual attack will not be fruitful.

If context-dependent access control is being used to protect against inference attacks, the database software would need to keep track of what the user is requesting. So Julio makes a request to see field 1, then field 5, then field 20 which the system allows, but once he asks to see field 15 the database does not allow this access attempt. The software must be preprogrammed (usually through a rule-based engine) as to what sequence and how much data Julio is allowed to viewed. If he is allowed to view more information, he may have enough data to infer something we don’t want him to know.

Often, security is not integrated into the planning and development of a database. Security is an afterthought, and a trusted front end is developed to be used with the database instead. This approach is limited in the granularity of security and in the types of security functions that can take place.

A common theme in security is a balance between effective security and functionality. In many cases, the more you secure something, the less functionality you have. Although this could be the desired result, it is important not to impede user productivity when security is being introduced.

**Database Views**

*Don’t show your information to everybody, only a select few.*

Databases can permit one group, or a specific user, to see certain information while restricting another group from viewing it altogether. This functionality happens through the use of database views, illustrated in Figure 11-10. If a database administrator wants to allow middle management members to see their departments’ profits and expenses but not show them the whole company’s profits, she can implement views. Senior management would be given all views, which contain all the departments’ and the company’s profit and expense values, whereas each individual manager would only be able to view his or her department values.

Like operating systems, databases can employ discretionary access control (DAC) and mandatory access control (MAC) (explained in Chapter 4). Views can be displayed according to group membership, user rights, or security labels. If a DAC system was employed, then groups and users could be granted access through views based on their identity, authentication, and authorization. If a MAC system was in place, then groups and users would be granted access based on their security clearance and the data’s classification level.
Polyinstantiation

Polyinstantiation.

Response: Gesundheit.

Sometimes a company does not want users at one level to access and modify data at a higher level. This type of situation can be handled in different ways. One approach denies access when a lower-level user attempts to access a higher-level object. However, this gives away information indirectly by telling the lower-level entity that something sensitive lives inside that object at that level.

Another way of dealing with this issue is polyinstantiation. This enables a table that contains multiple tuples with the same primary keys, with each instance distinguished by a security level. When this information is inserted into a database, lower-level subjects must be restricted from it. Instead of just restricting access, another set of data is created to fool the lower-level subjects into thinking the information actually means something else. For example, if a naval base has a cargo shipment of weapons going from Delaware to Ukraine via the ship, Oklahoma, this type of information could be classified as top secret. Only the subjects with the security clearance of top secret and above should know this information, so a dummy file is created that states the Oklahoma is carrying a shipment from Delaware to Africa containing food, and it is given a security clearance of unclassified, as shown in Table 11-1. It will be obvious that the Oklahoma is gone, but individuals at lower security levels will think the ship is on its way to Africa, instead of Ukraine. This also makes sure no one at a lower level tries to commit the Oklahoma for any other missions. The lower-level subjects know that the Oklahoma is not available, and they will assign other ships for cargo shipments.

**NOTE** Polyinstantiation is a process of interactively producing more detailed versions of objects by populating variables with different values or other variables. It is often used to prevent inference attacks.
In this example, polyinstantiation was used to create two versions of the same object so lower-level subjects did not know the true information, and thus stopped them from attempting to use or change that data in any way. It is a way of providing a cover story for the entities that do not have the necessary security level to know the truth. This is just one example of how polyinstantiation can be used. It is not strictly related to security, however, even though that is its most common use. Whenever a copy of an object is created and populated with different data, meaning two instances of the same object have different attributes, polyinstantiation is in place.

### Online Transaction Processing

What if our databases get overwhelmed?  
Response: OLTP to the rescue!

Online transaction processing (OLTP) is usually used when databases are clustered to provide fault tolerance and higher performance. OLTP provides mechanisms that watch for problems and deal with them appropriately when they do occur. For example, if a process stops functioning, the monitor mechanisms within OLTP can detect this and attempt to restart the process. If the process cannot be restarted, then the transaction taking place will be rolled back to ensure no data is corrupted or that only part of a transaction happens. Any erroneous or invalid transactions detected should be written to a transaction log. The transaction log also collects the activities of successful transactions. Data is written to the log before and after a transaction is carried out so a record of events exists.

The main goal of OLTP is to ensure that transactions happen properly or they don’t happen at all. Transaction processing usually means that individual indivisible operations are taking place independently. If one of the operations fails, the rest of the operations needs to be rolled back to ensure that only accurate data is entered into the database.

The set of systems involved in carrying out transactions are managed and monitored with a software OLTP product to make sure everything takes place smoothly and correctly.

OLTP can load balance incoming requests if necessary. This means that if requests to update databases increase, and the performance of one system decreases because of the large volume, OLTP can move some of these requests to other systems. This makes sure all requests are handled and that the user, or whoever is making the requests, does not have to wait a long time for the transaction to complete.

When there is more than one database, it is important they all contain the same information. Consider this scenario—Katie goes to the bank and withdraws $6500
from her $10,000 checking account. Database A receives the request and records a new checking account balance of $3500, but database B does not get updated. It still shows a balance of $10,000. Then, Katie makes a request to check the balance on her checking account, but that request gets sent to database B, which returns inaccurate information because the withdrawal transaction was never carried over to this database. OLTP makes sure a transaction is not complete until all databases receive and reflect this change.

OLTP records transactions as they occur (in real time), which usually updates more than one database in a distributed environment. This type of complexity can introduce many integrity threats, so the database software should implement the characteristics of what’s known as the ACID test:

- **Atomicity** Divides transactions into units of work and ensures that all modifications take effect or none takes effect. Either the changes are committed or the database is rolled back.
- **Consistency** A transaction must follow the integrity policy developed for that particular database and ensure all data are consistent in the different databases.
- **Isolation** Transactions execute in isolation until completed, without interacting with other transactions. The results of the modification are not available until the transaction is completed.
- **Durability** Once the transaction is verified as accurate on all systems, it is committed, and the databases cannot be rolled back.

**Data Warehousing and Data Mining**

*Data warehousing* combines data from multiple databases or data sources into a large database for the purpose of providing more extensive information retrieval and data analysis. Data from different databases is extracted and transferred to a central data storage device called a warehouse. The data is normalized, which means redundant information is stripped out and data are formatted in the way the data warehouse expects it. This enables users to query one entity rather than accessing and querying different databases.

The data sources the warehouse is built from are used for operational purposes. A data warehouse is developed to carry out analysis. The analysis can be carried out to make business forecasting decisions, identify marketing effectiveness, business trends, and even fraudulent activities.

Data warehousing is not simply a process of mirroring data from different databases and presenting the data in one place. It provides a base of data that is then processed and presented in a more useful and understandable way. Related data is summarized and correlated before it is presented to the user. Instead of having every piece of data presented, the user is given data in a more abridged form that best fits her needs.
Although this provides easier access and control, because the data warehouse is in one place, it also requires more stringent security. If an intruder got into the data warehouse, he could access all of the company’s information at once.

**Data mining** is the process of massaging the data held in the data warehouse into more useful information. Data-mining tools are used to find an association and correlation in data to produce *metadata*. Metadata can show previously unseen relationships between individual subsets of information. It can reveal abnormal patterns not previously apparent. A simplistic example in which data mining could be useful is in detecting insurance fraud. Suppose the information, claims, and specific habits of millions of customers are kept in a database warehouse, and a mining tool is used to look for certain patterns in claims. It might find that each time John Smith moved, he had an insurance claim two to three months following the move. He moved in 1967 and two months later had a suspicious fire, then moved in 1973 and had a motorcycle stolen three months after that, and then moved again in 1984 and had a burglar break-in two months afterward. This pattern might be hard for people to manually catch because he had different insurance agents over the years, the files were just updated and not reviewed, or the files were not kept in a centralized place for agents to review.

Data mining can look at complex data and simplify it by using fuzzy logic, a set theory, and expert systems to perform the mathematical functions and look for patterns in data that are not so apparent. In many ways, the metadata is more valuable than the data it was derived from; thus, it must be highly protected. (Fuzzy logic and expert systems are discussed later in this chapter, in the “Artificial Neural Networks” section.)

The goal of data warehouses and data mining is to be able to extract information to gain knowledge about the activities and trends within the organization, as shown in Figure 11-11. With this knowledge, people can detect deficiencies or ways to optimize operations. For example, if we worked at a retail store company, we would want consumers to spend gobs and gobs of money there. We can better get their business if we understood customers’ purchasing habits. If candy and other small items are placed at the checkout stand, purchases of those items go up 65 percent compared to if the items were somewhere else in the store. If one store is in a more affluent neighborhood and we see a constant (or increasing) pattern of customers purchasing expensive wines there, that is where we would also sell our expensive cheeses and gourmet items. We would not place our gourmet items at another store that frequently accepts food stamps.

**NOTE**  
Data mining is the process of analyzing a data warehouse using tools that look for trends, correlations, relationships, and anomalies without knowing the meaning of the data. Metadata is the result of storing data within a data warehouse and mining the data with tools. Data goes into a data warehouse and metadata comes out of that data warehouse.
So we would carry out these activities if we want to harness organization-wide data for comparative decision making, workflow automation, and/or competitive advantage. It is not just information-aggregation; management’s goals in understanding different aspects of the company are to enhance business value and help employees work more productively.
Data mining is also known as *knowledge discovery in database (KDD)*, and is a combination of techniques to identify valid and useful patterns. Different types of data can have various interrelationships, and the method used depends on the type of data and the patterns sought. The following are three approaches used in KDD systems to uncover these patterns:

- **Classification** Groups together data according to shared similarities.
- **Probabilistic** Identifies data interdependencies and applies probabilities to their relationships.
- **Statistical** Identifies relationships between data elements and uses rule discovery.

It is important to keep an eye on the output from the KDD and look for anything suspicious that would indicate some type of internal logic problem. For example, if you wanted a report that outlines the net and gross revenues for each retail store, and instead get a report that states "Bob," there may be an issue you need to look into.

Table 11-2 outlines different types of systems that are used, depending on requirements of the resulting data.

### System Development

Security is most effective if it is planned and managed throughout the life cycle of a system or application versus applying a third-party package as a front end after the development. Many security risks, analyses, and events occur during a product’s lifetime, and these issues should be dealt with from the initial planning stage and continue through the design, coding, implementation, and operational stages. If security is add-

<table>
<thead>
<tr>
<th>Can Process</th>
<th>Data-Based System</th>
<th>Rules-Based System</th>
<th>Knowledge-Based System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>Data</td>
<td>Data</td>
<td></td>
</tr>
<tr>
<td>Rules</td>
<td>Knowledge</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Can Output</th>
<th>Data-Based System</th>
<th>Rules-Based System</th>
<th>Knowledge-Based System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Information</td>
<td>Information</td>
<td>Information decisions</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Real-time decisions</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Answers</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Expert advice</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Recommendations</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Commonly Used For</th>
<th>Data-Based System</th>
<th>Rules-Based System</th>
<th>Knowledge-Based System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hard-coded rules</td>
<td>Enterprise rules</td>
<td>Departmental rules</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Ideal For</th>
<th>Data-Based System</th>
<th>Rules-Based System</th>
<th>Knowledge-Based System</th>
</tr>
</thead>
<tbody>
<tr>
<td>IT/system rules</td>
<td>Simplistic business rules</td>
<td>Complex business rules</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Best for These Types of Applications</th>
<th>Data-Based System</th>
<th>Rules-Based System</th>
<th>Knowledge-Based System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Traditional information systems</td>
<td>Decisioning compliance</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Advising</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Product selection</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Recommending</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Troubleshooting</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Domain Scope</th>
<th>Data-Based System</th>
<th>Rules-Based System</th>
<th>Knowledge-Based System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Broad logic</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deep logic</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table 11-2** Various Types of Systems Based on Capabilities
ed at the end of a project development rather than at each step of the life cycle, the cost and time of adding security increases dramatically. Security should not be looked at as a short sprint, but should be seen as a long run with many hills and obstacles.

Many developers, programmers, and architects know that adding security at a later phase of the system’s life cycle is much more expensive and complicated than integrating it into the planning and design phase. Different security components can affect many different aspects of a system, and if they are thrown in at the last moment, they will surely affect other mechanisms negatively, restrict some already developed functionality, and cause the system to perform in unusual and unexpected ways. This approach costs more money because of the number of times the developers have to go back to the drawing board, recode completed work, and rethink different aspects of the system’s architecture.

Management of Development

Many developers know that good project management keeps the project moving in the right direction, allocates the necessary resources, provides the necessary information, and plans for the worst yet hopes for the best. Project management is an important part of product development, and security management is an important part of project management.

A security plan should be drawn up at the beginning of a development project and integrated into the functional plan to ensure that security is not overlooked. The first plan is broad, covers a wide base, and refers to documented references for more detailed information. The references could include computer standards (RFCs, IEEE standards, and best practices), documents developed in previous projects, security policies, accreditation statements, incident-handling plans, and national or international guidelines (Orange Book, Red Book, and Common Criteria). This helps ensure that the plan stays on target.

The security plan should have a lifetime of its own. It will need to be added to, subtracted from, and explained in more detail as the project continues. It is important to keep it up-to-date for future reference. It is always easy to lose track of actions, activities, and decisions once a large and complex project gets underway.

The security plan and project management activities may likely be audited so security-related decisions can be understood. When assurance in the system needs to be guaranteed, indicating that security was fully considered in each phase of the life cycle, the procedures, development, decisions, and activities that took place during the project will be reviewed. The documentation must accurately reflect how the system or product was built and how it operates once implemented into an environment.

Life-Cycle Phases

*There is a time to live, a time to die, a time to love…*

*Response: And a time to shut up.*

Several types of models are used for system and application development, which include varying life cycles. This section outlines the core components that are common to all of them. Each model basically accomplishes the same thing: the main difference is how the development and lifetime of a system is broken into sections.
A project may start with a good idea, only to have the programmers and engineers just wing it; or, the project may be carefully thought out and structured to follow the necessary life cycles, and the programmers and engineers may stick to the plan. The first option may seem more fun in the beginning, because the team can skip stuffy requirements, blow off documentation, and get the product out the door in a shorter time and under budget. However, the team that takes the time to think through all the scenarios of each phase of the life cycle would actually have more fun, because its product would be more sound and more trusted by the market, and the team would make more money in the long run and would not need to chaotically develop several service and security patches to fix problems missed the first time around.

The different models integrate the following phases in one fashion or another:

- Project initiation
- Functional design analysis and planning
- System design specifications
- Software development
- Installation/implementation
- Operational/maintenance
- Disposal

Security is not listed as an individual bullet point because it should be embedded throughout all phases. Addressing security issues after the product is released costs a lot more money than addressing it during the development of the product. Functionality is the main force driving product development, and several considerations need to take place within that realm, but this section addresses the security issues that must be examined at each phase of the product’s life cycle.

**Project Initiation**

*So what are we building and why?*

This is the phase when everyone involved attempts to understand why the project is needed and what the scope of the project entails. Either a specific customer needs a new system or application or a demand for the product exists in the market. During this phase, the project management team examines the characteristics of the system and proposed functionality, brainstorming sessions take place, and obvious restrictions are reviewed.

A conceptual definition of the project should be initiated and developed to ensure everyone is on the right page and that this is a proper product to develop and will be, hopefully, profitable. This phase could include evaluating products currently on the market and identifying any demands not being met by current vendors. It could also be a direct request for a specific product from a current or future customer.

In either case, because this is for a specific client or market, an initial study of the product needs to be started, and a high-level proposal should be drafted that outlines the necessary resources for the project and the predicted timeline of development. The estimated profit expected from the product also needs to be conducted. This information is submitted to senior management, who will determine whether the next phase should begin or further information is required.
In this phase, user needs are identified and the basic security objectives of the product are acknowledged. It must be determined if the product will be processing sensitive data, and if so, the levels of sensitivity involved should be defined. An initial risk analysis should be initiated that evaluates threats and vulnerabilities to estimate the cost/benefit ratios of the different security countermeasures. Issues pertaining to security integrity, confidentiality, and availability need to be addressed. The level of each security attribute should be focused upon so a clear direction of security controls can begin to take shape.

A basic security framework is designed for the project to follow, and risk management processes are established. Risk management will continue throughout the lifetime of the project. Risk information may start to be gathered and evaluated in the project initiation phase, but it will become more granular in nature as the phases graduate into the functional design and design-specification phase.

**Risk Management**

*Okay, question one. How badly can we screw up?*

One of the most important pieces of risk management is to know the right questions to ask. Risk management was discussed in Chapter 3, but that chapter dealt with identifying and mitigating risks that directly affect the business as a whole. Risk management must also be performed when developing and implementing software. Although the two functions are close in concepts, goals, and objectives, they have different specific tasks and focus.

Software development usually focuses on rich functionality and getting the product out the door and on shelves so customers can buy it as soon as possible. Most of the time, security is not part of the process or it quickly falls by the wayside when a deadline seems imminent. It is not just the programmer who should be thinking about coding in a secure manner, but the design of the product should have security integrated and layered throughout the project. Software engineers should address security threat scenarios and solutions during their tasks. It is not just one faction of a development team that might fall down when it comes to security. Security has never really been treated as an important function of the process—that is, until the product is bought by several customers who undergo attacks and compromises that tie directly to how the product was developed and programmed. Then, security is quite a big deal, but it is too late to integrate security into the project. Instead, a patch is developed and released.

The first step in risk management is to identify the threats and vulnerabilities and to calculate the level of risk involved. When all the risks are evaluated, management will decide upon the acceptable level of risk. Of course, it would be nice for management to not accept any risks and for the product to be designed and tested until it is foolproof; however, this would cause the product to be in development for a long time and to be too expensive to purchase. Compromises and intelligent business decisions must be made to provide a balance between risks and economic feasibility.

**Risk Analysis**

A risk analysis is performed to identify the relative risks and the potential consequences of what a customer can be faced with when using the particular product that is being developed. This process usually involves asking many, many questions to draw up the laundry list of vulnerabilities and threats, the probability of these vulnerabilities being
exploited, and the outcome if one of these threats actually becomes real and a compromise takes place. The questions vary from product to product—such as its intended purpose, the expected environment it will be implemented into, the personnel involved, and the types of businesses that would purchase and use this type of product. The following is a short list of the types of questions that should be asked during a software risk analysis:

- What is the possibility of buffer overflows, and how do we avoid and test for them?
- Does the product properly verify the format/validity of all user-supplied input?
- Are there threat agents outside and inside the environment? What are those threat agents?
- What type of businesses would depend on this product, and what type of business loss would arise if the product were to go offline for a specific period?
- Are there covert channel issues that need to be dealt with?
- What type of fault tolerance is to be integrated into the product, and when would it be initiated?
- Is encryption needed? Which type? What strength?
- Are contingency plans needed for emergency issues?
- Would another party (ISP or hosting agency) be maintaining this product for the customer?
- Is mobile code necessary? Why? And if so, how can it be implemented?
- Will this product be in an environment that is connected to the Internet? What effects could this have on the product?
- Does this product need to interface to vulnerable systems?
- How could this product be vulnerable to Denial-of-Service (DoS) attacks?
- How could this product be vulnerable to viruses?
- Are intrusion alert mechanisms necessary?
- Would there be motivation for insiders or outsiders to sabotage this product? Why? And how could such sabotage be accomplished?
- Would competitor companies of the purchaser want to commit fraud via this product? Why? And how could such fraud be accomplished?
- What other systems would be affected if this product failed?

This is a short list, and each question should branch off into other questions to ensure all possible threats and risks are identified and considered.

Once all the risks are identified, the probability of them actually taking place needs to be quantified, and the consequences of these risks need to be properly evaluated to ensure the right countermeasures are implemented within the development phase and the product itself. If a product will only be used to produce word documents, a lower level of security countermeasures and tests would be needed compared with a product that maintains credit card data.
Many of the same risk analysis steps outlined in Chapter 3 can be applied in the risk analysis that must be performed when developing a product. Once the threats are identified by the project team members, the probability of their occurrence is estimated, and their consequences are calculated, the risks can be listed in order of criticality. If the possibility of a DoS taking place is high and could devastate a customer, then this is at the high end of importance. If the possibility of fraud is low, then this is pushed down the priority list. The most probable and potentially devastating risks are approached first, and the less likely and less damaging are dealt with after the more important risks.

These risks need to be addressed in the design and architecture of the product as well as in the functionality the product provides, the implementation procedures, and the required maintenance. A banking software product may need to be designed to have web server farms within a demilitarized zone (DMZ) of the branch, but have the components and databases behind another set of firewalls to provide another layer of protection. This means the architecture of the product would include splitting it among different systems and developing communications methods between the different parts. If the product is going to provide secure e-mail functionality, then all the risks involved with just this service need to be analyzed and properly accounted for. Implementation procedures need to be thought through and addressed. How will the customer set up this product? What are the system and environment requirements? Does this product need to be supplied with a public key infrastructure (PKI)? The level of maintenance required after installation is important to many products. Will the vendor need to keep the customer abreast of certain security issues? Should any logging and auditing take place? The more these things are thought through in the beginning, the less scrambling will be involved at the end of the process.

It is important to understand the difference between project risk analysis and security risk analysis. They often are confused or combined. The project team may do a risk analysis pertaining to the risk of the project failing. This is much different from the security risk analysis, which has different threats and issues. The two should be understood and used, but in a distinctively different manner.

**Functional Design Analysis and Planning**

*I would like to design a boat to carry my yellow ducky.*

*Response: You are in the wrong meeting.*

In this phase, a project plan is developed by the software architectures to define the security activities and create security checkpoints to ensure quality assurance for security controls takes place and that the configuration and change control process is identified. At this point in the project, resources are identified, test schedules start to form, and evaluation criteria are developed to be able to properly test the security controls. A formal functional baseline is formed, meaning the expectations of the product are outlined in a formal manner, usually through documentation. A test plan is developed, which will be updated through each phase to ensure all issues are properly tested.

Security requirements can be derived from several different sources:

- Functional needs of the system or application
- National, international, or organizational standards and guidelines
- Export restrictions
- The sensitivity level of data being processed (militarily strategic data versus private-sector data)
- Relevant security policies
- Cost/benefit analysis results
- Required level of protection to achieve the targeted assurance level rating

The initial risk assessment will most likely be updated throughout the project as more information is uncovered and learned. In some projects, more than one risk analysis needs to be performed at different stages of the life cycle. For example, if the project team knows the product will need to identify and authenticate users in a domain setting that requires a medium level of security, it will perform an initial risk analysis. Later in the life cycle, if it is determined that this product should work with biometric devices and have the capability to integrate with systems that require high security levels, the project team will perform a whole new risk analysis, because new morsels have been added to the mix.

This phase addresses the functionality required of the product and is captured in a design document. If the product is being developed for a customer, the design document is used as a tool to explain to the customer what the developing team understands to be the requirements of the product. A design document is usually drawn up by analysts, with the guidance of engineers and architects, and presented to the customer. The customer can then decide if more functionality needs to be added or subtracted, after
which the customer and development team can begin hammering out exactly what is expected from the product.

With regard to security issues, this is where high-level questions are asked. Examples of these questions include the following: Are authentication and authorization necessary? Is encryption needed? Will the product need to interface with other systems? Will the product be directly accessed via the Internet?

Many companies skip the functional design phase and jump right into developing specifications for the product. Or a design document is not shared with the customer. This can cause major delays and retooling efforts, because a broad vision of the product needs to be developed before looking strictly at the details. If the customer is not involved at this stage, the customer will most likely think the developers are creating a product that accomplishes X, while the development team thinks the customer wants Y. A lot of time can be wasted developing a product that is not what the customer actually wants, so clear direction and goals must be drawn up before the beginning of coding. This is usually an important function of the project management team.

**System Design Specifications**

Software requirements come from three models:

- **Informational model**  Dictates the type of information to be processed and how it will be processed
- **Functional model**  Outlines the tasks and functions the application needs to carry out
- **Behavioral model**  Explains the states the application will be in during and after specific transitions take place

For example, an antivirus software application may have an informational model that dictates what information is to be processed by the program, such as virus signatures, modified system files, checksums on critical files, and virus activity. It would also have a functional model that dictates that the application should be able to scan a hard drive, check e-mail for known virus signatures, monitor critical system files, and update itself. The behavioral model would indicate that when the system starts up, the antivirus software application will scan the hard drive. The computer coming online would be the event that changes the state of the application. If a virus were found, the application would change state and deal with the virus appropriately. The occurrence of the virus is the event that would change the state. Each state must be accounted for to ensure that the product does not go into an insecure state and act in an unpredictable way.

The informational, functional, and behavioral model data goes into the software design as requirements. What comes out of the design is the data, architectural, and procedural design, as shown in Figure 11-12.

The architects and developers take the data design and the informational model data and transform it into the data structures that will be required to implement the software. The architectural design defines the relationships between the major structures and components of the application. The procedural design transforms structural components into descriptive procedures.
This is the point where access control mechanisms are chosen, subject rights and permissions are defined, the encryption method and algorithm are chosen, the handling of sensitive data is ironed out, the necessary objects and components are identified, the interprocessing communication is evaluated, the integrity mechanism is identified, and any other security specifications are appraised and solutions are determined.

The work breakdown structure (WBS) for future phases needs to be confirmed, which includes the development and implementation stages. This includes a timeline and detailed activities for testing, development, staging, integration testing, and product delivery.

The system design is a tool used to describe the user requirements and the internal behavior of a system. It then maps the two elements to show how the internal behavior actually accomplishes the user requirements.

This phase starts to look at more details of the product and the environment it will be implemented within. The required functionality was determined in the last phase. This phase addresses what mechanisms are needed to provide this functionality and determines how it will be coded, tested, and implemented.

The modularity and reusability of the product, or the product components, need to be addressed. Code that provides security-critical functions should be simple in design, to catch errors in a less confusing fashion, and should be small enough to be fully tested in different situations. Components can be called and used by different parts of the product or by other applications. This attribute—reusability—can help streamline the product and provide for a more efficient and structured coding environment.

The product could have portability issues that need to be dealt with and handled at the early stages of the product development. If the product needs to work on Unix or Windows systems, then different coding requirements are needed compared with a product that will be installed only on mainframes. Also, the environment that will implement this product should be considered. Will this product be used by individual
users, or will all the users within the network access this product in one fashion or another? Whether the product is a single-user product or a multiuser product has large ramifications on the development of the necessary specifications.

The testability of the product and components needs to be thought about at this early phase instead of at later phases. Programmers can code in hooks that show the testers the state of the product at different stages of data processing. Just because the product appears to act correctly and produces the right results at the end of the processing phases does not mean no internal errors exist. This is why testing should happen in modular ways, the flow of data through the product must be followed, and each step should be analyzed.

This phase should look closely at all the questions asked at the project initiation and ensure that specifications are developed for each issue addressed. For example, if authentication is required, this phase will lay out all the details necessary for this process to take place. If fraud is a large risk, then all the necessary countermeasures should be identified, and how they integrate into the product should be shown. If covert channels are a risk, then these issues should be addressed, and pseudocode should be developed to show how covert channels will be reduced or eliminated.

If the product is being developed for a specific customer, the specifications of the product should be shared with the customer to again ensure everyone is still on the same page and headed in the right direction. This is the stage to work out any confusion or misunderstanding before the actual coding begins.

The decisions made during the design phase are pivotal steps to the development phase. The design is the only way customer requirements are translated into software components; thus, software design serves as the foundation, and greatly affects software quality and maintenance. If good product design is not put into place in the beginning of the project, the following phases will be much more challenging.

**Software Development**

*Code jockeys to your cubes and start punching those keys!*

This is the phase where the programmers and developers become deeply involved. They are usually involved up to this point for their direction and advice, but at this phase, it is basically dropped into their laps. Let the programming and testing begin!

This is the stage where the programmers should code in a way that does not permit software compromises. Among other issues to address, the programmers need to check input lengths so buffer overflows cannot take place; inspect code to prevent the presence of covert channels, check for proper data types, make sure checkpoints cannot be bypassed by users, verify syntax, and perform checksums. Different attack scenarios should be played out to see how the code could be attacked or modified in an unauthorized fashion. Debugging and code reviews should be carried out by peer developers, and everything should be clearly documented.

Most programmers do not like to document and will find a way to get out of the task. Six to twelve months later, no one will remember specific issues that were addressed, how they were handled, or the solutions to problems that have already been encountered—or the programmer who knew all the details will have gone to work for a competitor or won the lottery and moved to an island. This is another cause of rework and wasted man-hours. Documentation is extremely important, for many different reasons, and can save a company a lot of money in the long run.
Formal and informal testing should begin as soon as possible. Unit testing can start very early in development. After a programmer develops a component, or unit of code, it is tested with several different input values and in many different situations. Unit testing usually continues throughout the development phase. A totally different group of people should carry out the formal testing. This is an example of separation of duties. A programmer should not develop, test, and release software. The more eyes that see the code and the more fingers that are punching keys, the greater the chance that bugs will be found before the product is released.

Of course, any software hooks inserted for testing or modification purposes need to be removed from the application prior to being released to production because these can easily provide attackers backdoors into the product.

No cookie-cutter recipe exists for security testing, because the applications and products can be so diverse in functionality and security objectives. It is important to map security risks to test cases and code. Linear thinking can be followed by identifying a vulnerability, providing the necessary test scenario, performing the test, and reviewing the code for how it deals with such a vulnerability. At this phase, tests are conducted in an actual network, which should mirror the production environment to ensure the code does not work only in the labs.

Security attacks and penetrations usually take place during this phase to identify any missed vulnerabilities. Functionality, performance, and penetration resistance are evaluated. All the necessary functionality required of the product should be in a checklist to ensure each function is accounted for.

Security tests should be run to test against the vulnerabilities identified earlier within the project. Buffer overflows should be attempted, the product should be hacked and attacked, interfaces should be hit with unexpected inputs, DoS situations should be tested, unusual user activity should take place, and if a system crashes, the product should react by reverting back to a more secure state. The product should be tested in various environments with different applications, configurations, and hardware platforms. A product may respond fine when installed on a clean Windows 2000 installation on a stand-alone PC, but it may throw unexpected errors when installed on a laptop that is remotely connected to a network and has an SMS client installed.

Verification vs. Validation

Verification determines if the product accurately represents and meets the specifications. After all, a product can be developed that does not match the original specifications, so this step ensures the specifications are being properly met.

Validation determines if the product provides the necessary solution for the intended real-world problem. In large projects, it is easy to lose sight of the overall goal. This exercise ensures that the main goal of the project is met.

Separation of Duties

Different environmental types (development, testing, and production) should be properly separated, and functionality and operations should not overlap. Developers should not have access to code used in production. The code should be tested, submitted to the library, and then sent to the production environment.
At this stage, issues found in unit and formal testing are relayed to the development team in problem reports. The problems are fixed and retesting occurs. This is a continual process until everyone is satisfied that the product is ready for production. If there is a specific customer, the customer would run through a range of tests before formally accepting the product. Then the product is formally released to the market or customer.

**NOTE** Sometimes developers enter lines of code in a product that will allow them to do a few keystrokes and get right into the application. This allows them to bypass any security and access controls so they can quickly access the application’s code. This is referred to as a “backdoor” or “maintenance hook” and should be removed before the code goes into production.

**Installation/Implementation**

*This product doesn’t work.*

Response: Try plugging it in.

The implementation stage focuses on how to use and operate the developed system or application. At this phase, the customer has purchased the developed product and installed it into its environment. The product would then be configured for the right level of protection. Functionality and performance tests should be performed, and the results should be analyzed and compared with the company’s security requirements.

The configurations should be documented by the vendor and supplied with the product for the customer to use. User guides and operation and maintenance manuals are developed so users know how to properly use the systems and the technical staff knows how to properly configure the product if needed. Security activities need to be monitored to ensure the system or application performs in the manner promised by the service level agreement.

Accreditation should occur between the implementation and the beginning of operational use of the system or application. This process follows the certification process, which formally or informally tests all the security features to determine if they accomplish the required security needs. Certification is the process of reviewing and evaluating security controls and functionality. It is usually a task assigned to an outside, independent reviewer. (Certification and accreditation were covered in detail in Chapter 5.)

The accreditation is the formal acceptance of the system by management and an explicit acceptance of risk. The accreditation looks at the whole system, not just at an application or a newly upgraded feature, because security is a service that takes place at different layers of the system and can be manifested in many ways. The accreditation process forces the management and technical staff to work together to ensure quality and a level of protection provided by purchased and implemented technologies. The technical staff understand operational and mechanical issues, and the management staff understand mission, financial, and liability issues. Together, they can cover a lot of ground during the certification accreditation processes.

Once management is sure of the security provided by the new system and understands and accepts the risk, it should issue a formal accreditation statement.

Auditing needs to be enabled and monitored, and contingency recovery plans and procedures should be developed and tested to make sure the system and product react as planned in the event of a system failure or emergency situation.
Operational and Maintenance

Okay, the thing is out there. Now someone has to take care of that thing.

When you reach this phase, do not think that security is done and under control, and that all you have to do now is sit back and eat donuts. On the contrary, security is just as, or more, important during the operational phase than during earlier phases.

The initial part of this phase includes configuring the new system and inserting it properly into the network and environment. Many times, security controls are not enabled or properly configured for the environment, so even if they were correctly coded from the beginning, it does not really matter if they are not actually used or are used in an unintended way.

Operational assurance is carried out by continually conducting vulnerability tests, monitoring activities, and auditing events. It is through operational assurance activities that an administrator learns of new vulnerabilities or security compromises, so the proper actions can take place.

If major changes happen to the system, product, or environment, a new risk analysis may need to be performed along with a new certification and accreditation process. These major changes could include adding new systems and/or new applications, relocating the facility, or changing data sensitivity or criticality.

Disposal

All good things must come to an end.

When it is time for “out with the old and in with the new,” certain steps may need to take place to make sure this transition happens in a secure manner. Depending on the sensitivity level of the data held on a system, various disposal activities may be necessary. Information may need to be archived, backed up to another system, discarded, or destroyed. If the data are sensitive and need to be destroyed, they may need to be purged by overwriting, degaussing, or physically destroying the media. It depends on the data and the company’s policy about destroying sensitive information.

If the product being replaced is just a word processor or an antivirus software package, this phase can be easily taken care of. But if the software is integrated into every part of the company’s processing infrastructure, properly extracting it without affecting productivity and security can be an overwhelming task.

Testing Types

If we would like the assurance that the software is any good at all, we should probably test it. There are different types of tests the software should go through because there are different potential flaws we will be looking for. The following are some of the most common testing approaches:

- **Unit testing**  Individual component is in a controlled environment where programmers validate data structure, logic, and boundary conditions.
- **Integration testing**  Verifying that components work together as outlined in design specifications.
- **Acceptance testing**  Ensuring that the code meets customer requirements.
- **Regression testing**  After a change to a system takes place, retesting to ensure functionality, performance, and protection.
When testing software, we not only need to think inside and outside the box, we need to throw the box, kick the box, and use it as a whoopee cushion. It is very hard to think of all the ways users may potentially harm the software product. It is also hard to think of all the ways hackers are going to attempt to break the software. So the following items are just some of the things that should be done to test the software:

- Various types of data should be inputted.
- Data at the different points within acceptable data ranges should be inputted.
  - Perform bounds checking to look for buffer overflows.
  - Conduct data validation to ensure the software is only accepting the type of data it should (that is, numbers and not letters, ASCII and not Unicode, and so on).
- Submit data outside of acceptable ranges.
- Test for different user activities.
- Validate data before and after being processed to identify improper changes.
- Prevent object reuse vulnerabilities.
  - Subject may have unauthorized access to residual data in an object or memory space.

Data can be contaminated in many ways—when it is either going into an application or coming out as output—or both. To ensure the data to be processed are accurate, the following input and output controls should be put into place:

- Input controls
  - Error detection and correction
  - Message digest values
  - Transaction and monetary counts
  - Resubmission and self-validating controls
- Output controls
  - Error handling
  - Reconciliation of values
  - Handling procedures
  - Audit trails and logging
Chapter 11: Application Security

Get This Garbage Out of Here!
Garbage collection is an automated way for the operating system to carry out part of its memory management tasks. A garbage collector identifies blocks of memory that were once allocated but are no longer in use and deallocates the blocks and marks them as free. It also gathers scattered blocks of free memory and combines them into larger blocks. It helps provide a more stable environment and does not waste precious memory.

We’ve already covered some of these basic controls. The bad guys, however, will continually attempt to break through any and all of them in one way or another, and only solid programming and testing will help fortify your product against their evil ways.

Postmortem Review
“Lance did it!” “No, Sue did it!” “Fred did it.” “The boogeyman did it!”

It is important that the team gather after the completion of a project to exhale, talk about the overall project, and discuss things that should be improved for the next time. If this phase is taken seriously and handled properly, the company can save money and time on future projects, because the team can pass along lessons it has learned so mistakes will not be repeated and the process will be streamlined. All of these activities will help the next project run more smoothly with fewer mistakes and in less time.

This should be a structured event in which someone leads the meeting(s) and someone takes notes, but it should also be a relaxed atmosphere so each team member feels comfortable in expressing opinions and ideas. The review must not become a finger-pointing session or a cesspool of complaining. It is a method of looking at the project from an objective view and identifying issues that could be improved upon the next time around.

Some companies do not see the value in this exercise and just race off to start the next project, which will most likely be cursed with the problems of the prior project. Projects are learning processes, and business is about making the best product in the least amount of time for the least amount of money. It is beneficial for management to understand how the two go together and to make sure a postmortem review is part of every project. The most successful businesses streamline the processes of projects and project management and hone them to become a repeatable procedure that produces the expected level of quality. These businesses continually take the time to look at how processes can be improved and built upon.
This chapter explains a life cycle containing seven phases. Other models may use different cycles and a different number of phases, but they accomplish the same basic objectives. Table 11-3 illustrates the NIST’s software development life cycle (SDLC) model.

**System Life-Cycle Phases**

The following are the common phases of system and software development along with the core security tasks that take place at each phase.

- **Project initiation**
  - Conception of project definition
  - Proposal and initial study
  - Initial risk analysis

- **Functional design analysis and planning**
  - Requirements uncovered and defined
  - System environment specifications determined
  - Formal design created

- **System design specifications**
  - Functional design review
  - Functionality broken down
  - Detailed planning put into place
  - Code design

- **Software development**
  - Developing and programming software

- **Installation**
  - Product installation and implementation
  - Testing and auditing

- **Maintenance support**
  - Product changes, fixes, and minor modifications

- **Disposal**
  - Replace product with new product

**Software Development Methods**

Over the years several System Development Methods (SDMs) have been created to meet the various requirements of software developers and vendors. These methods are to help developers through the different stages (analysis, design, programming, mainte-
### Table 11-3  NIST’s Breakdown of the SDLC Model They Provide

<table>
<thead>
<tr>
<th>SDLC</th>
<th>Initiation</th>
<th>Acquisition/Development</th>
<th>Implementation</th>
<th>Operations/ Maintenance</th>
<th>Disposition</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Needs Determination:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>● Perception of a need</td>
<td>● Functional statement of need</td>
<td>● Installation</td>
<td>● Performance measurement</td>
<td>● Appropriateness of disposal</td>
</tr>
<tr>
<td></td>
<td>● Linkage of need to mission and performance objectives</td>
<td>● Market research</td>
<td>● Inspection</td>
<td>● Contract modifications</td>
<td>● Exchange and sale</td>
</tr>
<tr>
<td></td>
<td>● Assessment of alternatives to capital assets</td>
<td>● Feasibility study</td>
<td>● Acceptance testing</td>
<td>● Operations</td>
<td>● Internal organization screening</td>
</tr>
<tr>
<td></td>
<td>● Preparing for investment review and budgeting</td>
<td>● Requirements analysis</td>
<td>● Initial user training</td>
<td>● Maintenance</td>
<td>● Transfer and donation</td>
</tr>
<tr>
<td></td>
<td>(Security Considerations)</td>
<td>● Alternatives analysis</td>
<td>● Documentation</td>
<td></td>
<td>● Contract closeout</td>
</tr>
<tr>
<td></td>
<td>● Security categorization</td>
<td>● Cost-benefit analysis</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>● Preliminary risk assessment</td>
<td>● Software conversion study</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>● Cost analysis</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>● Risk management plan</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>● Acquisition planning</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>● Risk assessment</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>● Security functional requirements analysis</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>● Security assurance requirements analysis</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>● Cost considerations and reporting</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>● Security planning</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>● Security control development</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>● Developmental security test and evaluation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>● Other planning components</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Security Considerations:**
- Security categorization
- Preliminary risk assessment
nance) of program creation. They are commonly referred to as development guidelines, which is in fact what they are. There is a long list of them; the following are a handful:

- **Waterfall** A classical method using discrete phases of development that require formal reviews and documentation before moving into the next phase of the project.
- **Spiral** A method that builds upon the waterfall method with an emphasis on risk analysis, prototypes, and simulations at different phases of the development cycle. This method periodically revisits previous stages to update and verify design requirements.
- **Joint Analysis Development (JAD)** A method that uses a team approach in application development in a workshop-oriented environment.
- **Rapid Application Development (RAD)** A method of determining user requirements and developing systems quickly to satisfy immediate needs.
- **Cleanroom** An approach that attempts to prevent errors or mistakes by following structured and formal methods of developing and testing. This approach is used for high-quality and critical applications that will be put through a strict certification process.

**Computer-Aided Software Engineering**

*Computer-aided software engineering (CASE)* is the use of tools to create and manage software. "CASE tools" is a general term for many types of tools used by programmers, developers, project managers, and analysts that help them make application development faster and with fewer errors. This is because many of the manual tasks are taken care of through automation with the use of CASE tools. Different tools provide managerial, administrative, and technical help in software projects.

The first CASE tools were translators, compilers, assemblers, linkers, and loaders. However, as programming and projects became more complex, the need for more complex tools grew. The tools graduated into program editors, debuggers, code analyzers, and version-control mechanisms. These tools aid in keeping more detailed records of requirements, design, and implementation and in testing the program and project overall. A CASE tool is aimed at supporting one or more software engineering tasks and activities in the process of developing software. It applies engineering principles to the development and analysis of specifications using specific tools.

Many vendors can get their products to the market faster because they are “computer aided.” The CASE tools enable software engineering to be done correctly and fast, relatively speaking.

When the automation covers the complete life cycle of a product, the tools are referred to as *integrated computer-aided software engineering (I-CASE) tools*, and if tools are used for one specific part of the life cycle, then the tools are termed *CASE tools*. 
Prototyping

I would like a prototype of a mini-me.
Response: One of you is enough.

Many times, it is necessary to build for the customer and the developers a model of the gathered requirements of a software product. This model, called a prototype, can show the customer where the development team is headed and its interpretation of the customer’s stated requirements. This enables the customer to agree on the direction the team is headed and get an idea of what the product will look like, or it enables the customer to make changes and further explain any requirements that were uncertain or confusing. The prototype also enables testing to begin earlier in the development process so errors or problems can be uncovered and addressed.

Some projects are very large and may require the product to be partitioned so each part has its own prototype to be reviewed and built upon. In either case, while partitioning or prototyping the full product, an analyst will develop an abbreviated representation of the requirements for the prototype. The software can be created using prototyping tools, which speed up the process. This enables the design to be translated into executable form.

Security testing can begin at an earlier stage if a prototype is developed. Penetration, vulnerability, and data format tests can be performed at each stage of development and with each prototype developed.

If a software prototype is impractical, paper prototypes can be developed where interaction, queries, displays, and logic are shown on paper for the customer to see and the developers to walk through. A series of storyboard sheets can be used to represent each screenshot and the actions that take place behind the screen.

References

- “Software Lifecycles,” Prof. Steve Easterbrook, University of Toronto

Change Control

I am changing stuff left and right.
Response: Was it approved and tested first?

Changes during development or production can cause a lot of havoc if not done properly. Changes could take place for several reasons. During the development phases, a customer may alter requirements and ask that certain functionalities be added, removed, or modified. In production, changes may need to happen because of other changes in the environment, new requirements of a product or system, or newly
released patches or upgrades. These changes should be controlled to make sure they are approved, incorporated properly, and do not affect any original functionality in an adverse way. Change control is the process of controlling the life cycle of an application and documenting the necessary change control activities.

A process for dealing with changes needs to be in place at the beginning of a project so everyone knows how changes are dealt with and what is expected of each entity when a change request is made. Some projects have been doomed from the start because proper change control was not put into place and enforced. Many times in development, the customer and vendor agree on the design of the product, the requirements, and the specifications. The customer is then required to sign a contract confirming this is the agreement, and that if they want any further modifications, they will have to pay the vendor for that extra work. If this is not put into place, then the customer can continually request changes, which requires the development team to put in the extra hours to provide these changes, the result of which is that the vendor loses money and the product does not meet its completion deadline.

Other reasons exist to have change control in place. These reasons deal with organization, standard procedures, and expected results. If a product is in the last phase of development and a change request comes in, the team should know how to deal with it. Usually, the team leader must tell the project manager how much extra time will be required to complete the project if this change is incorporated and what steps need to be taken to ensure this change does not affect other components within the product. In addition, security cannot be compromised, and management must approve the change. If these processes are not controlled, one part of a development team could implement the change without another part of the team being aware of it. This could break some of the other development team’s software pieces. When the pieces of the product are integrated and it is found that some pieces are incompatible, some jobs may be in jeopardy, because management never approved the change in the first place.

The change must be approved, documented, and tested. Some tests may need to be rerun to ensure the change does not affect the product’s capabilities. When a programmer makes a change to source code, it should be done on the test version of the code. Under no conditions should a programmer change the code that is already in production. The changes to the code should be made and tested, and then the new code should go to the librarian. Production code should come only from the librarian and not from a programmer or directly from a test environment.

Official Definitions of Configuration Management

- The procedures used to carry out changes that affect the network, individual systems, or software
- Identifying, controlling, accounting for, and auditing changes made to the baseline trusted computing base (TCB), which includes changes to hardware, software, and firmware
- A system that controls changes and tests documentation through the operational life cycle of a system
Change control should be evaluated during system audits. It is possible to overlook a problem that a change has caused in testing, so the procedures for how change control is implemented and enforced should be examined during a system audit.

The following are some necessary steps for a change control process:

1. Make a formal request for a change.
2. Analyze the request.
   A. Develop the implementation strategy.
   B. Calculate the costs of this implementation.
   C. Review any security implications.
3. Record the change request.
4. Submit the change request for approval.
5. Develop the change.
   A. Recode segments of the product and add or subtract functionality.
   B. Link these changes in the code to the formal change control request.
   C. Submit software for testing and quality approval.
   D. Repeat until quality is adequate.
   E. Make version changes.
6. Report results to management.

The changes to systems may require another round of certification and accreditation. If the changes to a system are significant, then the functionality and level of protection may need to be reevaluated (certified), and management would have to approve the overall system, including the new changes (accreditation).

**The Capability Maturity Model**

The *Capability Maturity Model (CMM)* describes procedures, principles, and practices that underlie software development process maturity. This model was developed to help software vendors improve their development processes by providing an evolutionary path from an ad hoc “fly by the seat of your pants” approach, to a more disciplined and repeatable method that improves software quality, reduces the life cycle of development, provides better project management capabilities, allows for milestones to be created and met in a timely manner, and takes a more proactive approach than the less effective reactive approach.

This model provides policies, procedures, guidelines, and best practices to allow an organization to develop a standardized approach to software development that can be used across many different groups. The goal is to continue to review and improve upon the processes to optimize output, increase capabilities, and provide higher-quality software at a lower cost.

The model offers a layered framework that enables different organizations to implement continuous improvement. It is a tool for the software development company and one for those wanting to assess a vendor’s development consistency and quality. For
example, if the company StuffRUs wants a software development company, SoftwareRUs, to develop an application for it, it can choose to buy into the sales hype about how wonderful SoftwareRUs is, or it can ask for SoftwareRUs to be evaluated against the CMM model. Third-party companies evaluate software development companies to certify organizations’ product development processes. Many software companies have this evaluation done so they can use this as a selling point to attract new customers.

Five maturity levels are used:

- **Initial** Development process is ad hoc or even chaotic. The company does not use effective management procedures and plans. There is no assurance of consistency, and quality is unpredictable.

- **Repeatable** A formal management structure, change control, and quality assurance are in place. The company can properly repeat processes throughout each project. The company does not have formal process models defined.

- **Defined** Formal procedures are in place that outline and define processes carried out in each project. The organization has a way to allow for quantitative process improvement.

- **Managed** The company has formal processes in place to collect and analyze qualitative data, and metrics are defined and fed into the process-improvement program.

- **Optimizing** The company has budgeted and integrated plans for continuous process improvement.
Software Escrow

Will someone keep a copy of my source code?

If a company pays another company to develop software for it, it should have some type of software escrow in place for protection. In a software escrow, a third party keeps a copy of the source code, and possibly other materials, which it will release to the customer only if specific circumstances arise, mainly if the vendor who developed the code goes out of business or for some reason is not meeting its obligations and responsibilities. This procedure protects the customer, because the customer pays the vendor to develop software code for it, and if the vendor goes out of business, the customer otherwise would no longer have access to the actual code. This means the customer code could never be updated or maintained properly.

A logical question would be, “Why doesn’t the vendor just hand over the source code to the customer, since the customer paid for it to be developed in the first place?” It does not work that way. The code is the vendor’s intellectual property. The vendor employs and pays people with the necessary skills to develop that code, and if the vendor were to just hand it over to the customer, it would be giving away its intellectual property, its secrets. The customer gets compiled code instead of source code. Compiled code is code that has been put through a compiler and is unreadable to humans. Most software profits are based on licensing, which outlines what customers can do with the compiled code.

Application Development Methodology

Applications are written in programming code, which tells the operating system and processor what needs to happen to accomplish the user’s requirements when operating a specific application. Programming languages have gone through several generations, each generation building on the next, providing richer functionality and giving the programmers more powerful tools as the languages evolve.

Different types of languages can be used: machine language, assembly language, and high-level languages. Machine language is in a form that the processor can understand and work with directly. Assembly and high-level languages cannot be understood directly by the processor and must be translated, which results in machine language. The process is typically done by a compiler, whose function is to turn human-understandable programming language into machine-understandable language, or object code.

**NOTE** Assembly code will be turned into machine code by an assembler program. Assembly code works at the lower layers of a software architecture. Many drivers contain assembly language components.

Source Code vs. Machine Code

When source code is processed by a compiler, the result is object code, which is written for a specific platform and processor. This object code is the executable form of an application that a user purchases from a vendor. When the object code runs, it is in machine code, which is what the processor actually understands.
When a customer purchases a program, it is in object code form. The program has already been compiled, and it is ready to be executed and set up on the system. The compiler will put it into a form that specific processors can understand. This is why a program that works on a computer with an Alpha processor may not work on a computer with a Pentium processor. The programs work with different processors and operating systems that require different forms of machine language.

If the program was actually sold in the form of original source code, it would have to be compiled on the customer’s computer; thus, the customer must have the correct compiler. Also, original source code would enable competing vendors to view each other’s original ideas and techniques. Source code is considered intellectual property by vendors who develop software, and thus should be highly protected.

Various programs are used to turn high-level programming code (or source code) into object or machine code. These programs are interpreters, compilers, and assemblers. They work as translators.

Interpreters translate one command at a time during execution, and compilers translate large sections of code at a time. Assemblers translate assembly language into machine language. Most applications are compiled, whereas many scripting languages are interpreted.

### Generations of Languages

Program languages have evolved over time to provide programmers and systems with more functionality. The following is the current list of software program generations:

- **Generation One**  Machine language
- **Generation Two**  Assembly language
- **Generation Three**  High-level language
- **Generation Four**  Very high-level language
- **Generation Five**  Natural language

When a customer purchases a program, it is in object code form. The program has already been compiled, and it is ready to be executed and set up on the system. The compiler will put it into a form that specific processors can understand. This is why a program that works on a computer with an Alpha processor may not work on a computer with a Pentium processor. The programs work with different processors and operating systems that require different forms of machine language.

If the program was actually sold in the form of original source code, it would have to be compiled on the customer’s computer; thus, the customer must have the correct compiler. Also, original source code would enable competing vendors to view each other’s original ideas and techniques. Source code is considered intellectual property by vendors who develop software, and thus should be highly protected.

Various programs are used to turn high-level programming code (or source code) into object or machine code. These programs are interpreters, compilers, and assemblers. They work as translators. **Interpreters** translate one command at a time during execution, and **compilers** translate large sections of code at a time. **Assemblers** translate assembly language into machine language. Most applications are compiled, whereas many scripting languages are interpreted.

### Object-Oriented Concepts

*Objects are so cute, and small, and modular. I will take one in each color!*

Software development used to be done by classic input-processing-output methods. This development used an information flow model from hierarchical information structures. Data was input into a program, and the program passed the data from the beginning to end, performed logical procedures, and returned a result.

**Object-oriented programming (OOP)** methods perform the same functionality but with different techniques that work in a more efficient manner. First, you need to understand the basic concepts of OOP.

OOP works with classes and objects. A real-world object, such as a table, is a member (or an instance) of a larger class of objects called “furniture.” The furniture class will have a set of attributes associated with it, and when an object is generated, it inherits these attributes. The attributes may be color, dimensions, weight, style, and cost. These attributes apply if a chair, table, or loveseat object is generated, also referred to as instantiated. Because the table is a member of the class furniture, the table inherits all attributes defined for the class (see Figure 11-13).
The programmer develops the class and all of its characteristics and attributes. The programmer does not develop each and every object, which is the beauty of this approach. As an analogy, let’s say you developed a super-duper coffee maker with the goal of putting Starbucks out of business. A customer punches the available buttons on your coffee maker interface, which is a large latte, with skim milk, vanilla and raspberry flavoring, and an extra shot of espresso, where the coffee is served at 250 degrees. Your coffee maker does all of this through automation and provides the customer with a lovely cup of coffee exactly to her liking. The next customer wants a mocha frappuccino, with whole milk, and extra foam. (Remember the days when we just asked for a cup of coffee and life wasn’t this complicated?) So the goal is to make something once (coffee maker, class), allow it to accept requests through an interface, and create various results (cups of coffee, objects) depending upon the requests submitted.

But how does the class create objects based on requests? A piece of software that is written in OOP will have a request sent to it, usually from another object. The requesting object wants a new object to carry out some type of functionality. Let’s say that object A wants object B to carry out subtraction on the numbers sent from A to B. When this request comes in, an object is built (instantiated) with all of the necessary programming code. Object B carries out the subtraction task and sends the result back to object A. It does not matter what programming language the two objects are written in, what matters is if they know how to communicate with each other. One object can communicate with another object if it knows the application programming code (API) communication requirements. An API is the mechanism that allows objects to talk to each other. Let’s say if I want to talk to you, I can only do it by speaking French and I can only use three phrases or less, because that is all you understand. As long as I follow these rules, I can talk to you. If I don’t follow these rules, I can’t talk to you.

**NOTE** An object is preassembled code that is a self-contained module.
So what’s so great about OOP? If you look at Figure 11-14, you can see the difference between OOP and non-OOP techniques. Non-OOP applications are written as monolithic entities. This means an application is just one big pile of code. If you need to change something in this pile, you would need to go through the whole program’s logic functions to figure out what your one change is going to break. If the program contains hundreds or thousands of lines of code, this is not an easy or enjoyable task. Now, if you choose to write your program in an object-oriented language, you don’t have one monolithic application, but an application that is made up of smaller components (objects). If you need to make changes or updates to some functionality in your application, you can just change the code within the class that creates the object carrying out that functionality and not worry about everything else the program actually carries out. The following breaks down the benefits of OOP:

- **Modularity**
  - Autonomous objects, cooperation through exchanges of messages.
- **Deferred commitment**
  - The internal components of an object can be redefined without changing other parts of the system.
- **Reusability**
  - Refining classes through inheritance.
  - Other programs using same objects.
- **Naturalness**
  - Object-oriented analysis, design, and modeling maps to business needs and solutions.

Most applications have some type of functionality in common. Instead of developing the same code to carry out the same functionality for ten different applications, using OOP allows you to just create the object once and let it be reused in other applications. This reduces development time and saves money.

Now that we understand the concepts, let’s figure out the different terminology used. A **method** is the functionality or procedure an object can carry out. An object may be constructed to accept data from a user and to reformat the request so a back-end server can understand and process it. Another object may perform a method that extracts data from a database and populates a web page with this information. Or an object may carry out a withdrawal procedure to allow the user of an ATM to extract money from her account.

The objects encapsulate the attribute values, which means this information is packaged under one name and can be reused as one entity by other objects. Objects need to be able to communicate with each other, and this happens by using **messages** that are sent to the receiving object’s API. If object A needs to tell object B that a user’s checking account must be reduced by $40, it sends object B a message. The message is made up of the destination, the method that needs to be performed, and the corresponding arguments. Figure 11-15 shows this example.
An object can have a shared portion and a private portion. The shared portion is the interface (API) that enables it to interact with other components. Messages enter through the interface to specify the requested operation, or method, to be performed. The private part of an object is how it actually works and performs the requested operations. Other components need not know how each object works internally—only that it does the job requested of it. This is how **information hiding** is possible. The details of the processing are hidden from all other program elements outside the object. Objects communicate through well-defined interfaces; therefore, they do not need to know how each of their siblings works.

**NOTE** Data hiding is provided by encapsulation, which protects an object’s private data from outside access. No object should be allowed to, or have the need to, access another object’s internal data or processes.

Figure 11-14  Procedural versus object-oriented programming

![Procedural versus object-oriented programming](image)

Figure 11-15  Objects communicate via messages.

![Objects communicate via messages](image)
These objects can grow to great numbers, so the complexity of understanding, tracking, and analyzing can get a bit overwhelming. Many times, the objects are shown in connection to a reference or pointer in documentation. Figure 11-16 shows how related objects are represented as a specific piece, or reference, in an ATM system. This enables analysts and developers to look at a higher level of operation and procedures without having to view each individual object and its code. Thus, this modularity provides for a more easily understood model.

**Abstraction** is the capability to suppress unnecessary details so the important, inherent properties can be examined and reviewed. It enables the separation of conceptual aspects of a system. For example, if a software architect needs to understand how data flows through the program, she would want to understand the big pieces of the program and trace the steps the data takes from first being input into the program all the way until it exits the program as output. It would be difficult to understand this concept if the small details of every piece of the program were presented. Instead, through abstraction, all the details are suppressed so she can understand a crucial part of the product. It is like being able to see a forest without having to look at each and every tree.

Messaging can happen in several ways. Two objects can have a single connection (one to one), a multiple connection (one to many), and a mandatory connection or an optional connection. It is important to map these communication paths to identify if information can flow in a way that is not intended. This will help ensure that sensitive data cannot be passed to objects of a lower security level.

![Diagram of object relationships within a program](image-url)
Each object should have specifications it should adhere to. This discipline provides cleaner programming and reduces programming errors and omissions. The following list is an example of what should be developed for each object:

- Object name
- Attribute descriptions
- Attribute name
- Attribute content
- Attribute data type
- External input to object
- External output from object
- Operation descriptions
- Operation name
- Operation interface description
- Operation processing description
- Performance issues
- Restrictions and limitations
- Instance connections
- Message connections

The developer creates a class that outlines these specifications. When objects are instantiated, they inherit these attributes.

Each object can be reused as stated previously, which is the beauty of OOP. This enables a more efficient use of resources and the programmer's time. Different applications can use the same objects, which reduces redundant work, and as an application grows in functionality, objects can be easily added and integrated into the original structure.

The objects can be catalogued in a library, which provides an economical way for more than one application to call upon the objects (see Figure 11-17). The library provides an index and pointers to where the objects actually live within the system or on another system.

When applications are developed in a modular approach, like object-oriented methods, components can be reused, complexity is reduced, and parallel development can be done. These characteristics allow for fewer mistakes, easier modification, resource efficiency, and more timely coding than the classic information flow models. OOP also provides functional independence, which means each module addresses a specific subfunction of requirements and has an interface that is easily understood by other parts of the application.

An object is encapsulated, meaning the data structure (the operation's functionality) and the acceptable ways of accessing it are grouped into one entity. Other objects, subjects, and applications can use this object and its functionality by accessing it through controlled and standardized interfaces and sending it messages (see Figure 11-18).
**Polymorphism**

Polymorphism is a funny name.  
Response: You are funny looking.

*Polymorphism* comes from the Greek meaning “having multiple forms.” This concept usually confuses people, so let’s jump right into an example. If I developed a program in an OOP language, I can create a variable that can be used in different forms. The application will determine what form to use at the time of execution (run time). So
if my variable is named USERID and I develop the object so the variable can accept either an integer or letters, this provides flexibility. This means the user ID can be accepted as a number (account number) or name (characters). If application A uses this object, it can choose to use integers for the user IDs, while application B can choose to use characters.

What confuses people with this term is that (ISC)² commonly uses the definition or description: “Two objects can receive the same input and have different outputs.” Clear as mud.

As a simplistic example of polymorphism, suppose three different objects receive the input “Bob.” Object A would process this input and produce the output “43-year-old white male.” Object B would receive the input “Bob” and produce the output “Husband of Sally.” Object C would produce the output of “Member of User group.” Each object received the same input, but responded with a different output.

Polymorphism can also take place in the following example: Object A and Object B are created from the same parent class, but Object B is also under a subclass. Object B would have some different characteristics from Object A because of this inheritance from the parent class and the subclass. When Object A and Object B receive the same input, they would result in different outputs because only one of them inherited characteristics from the subclass.

NOTE Polymorphism is when different objects respond to the same command, input, or message in different ways.

From here, the programmer uses OOP to create the components laid out in the design and those that must be tested.
Data Modeling

Let's see. The data went thataway. Oh no, it went thataway. Oops, I lost the data.

The previous paragraphs have provided a simple look at a structured analysis approach. A full-structured analysis approach looks at all objects and subjects of an application and maps the interrelationships, communications paths, and inheritance properties. This is different from data modeling, which considers data independently of the way the data are processed and of the components that process the data. A data model follows an input value from beginning to end and verifies that the output is correct. OOA is an example of a structured analysis approach. If an analyst is reviewing the OOA of an application, she will make sure all relationships are set up correctly, that the inheritance flows in a predictable and usable manner, that the instances of objects are practical and provide the necessary functionality, and that the attributes of each class cover all the necessary values used by the application. When another analyst does a data model review of the same application, he will follow the data and the returned values after processing takes place. An application can have a perfect OOA structure, but when 1 + 1 is entered and it returns −3, something is wrong. This is what the data modeling looks at.

Another example of data modeling deals with databases. Data modeling can be used to provide insight into the data and the relationships that govern it. A data item in one file structure, or data store, might be a pointer to another file structure or to a different data store. These pointers must actually point to the right place. Data modeling would verify this, not OOA structure analysis.

Software Architecture

Software architecture relates the components that make up a software solution to the parts of a real-world problem. Software architects view the application at a higher level than do the programmers, who are focused on data structures, coding rules, variables, and communication paths between objects. An architectural view looks at how the application actually meets and fulfills the requirements recognized and agreed upon in the design phase.

Software architecture involves the process of partitioning requirements into individual problems that can be solved by individual software solutions. This process is the transition phase between the software requirement analysis and the design of the actual components that make up the resulting application.
If the requirements are that the application will scan hard drives and e-mail messages for viruses, the software architecture will break these requirements into individual units that need to be achieved by functionality within that application. These units may include the following functionalities:

- Virus signature storage
- An agent that compares software strings on hard drives to virus signatures
- A process of parsing an e-mail message before the user can view it
- Procedures necessary if data on a hard drive is compressed
- Actions taken if a virus is found
- Actions taken if an e-mail attachment is encrypted

This way of developing a product provides more control and modularity of issues and solutions. If a group of programmers is told to develop an antivirus software package, the group may sit there like deer caught in headlights. However, if one developer is told to write a piece of the program that holds and updates signature files, another developer is told to determine how data on the hard drive will be compared with the signatures within the signature files, and another developer is instructed to program a way to read compressed files, then the programmers will have set goals and start pounding at the keyboards.

Software architects need to provide this type of direction: a high-level view of the application’s objectives and a vision of the overall goals of the project.

**Data Structures**

A **data structure** is a representation of the logical relationship between elements of data. It dictates the degree of association between elements, methods of access, processing alternatives, and the organization of data elements.

The structure can be simple in nature, like the scalar item, which represents a single element that can be addressed by an identifier and accessed by a single address in storage. The scalar items can be grouped in arrays, which provide access by indexes. Other data structures include hierarchical structures by using multilinked lists that contain scalar items, vectors, and possibly arrays. The hierarchical structure provides categorization and association. If a user can make a request of an application to find all computer books written on security, and that application returns a list, then this application is using a hierarchical data structure of some kind. Figure 11-19 shows simple and complex data structures.

**Cohesion and Coupling**

*I do a bunch of stuff and rely on many other modules.*

Response: Low cohesion and high coupling describes you best, then.

**Cohesion** is a term that reflects how many different types of tasks a module can carry out. If a module carries out only one task (subtraction of values) or several tasks that are very similar (subtract, add, multiply), it is described as having high cohesion,
which is a good thing. The higher the cohesion the easier it is to update or modify and not affect other modules that interact with it. This also means the module is easier to reuse and maintain because it is more straightforward when compared to a module with low cohesion. A model with low cohesion carries out multiple different tasks and increases the complexity of the module, which makes it harder to maintain and reuse.

**Coupling** is a measurement that indicates how much interaction one module requires to carry out its tasks. If a module has low (loose) coupling, this means the module does not need to communicate to many other modules to carry out its job. High (tight) coupling means a module depends upon many other modules to carry out its tasks. Low coupling is more desirable because the modules are easier to understand, easier to reuse, and changes can take place and not affect many modules around it. Low coupling indicates that the programmer created a well-structured module. As an analogy, a company would want their employees to be able to carry out their individual jobs with the least amount of dependencies on other workers. If Joe had to talk with five other people just to get one task done, too much complexity exists, it’s too time-consuming, and more places are created where errors can take place.

An example of *low coupling* would be one module passing a variable value to another module. As an example of *high coupling*, Module A would pass a value to Module B, another value to Module C, and yet another value to Module D. Module A cannot complete its tasks until Modules B, C, and D complete their tasks and return results back to Module A.
NOTE Modules should be self-contained and perform a single logical function, which is high cohesion. Modules should not drastically affect each other, which is low coupling.

Distributed Computing

Many of our applications work in a client/server model, which means the smaller part (client) of the application can run on different systems and the larger piece (server) of the application runs on a server. The server portion carries out more functionality and horsepower compared to the clients. The clients will send the server portion requests, and the server will respond with a result. Simple enough, but how do the client and server pieces actually carry out communication with each other?

The three main inter-component communication architectures used today are Common Object Request Broker Architecture (CORBA), Microsoft COM model, and EJB (Enterprise Java Beans). We will be covering these in the next sections.

A distributed object computing model needs to register the client and server components, which means to find out where they live on the network, what their names or IDs are, and what type of functionality the different components carry out. So the first step is basically, "Where are all the pieces, how do I call upon them when I need them, and what do they do?" This organization must be put in place because the coordination between the components should be controlled and monitored, and requests and results must be able to pass back and forth between the correct components.

Life might be easier if we had just one inter-component communication architecture for developers to follow, but what fun would that be? The different architectures work a little differently from each other and are necessary to work in different programming environments. Nevertheless, they all perform the basic function of allowing components on the client and server side to communicate with each other.

CORBA and ORBs

*Has anyone seen my ORB? I need to track down an object.*

If we want components to be able to communicate, this means standardized interfaces and communication methods must be used. This is the only way interoperability can take place.

**Common Object Request Broker Architecture (CORBA)** is an open object-oriented standard architecture developed by the Object Management Group (OMG). It provides interoperability among the vast array of software, platforms, and hardware in environments today. CORBA enables applications to communicate with one another no matter where the applications are located or who developed them.

This standard defines the APIs, communication protocol, and client/server communication methods to allow heterogeneous applications written in different programming languages and run on various platforms to work together. Sounds just lovely.

The OMG developed the CORBA model for the use of these different services in an environment. The model defines object semantics so the external visible characteristics are standard and are viewed the same by all other objects in the environment. This standardization enables many different developers to write hundreds or thousands of
components that can interact with other components in an environment without having to know how the component actually works. The developers know how to communicate with the components because the interfaces are uniform and follow the rules of the model.

In the model, clients request services from objects. The client passes the object a message that contains the name of the object, the requested operation, and any necessary parameters.

The CORBA model provides standards to build a complete distributed environment. It contains two main parts: system-oriented components (object request brokers [ORBs] and object services) and application-oriented components (application objects and common facilities). The ORB manages all communications between components and enables them to interact in a heterogeneous and distributed environment, as shown in Figure 11-20. The ORB works independently of the platforms where the objects reside, which provides greater interoperability.

ORB is the middleware that establishes the client/server relationship between objects. When a client needs to access an object on a server for that object to perform an operation or method, the ORB intercepts the request and is responsible for finding the object. Once the object is found, the ORB invokes a method (or operation), passes the parameters, and returns the result to the client. The client software does not need to know where the object resides nor go through the trouble of finding it. That is the

![Figure 11-20](image-url)

**Figure 11-20** The ORB enables different components throughout a network to communicate and work with each other.
ORB’s job. The objects can be written in different languages and reside on different operating systems and platforms, but the client does not need to worry about any of this (see Figure 11-21).

When objects communicate with each other, they use pipes, which are intercomponent communications services. Different types of pipes are available, such as remote procedure calls (RPCs) and ORBs. ORBs provide communications between distributed objects. If an object on a workstation must have an object on a server process data, it can make a request through the ORB, which will track down the needed object and facilitate the communication path between these two objects until the process is complete. This is the client/server communication pipe used in many networking environments.

ORBs are mechanisms that enable objects to communicate locally or remotely. They enable objects to make requests to objects and receive responses. This happens transparently to the client and provides a type of pipeline between all corresponding objects. Using CORBA enables an application to be usable with many different types of ORBs. It provides portability for applications and tackles many of the interoperability issues that many vendors and developers run into when their products are implemented into different environments.

**COM and DCOM**

Component Object Model (COM) is a model that allows for interprocess communication within one application or between applications on the same computer system. The model was created by Microsoft and outlines standardized APIs, component naming schemes, and communication standards. So if I am a developer and I want my application to be able to interact with the Windows operating system and the different applications developed for this platform, I will follow the COM outlined standards.

![Diagram](image)

The main reason this works is because standardized interfaces are used.

**Figure 11-21** CORBA provides standard interface definitions, which offer greater interoperability in heterogeneous environments.
Distributed Component Object Model (DCOM) supports the same model for component interaction, and also supports distributed IPC. COM enables applications to use components on the same systems, while DCOM enables applications to access objects that reside in different parts of a network. So this is how the client/server-based activities are carried by COM-based operating systems and/or applications.

Without DCOM, programmers would have to write much more complicated code to find necessary objects, set up network sockets, and incorporate the services necessary to allow communication. DCOM takes care of these issues (and more), and enables the programmer to focus on his tasks of developing the necessary functionality within his application. DCOM has a library that takes care of session handling, synchronization, buffering, fault identification and handling, and data format translation.

DCOM works as the middleware that enables distributed processing and provides developers with services that support process-to-process communications across networks (see Figure 11-22).

Other types of middleware provide similar functionality: ORB, message-oriented middleware (MOM), RPC, ODBC, and so on. DCOM provides ORB-like services, data connectivity services, distributed messaging services, and distributed transaction services layered over its RPC mechanism. DCOM integrates all of these functionalities into one technology that uses the same interface as COM (see Figure 11-23).

**Enterprise JavaBeans**

**Enterprise JavaBeans (EJB)** is a structural design for the development and implementation of distributed applications written in Java. EJB provides interfaces and methods to allow different applications to be able to communicate across a networked environment. By using the Internet Inter-ORB Protocol (IIOP), the client portion does not have to be a program written in Java, but can be any valid CORBA client.

---

**NOTE** A Java component is called a Java Bean.

Figure 11-22  Although DCOM provides communication mechanisms in a distributed environment, it still works off of the COM architecture.
The Java Platform Enterprise Edition has several APIs, EJB being just one of them. EJB is used to encapsulate the business logic of an application on the server (in the client/server model). Just as the COM and CORBA models were created to allow a modular approach to programming code with the goal of interoperability, EJB defines a client/server model that is object-oriented and platform independent. The main goal is to have a standardized method of implementing back-end code that carries out business logic for enterprise-wide applications.

**Object Linking and Embedding**

Object linking and embedding (OLE) provides a way for objects to be shared on a local personal computer and to use COM as their foundation. OLE enables objects—such as graphics, pictures, and spreadsheets—to be embedded into documents. In this book, many graphics have been placed throughout the text, and OLE technology was used to do so.

OLE also allows for linking different objects and documents. For example, when Chrissy creates a document that contains a URL, that URL turns blue and is underlined, indicating a user can just double-click it to be taken to the appropriate web site. This is an example of linking capabilities. If Chrissy adds a spreadsheet to her document, this is an instance of embedding. If she needs to edit the spreadsheet, she can double-click the spreadsheet, and the operating system will open the correct environment (which might be Excel) to let her make her changes.

This technology was evolved to work on the World Wide Web and is called ActiveX. The components are like other components, but are meant to be portable. ActiveX components can run on any platform that supports DCOM (using the COM model) or that communicates using DCOM services.

**OLE**

The capability for one program to call another program is called *linking*. The capability to place a piece of data inside a foreign program or document is called *embedding*.
### Distributed Computing Environment

**Distributed Computing Environment (DCE)** is a standard developed by the Open Software Foundation (OSF), also called Open Group. It is basically middleware that is available to many vendors to use within their products. This middleware has the capability to support many types of applications across an enterprise. DCE provides an RPC service, security service, directory service, time service, and distributed file support.

DCE is a set of management services with a communications layer based on RPC. It is a layer of software that sits on the top of the network layer and provides services to the applications above it. DCE and DCOM offer much of the same functionality. DCOM, however, was developed by Microsoft and is more proprietary in nature.

DCE’s time service provides host clock synchronization and enables applications to determine sequencing and to schedule events based on this clock synchronization. This time synchronization is for applications. Users cannot access this functionality directly. The directory service enables users, servers, and resources to be contacted anywhere on the network. When the directory service is given the name, it returns the network address of the resource along with other necessary information. DCOM uses a *globally unique identifier* (GUID), while DCE uses a *universal unique identifier* (UUID). They are both used to uniquely identify users, resources, and components within an environment. DCE is illustrated in Figure 11-24.

The RPC function collects the arguments and commands from the sending program and prepares them for transmission over the network. RPC determines the network transport protocol to be used and finds the receiving host’s address in the directory service. The thread service provides real-time priority scheduling in a multithreading environment. The security services support authentication and authorization services. The

![Diagram](image-url)

**Figure 11-24** DCE provides many services, which are all wrapped into one technology.
distributed file service (DFS), on the other hand, provides a single integrated file system that all DCE users can use to share files. This is important because many environments have different operating systems that cannot understand other file systems. However, if DCE is being used, a DCE local file system exists alongside the native file system.

**Expert Systems and Knowledge-Based Systems**

*Hey, this computer is smarter than me!*

*Response: No surprise there.*

Expert systems, also called knowledge-based systems, use artificial intelligence (AI) to solve problems.

AI software uses nonnumerical algorithms to solve complex problems, recognize hidden patterns, prove theorems, play games, mine data, and help in forecasting and diagnosing a range of issues. The type of computation done by AI software cannot be accomplished by straightforward analyses and regular programming logic techniques.

Expert systems emulate human logic to solve problems that would usually require human intelligence and intuition. These systems represent expert knowledge as data or rules within the software of a system, and this data and these rules are called upon when it is necessary to solve a problem. Knowledge-based systems collect data of human know-how and hold it in some type of database. These fragments of data are used to reason through a problem.

A regular program can deal with inputs and parameters only in the ways in which it has been designed and programmed. Although a regular program can calculate the mortgage payments of a house over 20 years at an 8-percent interest rate, it cannot necessarily forecast the placement of stars in 100 million years because of all the unknowns and possible variables that come into play. Although both programs—a regular program and an expert system—have a finite set of information available to them, the expert system will attempt to “think” like a person, reason through different scenarios, and provide an answer even without all the necessary data. Conventional programming deals with procedural manipulation of data, whereas humans attempt to solve complex problems using abstract and symbolic approaches.

A book may contain a lot of useful information, but a person has to read that book, interpret its meaning, and then attempt to use those interpretations within the real world. This is what an expert system attempts to do.

Professionals in the field of AI develop techniques that provide the modeling of information at higher levels of abstraction. The techniques are part of the languages and tools used, which enable programs to be developed that closely resemble human logic. The programs that can emulate human expertise in specific domains are called expert systems.

**Expert Systems**

An expert system is a computer program containing a knowledge base and a set of algorithms and rules used to infer new facts from data and incoming requests.
Rule-based programming is a common way of developing expert systems. The rules are based on if-then logic units and specify a set of actions to be performed for a given situation. This is one way expert systems are used to find patterns, which is called pattern matching. A mechanism, called the inference engine, automatically matches facts against patterns and determines which rules are applicable. The actions of the corresponding rules are executed when the inference engine is instructed to begin execution.

Huh? Okay, let's say Dr. Gorenz is puzzled by a patient's symptoms and is unable to match the problems the patient is having to a specific ailment and find the right cure. So he uses an expert system to help him in his diagnosis. Dr. Gorenz can initiate the expert system, which will then take him through question-and-answer scenarios. The expert system will use the information gathered through this interaction, and it will go step by step through the facts, looking for patterns that can be tied to known diseases, ailments, and medical issues. Although Dr. Gorenz is very smart and one of the top doctors in his field, he cannot necessarily recall all possible diseases and ailments. The expert system can analyze this information because it is working off of a database that has been stuffed full of medical information that can fill up several libraries.

As the expert system goes through the medical information, it may see that the patient had a case of severe hives six months ago, had a case of ringing in the ears and blurred vision three months ago, and has a history of diabetes. The system will look at the patient’s recent complaints of joint aches and tiredness. With each finding, the expert system digs deeper, looking for further information, and then uses all the information obtained and compares it with the knowledge base available to it. In the end, the expert system returns a diagnosis to Dr. Gorenz that says the patient is suffering from a rare disease found only in Brazil that is caused by a specific mold that grows on bananas. Because the patient has diabetes, his sensitivity is much higher to this contaminant. The system spits out the necessary treatment. Then, Dr. Gorenz marches back into the room where the patient is waiting and explains the problem and protects his reputation of being a really smart doctor.

The system not only uses a database of facts, but also collects a wealth of knowledge from experts in a specific field. This knowledge is captured by using interactive tools that have been engineered specifically to capture human knowledge. This knowledge base is then transferred to automated systems that help in human decisions by offering advice, free up experts from repetitive routine decisions, ensure decisions are made in a consistent and quick manner, and allow a company to retain its organization’s expertise even as employees come and go.

An expert system usually consists of two parts: an inference engine and a knowledge base. The inference engine handles the user interface, external files, scheduling, and program-accessing capabilities. The knowledge base contains data pertaining to a specific problem or domain. Expert systems use the inference engine to decide how to execute a program or how the rules should be initiated and followed. The inference engine of a knowledge-based system provides the necessary rules for the system to take the original facts and combine them to form new facts.

The systems employ AI programming languages to allow for real-world decision making. The system is built by a knowledge system builder (programmer), a knowledge engineer (analyst), and subject matter expert(s), as shown in Figure 11-25. It is built on facts, rules of thumb, and expert advice. The information gathered from the expert(s)
during the development of the system is kept in a knowledge base and is used during the question-and-answer session with the end user. The system works as a consultant to the end user and can recommend several alternative solutions by considering competing hypotheses at the same time.

Expert systems are commonly used to automate a security log review for an IDS.

**NOTE** Expert systems use automatic logical processing, inference engine processing, and general methods of searching for problem solutions.

### Artificial Neural Networks

An artificial neural network (ANN) is an electronic model based on the neural structure of the brain. Computers perform activities like calculating large numbers, keeping large ledgers, and performing complex mathematical functions, but they cannot recognize patterns or learn from experience as the brain can. ANNs contain many units that stimulate neurons, each with a small amount of memory. The units work on data that are input through their many connections. Via training rules, the systems are able to learn from examples and have the capability to generalize.

The brain stores information in forms of patterns. People have the ability to recognize another person’s face from several different angles. Each angle of that person’s face is made up of a complex set of patterns. Even if only half the face is seen or the face is viewed in a shadow or dark lighting, the human brain can insert the missing pieces, and people can recognize their friends and acquaintances. A computer that uses conventional programming and logic would have to have each piece of the pattern and the full face in one particular angle to be able to match or recognize it.
The brain uses neurons to remember, think, apply previous experiences, perform logic, and recognize patterns. The capacity of the brain comes from the large number of neurons and the multiple connections between them. The power of the brain is a result of genetic programming and learning.

ANNs try to replicate the basic functions of neurons and their circuitry to solve problems in a new way. They consist of many simple computational neural units connected to each other. An input value is presented to one, some, or all the units, which in turn perform functions on the data.

The brain has clusters of neurons that process in an interactive and dynamic way. Biologically, neurons have no restrictions of interconnections among themselves; therefore, a neuron can have thousands of connections. In addition, the neurons in the brain work in a three-dimensional world, whereas the electronic units in ANNs have a physical limitation on the possible number of connections and thus operate in a two-dimensional world.

Like the brain, the ANN’s real power comes from its capability to learn. Within the brain, when something is learned and used often, the connection path to where that information is stored is strengthened to provide quicker access. This is why sometimes you know something but “can’t put your finger on it.” This means there is no pronounced pathway to the information stored somewhere in the brain. If a person is asked her phone number, she can rattle it off without any real energy. But if you ask her who her third-grade teacher was, it might take more time and energy. Both facts are held within her brain, but the phone number has a more pronounced connection path and comes to mind more quickly and easily. In an ANN, a connection between two units that are often activated might be strengthened, which is a form of learning.

It is known that when something happens to someone in a highly emotional state, that person is more likely to remember specifics about a situation or incident. If Joyce had a surprise party on her 35th birthday, which was filled with fun and emotion, she will most likely remember that birthday more than her 36th birthday, when her husband only bought her a card. But she and her husband will probably remember her 45th birthday, when her husband forgot her birthday, because the fight that occurred was full of emotion and activity. The reason some memories are more vivid than others is because more emotion is tied to them, or more weight is assigned to them. In ANNs, some inputs have higher weights assigned to them than other inputs, which amplifies the meaning or importance of the inputs, just like emotion does with human beings.

NOTE Decisions by ANNs are only as good as the experiences they are given.

Intuition is a hard quality to replicate in electrical circuits and logic gates. Fuzzy logic and other mathematical disciplines are used for intuition, forecasting, and intelligent guessing. These approaches work with types of probability within mathematics and memberships of different sets. One simple example of the use of fuzzy logic is a washing machine that has built-in intelligence. When a person puts a load of clothes in the washing machine and the tank fills with water, the machine using fuzzy logic sends a beam from one part of the tank to the other. Depending on how much light actually
was received at the other end, it can determine how dirty the clothes are because of the density of the dirt in the water. Additional tests can be used to see if the dirt is oily or dry and to check other relevant attributes. The washing machine takes in all this information and estimates the right temperature and the right amount of laundry soap to use. This provides a more efficient way of washing, by saving on water and soap and washing the clothes at the right temperature for the right amount of time. The washing machine does not necessarily know all the facts or know that the information it gathered is 100-percent correct, but it can make guesses that will be pretty close to reality.

ANNs are programmed with the capability to decide and to learn to improve their functionality through massive trial-and-error decision making.

Fuzzy logic is necessary because a regular computer system does not see the world in shades of gray. It cannot differentiate between good, bad, few, and many. Fuzzy logic is an approach to enable the computer to use these vague values that mean something to humans but nothing to computers.

Stock market forecasting and insurance and financial risk assessments are examples of where fuzzy logic can be used and is most beneficial. They require a large number of variables and decision-making information from experts in those particular domains. The system can indicate which insurance or financial risks are good or bad without the user having to input a stack of conditions, if-then statements, and variable values.

Conventional systems see the world in black and white and work in a sea of precision and decimal points. Fuzzy logic enables the computer to incorporate imprecision into the programming language, which opens up a whole new world for computing and attacking complex issues. Neural network researchers attempt to understand more of how the brain works and of nature’s capabilities so newly engineered ANN solutions can solve more complex problems than could be solved through traditional computing means.

**Web Security**

When it comes to the Internet and web-based applications, many situations are unique to this area. Rarely are threats of vandalism an issue in typical computing environments. Also, the potential risk for fraud is higher due to the universal availability of these applications over the Internet. The reason we are using the Internet is to expose our product or service to the widest possible audience. We smartly put these web servers in the DMZ so those who access these servers don’t have direct access to our other internal servers. One of the unfortunate issues when using web-based applications is that you need to allow the Internet to access them in order for them to function, so you must open up the ports related to the Web (80 and 443) on your firewall—so now any attack that can come through on these ports is “game on.”

The applications themselves are somewhat mysterious to the purveyors of the Internet as well. If you want to sell your homemade pies via the Internet, you’ll typically need to display them in graphic form and allow some form of communication for questions (via e-mail or online chat). You’ll need some sort of shopping cart if you want to actually collect money for your pies, and typically you’ll have to deal with interfacing with shipping and payment processing channels…all of this from someone who just wanted to sell pies! If you are a master baker, you probably aren’t a webmaster, so now you’ll have to rely on someone else to set up your web site and load the appropriate
applications on it. Should you develop your own PHP- or JAVA-based application? The benefits could be wonderful, having a customized application that would further automate your business, but the risks of developing an in-house application (especially if it’s your first time) are great if you haven’t developed the methodology, development process, QA, and change control, as well as identified the risks and vulnerabilities… All of this to sell pies? Now you understand why those nice ladies and gents sell them by the roadside…no web application headaches!

The alternative to developing your own web application is using an off-the-shelf variety instead. Many commercial and free options are available for nearly every e-commerce need. These are written in a variety of languages, by a variety of entities, so now the issue is “Whom should we trust?” Do these developers have the same processes in place that you would have used yourself? Have these applications been developed and tested with the appropriate security in mind? Will these applications introduce any vulnerabilities along with the functionality they provide? Does your webmaster understand the security implications associated with the web application he suggests you use on your site for certain functionality? These are the problems that plague not only those wanting to sell homemade pies on the Internet, but also financial institutions, auction sites, and everyone who is involved in e-commerce.

With all of these issues in mind, let’s try to define the most glaring threats associated with having a web server connected to the Internet.

**Vandalism**

This attack usually involves replacing the legitimate graphics and titles on a web site with ones modified by the attacker. You may wonder why vandalism is a threat to your web server. Actually, many script-kiddies are hacking systems solely for bragging rights. Messing up your pie shop may not be as impressive to the hacking elite as defacing a .gov or .mil site, but any site must be guarded. Public perception is also at risk here: even though technically astute customers can understand that the hacker only replaced a graphic on the front page of your site, everyone else believes that hackers accessed the entire customer database. Remember: perception is reality!

**Financial Fraud**

Money is a strong motivator to those who would like something for nothing. Whenever transactions are involved, the potential for fraud is present, especially in an anonymous environment like the Internet. The same people who wouldn’t dare steal a newspaper from a broken vending machine in public find it easy to rationalize the theft of web-based goods and services over the Internet.

**Privileged Access**

The web and its players are distributed all over the planet. You can have employees making pies in Tulsa, a webmaster who lives in Singapore, and your servers hosted in London. This being the case, there has to be a mechanism for remote administrative
access, which introduces the risk of someone other than an authorized administrator gaining access to your system. Once an attacker gains access in this way, you can no longer trust the system, the logs, or the transaction information.

**Theft of Transaction Information**

To collect money, ship goods, and identify one customer from another, you will have to collect and store data. How you collect and store this information will ultimately tell the tale of your success. Of course, transaction information will be a target for hackers who would like to steal identities, sell credit card information to organized crime rings, or simply use the collected information themselves to commit fraud.

**Theft of Intellectual Property**

Once a web server is compromised, that system can be used to attack the internal network and any system connected to it. This puts an attacker one hop from databases and file stores that may reveal company secrets. Can you afford to lose your competitive advantage?

**Denial-of-Service (DoS) Attacks**

One of the oldest attacks in the hacker’s repertoire is the Denial-of-Service attack. This is the simple but effective technique of overwhelming a system or service with requests that tie up resources to the point that legitimate requests cannot be fulfilled. Some attacks in this genre will cause a system or service to fail in an uncontrolled manner and stop processing—not a good state for a web server.

As mentioned before, web servers and the applications that run on them are widely accessible due to the fact that we want people to access them. Although in the past there have been highly publicized cases of vulnerabilities in web servers themselves (Microsoft’s IIS 4.0, for example), the majority of the hacks being performed today are exploiting the web applications running on top of the web servers at the application level. To compound the problem, usually so much traffic is traveling to web servers that logging (and, of course, reviewing) all of the pertinent information is exhausting for most organizations. Firewalls will allow traffic coming in on port 80 to your web server because this is required. Some webmasters believe that using SSL (Secure Sockets Layer), via port 443, for all connections will protect them in some way. This will encrypt the connections of legitimate users and protect them from sniffing attacks, but using SSL will also encrypt the attacker’s traffic from any intrusion detection systems and do nothing to protect the web application itself. If we employed intrusion detection in the DMZ, first of all, it would be a full-time job. Second, if it’s a standard network-based IDS, it wouldn’t provide much help anyway. This is not to say that logging, SSL, firewalls, and IDSs shouldn’t be deployed; it’s simply that each of these must be evaluated for its effectiveness in your organization’s “defense in depth” strategy.

In addition to the countermeasures already mentioned, consider use of the following as safeguards for mitigating security risks to your web-based applications.
Create a Quality Assurance Process

A quality assurance process is quite effective in ensuring that the servers that host your web apps are installed and configured properly. Even the most secure web application can be compromised if the underlying operating system has vulnerabilities. This process should include all aspects of the server, from the operating system to installed patches, the web server, and the removal of unwanted services, documentation, and libraries. To verify that the system meets required criteria, the execution of a web and network scan should be completed before deployment of the system.

Web Application Firewalls

Unlike traditional firewalls that only look at destination/source addresses and port numbers, application-layer firewalls do deep packet inspection, which means they are capable of looking for and blocking specific attack behaviors or anomalies in protocol verbs like HTTP’s POST command.

Intrusion Prevention Systems

An Intrusion Prevention System (IPS), as opposed to an Intrusion Detection System, can actually prevent attacks it identifies. These types of systems are typically placed inline, which is to say that all traffic must be evaluated and passed through the IPS before it can reach the servers behind it. This raises a concern of performance, which typically raises the cost and hardware requirements. These systems are considered by some to be extensions of IDS systems.

Implement SYN Proxies on the Firewall

If you search the Internet, you will find that the most common Denial-of-Service attacks are SYN floods. A SYN flood is where attackers send fake SYN requests to servers in an attempt to exhaust the amount of legitimate requests the server can maintain. If the attacker is successful, the server will wait for a predetermined timeout period for the bogus connections to complete (which, of course, never do), and most legitimate requests will be ignored by the server.

By implementing a SYN proxy on the firewall, now the firewall can manage the connections to the server. If a predefined threshold of SYN requests (let’s say 500 in a second) occurs, the firewall is on guard. If requests continue to come in, the firewall has the option of dropping the oldest requests that haven’t resulted in an established connection, thereby allowing legitimate connections to make their way to be processed by the server. Not all firewalls support this option, but those which support connection “state tables” usually will.

All of these solutions are wonderful, but at the core of all of this is the web-based application itself. True web security should start with designing and deploying secure application services and allowing the other controls to mitigate the risk. Now we will dive deeper into some of the specific threats and vulnerabilities associated with this topic.
Specific Threats for Web Environments

The most common types of vulnerabilities, threats, and complexities are covered in the following sections, which we will attack one at a time:

- Information gathering
- Administrative interfaces
- Authentication and access control
- Configuration management
- Input validation
- Parameter validation
- Session management

Information Gathering

Information gathering is usually the first step in an attacker’s methodology. Information gathered may allow an attacker to infer additional information that can be used to compromise systems. Unfortunately, most of the information gathered is from sources that are available to anyone who asks. The big search engines make it even easier for an attacker to gather information because they aggregate information and can return results from the search engine’s cache without the attacker ever connecting to the target company’s web server.

The majority of the culprits to information disclosure are developers and web server administrators who are just trying to do their jobs. The comments in the HTML source code put in by the developer to explain a routine or the backup files that have been stored on the web server by the admin aren’t glaring security issues, but if they can be accessed by unauthorized users, they could reveal much more than an organization would normally allow. Even the error messages returned by the server when an improper request is made can contain physical paths to the database, version numbers, and so on, which can be interpreted by an attacker as a foothold to gain unauthorized access to a system.

More sophisticated attackers go beyond the search engines to explore the contents of all accessible files on a server for possible clues to the structure of the internal network or the connection string used by the web server to connect to the database server.

In order for a web server to provide the active content and common interfaces that web users demand these days, the servers must access data sources, process code, and return the results to the web clients. To employ these mechanisms, the appropriate code must be written and presented to the web browser in the appropriate format. One technology called server side includes (SSI) allows web developers to reuse content by inserting the same content into multiple web documents. This typically involves use of an include statement in the code and a file (.inc) that is to be included. However, if these files are able to be accessed by an attacker, the code would be visible and could be changed to “include” other files containing sensitive information. Other technologies like Active Server Pages (ASP) (pages that have an .asp file extension) are used to provide an “active” user environment. These files can disclose any contained sensitive code
if they were able to be viewed. Developers should avoid using any sensitive code in your SSI file or ASP files (like your database connection strings or some proprietary business logic) so in the event the document should ever find itself in anyone’s hands unparsed by the server, the code isn’t readily available. There have been too many vulnerabilities with these types of files in the past to assume they will not be able to be read.

Another tip that will allow developers to avoid exposing the physical location or passwords used to connect to a database is to use a Data Source Name (DSN). This is a logical name for the data store, rather than the drive letter and directory location of the database, that can be used when programming to the Open Database Connectivity (ODBC) interface. When an ODBC DSN is used to store these values, they are stored in the Registry of the system, not in the code itself. This technique actually makes the code easier to modify, since the connection strings are a variable stored in the Registry, so it would be a good best practice.

The countermeasures to information gathering techniques that attackers use are to be aware of the information you are making available to the public and limit its availability to only the minimal amount necessary. Developers should be aware of the potential of their code to be viewed by someone outside of the organization, and administrators should routinely check search engines for references to their web sites, e-mail addresses, file types, and data stores. Many web sites and entire books are dedicated to information gathering via publicly available databases, so it would be a good example of due diligence to check these out.

**Administrative Interfaces**

Everyone wants to work from the coffee shop or at home in their pajamas. Webmasters and web developers are particularly fond of this concept. Although some systems mandate that administration be carried out from a local terminal, in most cases there is an interface to administer the systems remotely, even over the Web. While this may be “wicked cool” to the webmaster, it also provides an entry point into the system for an unauthorized user.

Since we are talking about the Web, using a web-based **administrative interface** is in most opinions a bad idea. If we’ve identified the vulnerability and are willing to accept the risk, the administrative interface should be at least (if not more) secure as the web application or service we are hosting.

A bad habit that happens even in high-security environments is hard coding authentication credentials into the links to the management interfaces, or enabling the “remember password” option. This does make it easier on the administrator but offers up too much access to someone who stumbles across the link regardless of their intentions.

So, let’s face the facts, most commercial software and web application servers install some type of administrative console by default. Knowing this and remembering the information gathering techniques we’ve already spoken about should be enough for organizations to take this threat seriously. If the interface is not needed, it should be disabled. When custom applications are developed, the existence of management interfaces is less known, so consideration should be given to this in policy and procedures.

The simple countermeasure for this threat requires that the management interfaces be removed, but this may upset your administrators. Using a stronger authentication
mechanism would be better than the standard username\password scenario. Controlling which systems are allowed to connect and administer the system is another good technique. Many systems allow specific IP addresses or network IDs be defined that will only allow administrative access from these stations.

Ultimately, the most secure management interface for a system would be one that is out-of-band, meaning a separate channel of communication is used to avoid any vulnerabilities that may exist in the environment that the system operates in. An example of out-of-band would be using a modem connected to a web server to dial in directly and configure it using a local interface, as opposed to connecting via the Internet and using a web interface. This should only be done through an encrypted channel, as in SSH.

Authentication and Access Control
If you’ve used the Internet for banking, shopping, registering for classes, or working from home, I’d be willing to bet you’ve had to log in through a web-based application. From the consumer side or the provider side, the topic of authentication and access control is an obvious issue. Consumers want an access control mechanism that provides the security and privacy they would expect from a trusted entity, but they also don’t want to be too burdened by the process. From the service providers’ perspective, they want to provide the highest amount of security to the consumer that performance, compliance, and cost will allow. So, from both of these perspectives, typically usernames and passwords are still used to control access to most web applications.

The problem with using passwords to authenticate users on a web site is probably the same reason you use the Internet to deliver your service in the first place: accessibility. Accessibility is great if all the people accessing your site are legitimate users. Accessibility isn’t that great when everyone on the planet who’s inclined to attempt unauthorized access to your site can anonymously give it a shot. Passwords don’t really prove much. They are used because they are cheap and reasonably effective, but they really don’t prove that the user “jsmith” is really John Smith; they just prove that the person using the account jsmith has typed in the correct password. That could be anyone! Have you ever used anyone else’s account for anything? Tell the truth!

It wouldn’t be a stretch to think of a system that held sensitive information (medical, financial, and so on) to be an identified target for attackers. Mining usernames via search engines or simply using common usernames (like jsmith) and attempting to log in to these sites is very common. If you’ve ever signed up at a web site for access to download a “free” document or file, what username did you use? Is it the same one you use for other sites? Maybe even the same password? Crafty attackers might be mining information via other web sites that seem rather friendly, offering to evaluate your IQ and send you the results, or enter you into a sweepstakes. Remember that untrained, unaware users are an organization’s biggest threat. Beware!

Another weakness in authentication (especially when passwords are used) is the fact that illegitimate (as well as legitimate) users can lock out the account after a threshold of bad logon attempts are made. This is a good policy to help prevent password guessing and brute force attacks against your system. As you probably know, brute force attacks attempt every possible combination of characters to get into a system, and web applications are just as vulnerable as other systems. The countermeasure of account
lockout does, in fact, keep this attack in check, but if this type of attack attempts to log in as every user account on the system, it effectively locks all users out. What would be the impact if this was a financial institution and now 100,000 users suddenly needed the Help Desk to assist them with unlocking their accounts? An administrative nightmare! This may also lead to an evaluation of your organization’s account creation and password reset policy. How do you authenticate a user who has lost their password? What do you reset the password to? When an account is created, do you use a default password? All of these should be defined in policy and procedure. Remember, you are only as secure as your weakest link.

The solution to the massive account lockout DoS attack could be to only lock out the account for a limited amount of time—30 minutes for low risk sites to three hours or even a day depending on the amount of risk. Ultimately, your organization must determine what level of risk you’re willing to accept. Using a multifactor authentication mechanism won’t necessarily stop these types of attacks either, but it will make the success of unauthorized access less likely.

Log files should be analyzed to determine the offending system or systems, although these will seldom be the actual machines belonging to the attacker. Such analysis will allow your organization to see where the attempts are originating from and adjust the access rules of firewalls and systems accordingly.

Finally, a best practice would be to exchange all authentication information via a secure mechanism. This will typically mean using encryption of the password and credentials or securing the channel of communication. These days, it would seem silly to have to remind web sites that the benefits of using SSL (when your http:// changes to https://) are well worth the price you pay for the server certificate required and the processing related to the encryption/decryption process on each end. Some large sites, however, still don’t use encrypted authentication mechanisms and have exposed themselves to the threat of attackers sniffing usernames and passwords.

**Configuration Management**

*Configuration management* is simply the concept of managing the configuration of your systems. The default accounts and their passwords, the sample files on the system, and the management interfaces should all be identified and a security baseline defined in a web environment. Before a system goes live in production, there should be a verification process to ensure compliance with the policy. This is, of course, a “perfect world” scenario.

In the real-world arena of web application development, the process may only mean that the requirements for the application are specified, a developer sets up a “test” environment that emulates the production environment (with the same operating system and web server installed), and the developer writes the code, tests the application for functionality, and passes it off to whomever she reports to, whereupon the application is put into production without another thought. This isn’t the section of this book where we should go into the security that surrounds sound development practices, but to iterate briefly: a common mistake is taking the test environment (since everything worked in the test) and making that the production environment. The application may certainly work, but the developer almost certainly set up the test environ-
ment without the baseline security levels that would be used in the production environment. Too often the company is enticed to get an application online with a focus on availability rather than integrity or confidentiality. This (erroneous) school of thought would be “Hey, let’s see if it works and then we’ll lock it down,” but this lockdown may be forgotten, and too often this leads to an eventual system compromise.

If an organization doesn’t have their own in-house development team and simply acquires an application they want to put into production, there should still be a process in place to identify vulnerabilities and verify that the application is implemented securely. Applications that have been installed with the default configurations are rarely secure. Many applications have default administrative accounts and passwords that are widely known by the hacker community. The presence of error pages or the sample files that are installed to help with an implementation are signs of weakness to attackers. You may be wondering how an attacker would be able to detect these things on your web server: The answer is your favorite search engine.

Search engines (like Google) are really good at analyzing web pages for the links in them and then following all of the links, cataloguing everything they find along the way. This is a great feature for those of us who use Internet search engines to find recipes, directions, or movie quotes, but hackers using craftily articulated search requests can easily reveal management interfaces, error pages that show misconfigurations, and even areas of your web site that you may not have wanted the world to see.

The solution to all of these threats is to have a defined policy and procedures. The procedures should involve removing default configurations and applying the security patches from vendors in a timely fashion. Another smart approach would be when an application goes from testing to production, all of the systems’ access control lists (ACLs) should be reviewed to verify that the appropriate security is applied so the most pristine system files are protected. Finally, remove anything and everything on your system that you would not want accessible to the world via search engines. This would include system documentation, internal files, sample files, and default error pages.

Input Validation
Web servers aren’t that smart, they just do what they are told to. They are designed to process requests via a certain protocol. The term protocol means the rules that are followed in a certain situation to ensure appropriate communication. When a person sits at their web browser and types in a request for http://www.website.com/index.htm, they are using a protocol called Hypertext Transfer Protocol (HTTP) to request the file “index.htm” from the server “www” in the “website.com” namespace. A request in this form is called a Uniform Resource Locator (URL), and it’s pretty close to the way we speak—well, at least we can read it. Like many situations in computerland, there is more than one way to request something because computers speak several different “languages”—like binary, hexadecimal, and many other coding mechanisms—each of which is interpreted and processed by the system as valid commands. Validating that these requests are allowed is part of input validation and is usually tied to some coded validation rules. The fact that these rules have to be coded means that it’s possible some sneaky requests may slip through the coded validation rules.
Some sneaky examples follow:

- **Path or directory traversal**  This attack is also known as the "dot dot slash" because it is perpetrated by inserting the characters "../" several times into a URL to back up or traverse into directories that weren’t supposed to be accessible from the Web. The command "../" at the command prompt tells the system to back up to the previous directory (try it, "cd ../"). If a web server’s default directory was "c:\inetpub\www", a URL requesting http://www.website.com/scripts/../../../../../windows/system32/cmd.exe/?/c+dir+c:\ would issue the command to back up several directories to ensure it has gone all the way to the root of the drive and then make the request to change to the operating system directory (windows\system32) and run the cmd.exe listing the contents of the c: drive.

- **Unicode encoding**  Unicode is an industry standard mechanism developed to represent the entire range of over 100,000 textual characters in the world as a standard coding format. Web servers support Unicode to support different character sets (like Chinese), and, at a time, many supported it by default. So, even if we told our systems to not allow the "../" directory traversal request mentioned earlier, an attacker using Unicode could effectively make the same directory traversal request without using "/", but with any of the Unicode representation of that character (three exist: %c1%1c, %c0%9v, and %c0%af). That request may slip through unnoticed and be processed.

- **URL encoding**  If you've ever noticed that a "space" appears as "%20" in a URL in a web browser (Why is it only me who notices that?), the "%20" represents the space because spaces aren’t allowed characters in a URL. Much like the attacks using Unicode characters, attackers found that they could bypass filtering techniques and make requests by representing characters differently.

Besides just serving static files to users, almost every web application is going to have to accept some input. When you use the Web, you are constantly asked to input information like usernames, passwords, and credit card information. To a web application, this input is just data that is to be processed like the rest of the code in the application. Usually, this input is used as a variable and fed into some code that will process it based on some logic like IF [username input field]=X AND [password input field]=Y THEN Authenticate. This will function well assuming there is always correct information put into the input fields, but what if the wrong information is input? Developers have to cover all the angles. They have to assume that sometimes the wrong input will be given, and they have to handle that situation appropriately. To deal with this, a routine is usually coded in that will tell the system what to do if the input isn’t what was expected.

The buffer overflow is probably the most notorious of input validation mistakes. A buffer is an area reserved by an application to store something in it, like some user input. After the application receives the input, an instruction pointer points the application to do something with the input that’s been put in the buffer. A buffer overflow occurs when an application erroneously allows an invalid amount of input to be written into the buffer area, overwriting the instruction pointer in the code that told the program what to do with the input. Once the instruction pointer is overwritten, what-
ever code has been placed in the buffer can then be executed, all under the security context of the application.

Client-side validation is when the input validation is done at the client before it is even sent back to the server to process. If you’ve missed a field in a web form before and after clicking Submit, you immediately receive a message informing you that you’ve forgotten to fill in one of the fields. Here, you’ve experienced client-side validation. This is a good idea, rather than sending incomplete requests to the server and the server having to kick back the error. The problem arises when the client-side validation is the only validation that takes place. In this situation, the server trusts that the client has done its job correctly and processes the input as if it is valid. In normal situations, accepting this input would be fine, but when an attacker can intercept the traffic between the client and server and modify it or just directly make illegitimate requests to the server without using a client, a compromise is more likely.

In an environment where input validation is weak, an attacker will try to input specific operating system (OS) commands into the input fields instead of what the system was expecting (like the username and password) in an effort to trick the system into running them. Remember that computers do what they’re told, and if an attacker can get them to run an OS command, they will be able to execute them as if they were the application. If the web application is written to access a database, as most are, there is the threat of SQL injection, where instead of valid input the attacker puts actual database commands into the input fields, which are then parsed and run by the application. SQL (Structured Query Language) statements can be used by attackers to bypass authentication and reveal all records in a database.

Remember that different layers of a system (see Figure 11-26) all have their own vulnerabilities that must be identified and fixed.

A similar sounding attack is cross-site scripting, which, in the security community, has replaced buffer overflows as the biggest threat in web applications. The term cross-site scripting (XSS) refers to an attack where a vulnerability is found on a web site that allows an attacker to inject malicious code into a web application. The malicious code can then be executed in the browsers of unsuspecting users as they access the site. Turning off all scripting would fix this vulnerability, but would break a lot of web applications.

All of the attacks in this section have the related issue of erroneously assuming what input data is possible, the effects that specially encoded data has on an application, and believing the input that is received is always valid. The countermeasures to all of these would be to filter out all “known” malicious requests, never trust information coming from the client without first validating it, and implement a strong policy to include appropriate parameter checking in all applications.

**Parameter Validation**

The issue of parameter validation is akin to the issue of input validation mentioned earlier. **Parameter validation** is where the values that are being received by the application are validated to be within defined limits before the server application processes them within the system. The main difference between parameter validation and input validation would have to be whether the application was expecting the user to input a value as opposed to an environment variable that is defined by the application. Attacks
in this area deal with manipulating values that the system would assume are beyond the client being able to configure, mainly because there isn’t a mechanism provided in the interface to do so. We know that we should never assume—don’t make me say the saying—and this is especially true when dealing with computers because they lack the common sense we’ve been endowed with as humans.

In an effort to provide a rich end-user experience, web application designers have to employ mechanisms to keep track of the thousands of different web browsers that could be connected at any given time. The HTTP protocol by itself doesn’t really facilitate managing the state of a user’s connection, it really just connects to a server, gets whatever objects (the .htm file, graphics, and so forth) are requested in the HTML code (HTTP’s Markup Language), and then just disconnects or times out. If the browser does,
in fact, disconnect or time out, how does the server know how to recognize it when it returns? Would you be irritated if you had to re-enter all of your information again because you spent too long looking at possible flights while booking a flight online? Since most people would, web developers employ the technique of passing a cookie to the client to help the server remember things about the state of the connection. A cookie isn’t a program, just some data that are passed and stored in memory (called a session cookie), or locally as a file (called a persistent cookie), to pass state information back to the server. An example of how cookies are employed would be a shopping cart application used on a commercial web site. As you put items into your cart, they are maintained by updating a session cookie on your system. You may have noticed the “Cookies must be enabled” message that some web sites issue as you enter their site.

Since accessing a session cookie in memory is usually beyond the reach of most users, most web developers didn’t think about this as a serious threat when designing their systems. It is not uncommon for web developers to enable account lockout after a certain number of unsuccessful login attempts have occurred (something we talked about earlier). If a developer is using a session cookie to keep track of how many times a client has attempted to log in, there may be a vulnerability here. If an application didn’t want to allow more than three unsuccessful logins before locking a client out, the server might pass a session cookie to the client, setting a value like “number of allowed logins = 3”; after each unsuccessful attempt, the server would tell the client to decrement the “number of allowed logins” value. When the value reaches 0, the client would be directed to a “Your account has been locked out” page.

A web proxy is a piece of software installed on a system that is designed to intercept all traffic between the local web browser and the web server. Using freely available web proxy software (like Achilles or Burp Proxy), an attacker could monitor and modify any information as it travels in either direction. In the preceding example, when the server tells the client via a session cookie that the “number of allowed logins = 3,” if that information is intercepted by an attacker using one of these proxies and changes the value to “number of allowed logins = 50000,” this would effectively allow a brute force attack on a system if the system has no other validation mechanism in place.

Using a web proxy can also exploit the use of hidden fields in web pages. Just like it sounds, a hidden field is not shown in the user interface but contains a value that is passed to the server when the web form is submitted. The exploit of using hidden values can occur when a web developer codes the prices of items on a web page as hidden values instead of referencing the items and their prices on the server. The attacker uses the web proxy to intercept the submitted information from the client and changes the value (the price) before it gets to the server. This is surprisingly easy to do and, assuming no other checks are in place, would allow the perpetrator to see the new values specified in the e-commerce shopping cart.

The countermeasure that would lessen the risk associated with these threats would be adequate parameter validation. Adequate parameter validation may include pre-validation and post-validation controls. In a client-server environment, pre-validation controls may be placed on the client side, prior to submitting requests to the server. Even when these are employed, the server should perform parallel pre-validation of input
prior to application submission because a client will have fewer controls than a server, and may have been compromised or bypassed.

- **Pre-validation** Input controls verifying data is in appropriate format, and compliant with application specifications, prior to submission to the application. An example of this would be form field validation, where web forms do not allow letters in a field that is expecting to receive a number (currency) value.

- **Post-validation** Insuring an application’s output is consistent with expectations (that is, within pre-determined constraints of reasonableness).

**Session Management**

As highlighted earlier, managing several thousand different clients connecting to a web-based application is a challenge. The aspect of session management requires consideration before delivering applications via the Web. Commonly, the most used method of managing client sessions is by assigning unique session IDs to every connection. A session ID is a value sent by the client to the server with every request that uniquely identifies the client to the server or application. In the event that an attacker was able to acquire or even guess an authenticated client’s session ID and render it to the server as its own session ID, the server would be fooled and the attacker would have access to the session.

The old “never send anything in clear text” rule certainly applies here. HTTP traffic is unencrypted by default and does nothing to combat an attacker sniffing session IDs off the wire. As session IDs are usually passed in, and maintained, via the HTTP protocol, they should be protected in some way.

An attacker being able to predict or guess the session IDs would also be a threat in this type of environment. Using sequential session IDs for clients would be a mistake. Random session IDs of an appropriate length would counter session ID prediction. Building in some sort of time stamp or time-based validation will combat replay attacks. (A replay attack is simply an attacker capturing the traffic from a legitimate session and replaying it to authenticate his session.) Finally, any cookies that are used to keep state on the connection should also be encrypted.

**Mobile Code**

Code that can be transmitted across a network, to be executed by a system or device on the other end, is called mobile code. There are many legitimate reasons to allow mobile code—for example, web browser applets that may execute in the background to download additional content for the web page, such as plug-ins that allow you to view a video.

The cautions arise when a web site downloads code intended to do malicious or compromising actions, especially when the recipient is unaware that the compromising activity is taking place. If a web site is compromised, it can be used as a platform from which to launch attacks against anyone visiting the site and just browsing. On a web browser, having security settings set to high, or disallowing various scripting or active web components, may be an appropriate countermeasure. Some of the common types of mobile code are covered in the next sections.
Java

Java is an object-oriented, platform-independent programming language. It is employed as a full-fledged programming language and is used to write complete programs and short programs, called applets, which run in a user's browser.

Other languages are compiled to object code for a specific operating system and processor. This is why a particular application may run on Windows but not on Macintosh. An Intel processor does not necessarily understand machine code compiled for an Alpha processor and vice versa. Java is platform independent because it creates intermediate code, bytecode, which is not processor-specific. The Java Virtual Machine (JVM) then converts the bytecode to the machine code that the processor on that particular system can understand (see Figure 11-27). Let's quickly walk through these steps:

1. A programmer creates a Java applet and runs it through a compiler.
2. The Java compiler converts the source code into bytecode (non-processor-specific).
3. The user downloads the Java applet.
4. The JVM converts the bytecode into machine-level code (processor-specific).
5. The applet runs when called upon.

When an applet is executed, the JVM will create a virtual machine within an environment called a sandbox. This virtual machine is an enclosed environment in which the applet carries out its activities. Applets are commonly sent over within a requested web page, which means the applet executes as soon as it arrives. It can carry out malicious activity on purpose or accidentally if the developer of the applet did not do his part correctly. So the sandbox strictly limits the applet’s access to any system resources. The JVM mediates access to system resources to ensure the applet code behaves and stays within its own sandbox. These components are illustrated in Figure 11-28.

**NOTE** The Java language itself provides protection mechanisms as in garbage collection, memory management, validating address usage, and a component that verifies adherence to predetermined rules.

![Bytecode](image1)

**Figure 11-27** The JVM interprets bytecode to machine code for that specific platform.
However, as with many other things in the computing world, the bad guys have figured out how to escape their confines and restrictions. Programmers have figured out how to write applets that enable the code to access hard drives and resources that are supposed to be protected by the Java security scheme. This code can be malicious in nature and cause destruction and mayhem to the user and her system.

**Browser Settings**

Java applets and the actions they perform can be prevented and controlled by specific browser settings. These settings do not affect full-fledged Java applications running outside of the browser.

![Java’s security model](image-url)
ActiveX

ActiveX is a Microsoft technology composed of a set of OOP technologies and tools based on COM and DCOM. A programmer uses these tools to create ActiveX controls, which are self-sufficient programs similar to Java applets. ActiveX controls can be reused by many applications within one system, or different systems, on the network. These controls can be downloaded from web sites to add extra functionality (as in providing animations for web pages), but they are also components of Windows operating systems themselves (dynamic link libraries [DLLs]) and carry out common operating system tasks.

Instead of trying to keep ActiveX controls in a safe area for various computations and activities, this technology practices security by informing the user where the program came from. The user can decide whether to trust this origin or not.

ActiveX technology provides security levels and authentication, letting users control the security of the ActiveX components they download. Unlike Java applets, ActiveX components are downloaded to a user’s hard drive when he chooses to add the functionality the component provides. This means the ActiveX component has far greater access to the user’s system compared to Java applets.

The security level setting of the user’s browser dictates whether an ActiveX component is downloaded automatically or whether the user is first prompted with a warning. The security level is configurable by the user via his browser controls. As the security level increases, so too does the browser’s sensitivity level to signed and unsigned components and controls, and to the initialization of ActiveX scripts.

The main security difference between Java applets and ActiveX controls is that Java sets up a sandbox for the code to execute in, and this restricts the code’s access to resources within the user’s computer. ActiveX uses Authenticode technology, which relies on digital certificates and trusting certificate authorities. (Signing, digital certificates, and certificate authorities are explained in detail in Chapter 8.) Although both are good and interesting technologies, they have inherent flaws. Java has not been able to ensure that all code stays within the sandbox, which has caused several types of security compromises. These instances are examples of malware. ActiveX doesn’t necessarily provide security—in fact, it often presents annoying dialog boxes to users. Since most users do not understand this technology, they continually click OK because they don’t understand the risks involved.

Malicious Software (Malware)

Several types of malicious code or malware exist, such as viruses, worms, Trojan horses, and logic bombs. They usually are dormant until activated by an event the user or system initiates. They can be spread by e-mail, sharing media (jump drives), sharing documents and programs, or downloading things from the Internet, or they can be purposely inserted by an attacker.

Adhering to the usual rule of not opening an e-mail attachment that comes from an unknown source is one of the best ways to combat malicious code. However, recent viruses and worms have infected personal e-mail address books, so this precaution is not a sure thing to protect systems from malicious code. If an address book is infected and used during an attack, the victim gets an e-mail message that seems to have come from a person he knows. Because he knows this person, he will proceed to open the
e-mail message and double-click the attachment. And Bam! His computer is now infected and uses the e-mail client’s address book to spread the virus to all his friends and acquaintances.

Antivirus software should be installed to watch for known virus signatures, and host intrusion detection software can be used to watch for suspicious activity, file access, and changes to help detect evildoers and their malicious activity.

Malicious code can be detected through the following clues:

- File size increase
- Many unexpected disk accesses
- A change in an update or modified timestamp
- A sudden decrease of hard drive space
- Unexpected and strange activity by applications
- A sudden increase in network activity

The following section quickly looks at a few types of malicious code.

**Viruses**

A virus is a small application, or string of code, that infects applications. The main function of a virus is to reproduce, and it requires a host application to do this. In other words, viruses cannot replicate on their own. A virus infects files by inserting or attaching a copy of itself to the file. The virus may also cause destruction by deleting system files, displaying graphics, reconfiguring systems, or overwhelming mail servers. Several viruses have been released that achieved self-perpetuation by mailing themselves to every entry in a victim’s personal address book. The virus masqueraded as coming from a trusted source. The ILOVEYOU, Melissa, and Naked Wife viruses are older viruses that used the programs Outlook and Outlook Express as their host applications and were replicated when the victim chose to open the message.

Macros are programs written in Word Basic, Visual Basic, or VBScript and are usually used with Microsoft Office products. Macros automate tasks that users would otherwise have to carry out themselves. Users can define a series of activities and common tasks for the application to perform when a button is clicked, instead of doing each of those tasks individually. A **macro virus** is a virus written in one of these macro languages and is platform independent. They infect and replicate in templates and within documents. Macro viruses are common because they are extremely easy to write, and Office products are in wide use.

Some viruses infect the boot sector (**boot sector viruses**) of a computer and either move data within the boot sector or overwrite the sector with new information. Some boot sector viruses have part of their code in the boot sector, which can initiate the virus, and the rest of their code in sectors on the hard drive it has marked off as bad. Because the sectors are marked as bad, the operating system and applications will not attempt to use those sectors; thus, they will not get overwritten.

Other types of viruses append themselves to executables on the system and compress them by using the user’s permissions (**compression viruses**). When the user chooses to use that executable, the system automatically decompresses it and the malicious code, which usually causes the malicious code to initialize and perform its dirty deeds.
What Is a Virus?

A virus is a program that searches out other programs and infects them by embedding a copy of itself. When the infected program executes, the embedded virus is executed, which propagates the infection.

A stealth virus hides the modifications it has made to files or boot records. This can be accomplished by monitoring system functions used to read files or sectors and forging the results. This means that when an antivirus program attempts to read an infected file or sector, the original uninfected form will be presented instead of the actual infected form. The virus can hide itself by masking the size of the file it is hidden in or actually move itself temporarily to another location while an antivirus program is carrying out its scanning process.

So a stealth virus is a virus that hides its tracks after infecting a system. Once infected, the virus can make modifications to make the computer appear the same as before. The virus can show the original file size of a file it infected instead of the new, larger size to try to trick the antivirus software into thinking no changes have been made.

A polymorphic virus produces varied but operational copies of itself. This is done in the hopes of outwitting a virus scanner. Even if one or two copies are found and disabled, other copies may still remain active within the system.

The polymorphic virus can use different encryption schemes requiring different decryption routines. This would require an antivirus scan for several scan strings, one for each possible decryption method, in order to identify all copies of this type of virus.

These viruses can also vary the sequence of their instructions by including noise, or bogus instructions, with other useful instructions. They can also use a mutation engine and a random-number generator to change the sequence of their instructions in the hopes of not being detected. A polymorphic virus has the capability to change its own code, enabling the virus to have hundreds or thousands of variants. These activities can cause the virus scanner to not properly recognize the virus and to leave it alone.

A multipart virus infects both the boot sector of a hard drive and executable files. The virus first becomes resident in memory and then infects the boot sector. Once it is in memory, it can infect the entire system.

A self-garbling virus attempts to hide from antivirus software by garbling its own code. As the virus spreads, it changes the way its code is formatted. A small portion of the virus code decodes the garbled code when activated.

Meme viruses are not actual computer viruses but types of e-mail messages that are continually forwarded around the Internet. They can be chain letters, e-mail hoax virus alerts, religious messages, or pyramid selling schemes. They are replicated by humans, not software, and can waste bandwidth and spread fear. Several e-mails have been passed around describing dangerous viruses, even though the viruses weren’t real. People believed the e-mails and felt as though they were doing the right thing by passing them along to tell friends about this supposedly dangerous malware, when really the people were duped and were themselves spreading a meme virus.

Macro Languages

Macro languages enable users to edit, delete, and copy files. Because these languages are so easy to use, many more types of macro viruses are possible.
Script viruses have been quite popular and damaging over the last few years. Scripts are files that are executed by an interpreter—for example, Microsoft Windows Script Host, which interprets different types of scripting languages. Web sites have become more dynamic and interactive through the use of script files written in Visual Basic (VBScript) and Java (Jscript), and other scripting languages that are embedded in HTML. When a web page that has these scripts embedded is requested by a web browser, these embedded scripts are executed and if they are malicious, then everything just blows up. Okay, this is a tad overdramatic. The virus will carry out the payload (instructions) that the virus writer has integrated into the script, whether it is sending out copies of itself to everyone in your contact list or deleting critical files. Scripts are just another infection vector used by malware writers to carry out their evil ways.

NOTE

The LoveLetter virus is one of the most well-known viruses because of the amount of damage it caused. It was written in VBScript.

Another type of virus (tunneling virus) attempts to install itself under the antivirus program. When the antivirus goes around doing its health check on critical files, file sizes, modification dates, and so on, it makes a request to the operating system to gather this information. Now, if the virus can put itself between the antivirus and the operating system, when the antivirus sends out a command (system call) for this type of information, the tunneling virus can intercept this call. Instead of the operating system responding to the request, the tunneling virus responds with information that indicates that everything is fine and healthy and that there is no indication of any type of infection.

NOTE

An EICAR test is done with antivirus software by introducing a string that all antivirus products recognize as hostile so that testing can be conducted. Antivirus software products have an EICAR.com file and a signature that matches this file. After software configurations are completed, you then put this file on the system to test the antivirus product’s reactions to a virus.

Malware Components

It is common for malware to have six main elements, although it is not necessary for them all to be in place.

- **Insertion**  Installs itself on the victim’s system
- **Avoidance**  Uses methods to avoid being detected
- **Eradication**  Removes itself after the payload has been executed
- **Replication**  Makes copies of itself and spreads to other victims
- **Trigger**  Uses an event to initiate its payload execution
- **Payload**  Carries out its function (that is, deletes files, installs a backdoor, exploits a vulnerability, and so on)
**Botnets**

A “bot” is short for “robot” and is a piece of code that carries out functionality for its master, who is the author of this code. Bots are a type of malware and are being installed on thousands of computers even now as you’re reading this sentence. It is a piece of dormant code, also known as a zombie, that is used to forward items sent to it. Items sent to bots can be spam, viruses, pornography, or attack code. The reason to send these types of items through bots is to help ensure that the original sender will not be located and identified.

Hackers compromise thousands of systems with this zombie code through many different methods: e-mail attachments, compromised web sites, links embedded in e-mail, Trojan horses, and so on. The zombie code sends a message to the hacker indicating that a specific system has been compromised and the system is now available to be used by the attacker as she wishes. When a hacker has a collection of these compromised systems, it is referred to as a botnet (network of bots). The hacker can use all of these systems to carry out powerful DDoS attacks or even rent these systems to spammers.

The owner of this botnet (commonly referred to as the bot herder) controls the systems remotely, usually through the internet relay chat (IRC) protocol.

The common steps of the development and use of a botnet are listed next:

1. A hacker sends out malicious code that has the bot software as its payload.
2. Once installed, the bot logs in to an IRC or web server that it is coded to contact. The server then acts as the controlling server of the botnet.
3. A spammer pays the hacker to use these systems and sends instructions to the controller server, which causes all of the infected systems to send out spam messages to mail servers.

Spammers use this method so their messages have a higher likelihood of getting through mail server spam filters since the sending IP addresses are those of the victim’s system. Thus, the source IP addresses change constantly. This is how you are constantly updated on the new male enhancement solutions and ways to purchase Viagra.

Figure 11-29 illustrates the life cycle of a botnet. The botnet herder works with, or pays, hackers to develop and spread malware to infect systems that will become part of the botnet. Whoever wants to tell you about a new product they just released, carry out identity theft, or conduct attacks, and so on, will pay the herder to use the botnet for their purposes.

The Symantec Internet Security Threat Report indicated that during the first six months of 2006, there were 4,696,903 active botnet computers. Mostly these were infected home computers that participated in botnets without even knowing it.

**Worms**

Worms are different from viruses in that they can reproduce on their own without a host application, and are self-contained programs. A worm can propagate itself by using e-mail, web-site downloads, and more. The definitions of a worm and virus are continually merging, and the distinction is becoming more blurred. The ILOVEYOU program was a worm. When the user executed an e-mail attachment, several processes were spawned automatically. The worm was copied and sent to all addresses within the
victim’s address book. Some files on the hard drive were deleted and replaced. If these were opened, the worm self-propagation started again. ILOVEYOU acts as a virus by requiring the use of an e-mail client, like Outlook, and works as a worm by reproducing itself when the user opens infected files that reside on his hard drive.

**Logic Bombs**

A *logic bomb* executes a program, or string of code, when a certain event happens or a date and time arrives. For example, if a user accesses her bank account software, a logic bomb may be initiated, and a program may be triggered to copy the user’s account number and transaction codes. Another example is when a user accesses the Internet through his cable modem. This action can initiate a planted bomb that sends a message to an attacker over the Internet to let him know the user is online and in position for an attack.

**Trojan Horses**

A *Trojan horse* is a program that is disguised as another program. For example, a Trojan horse can be named Notepad.exe and have the same icon as the regular Notepad program. However, when a user executes Notepad.exe, the program can delete system files. Trojan horses perform a useful functionality in addition to the malicious functionality in the background. So the Trojan horse named Notepad.exe may still run the Notepad program for the user, but in the background it will manipulate files or cause other malicious acts. A host-based IDS can be configured to watch certain files and detect when they grow in size, which is often a sign of a Trojan horse. If the original Notepad.exe
was 50KB in size and then grew to 2MB, it may indicate that a Trojan horse has infected that program.

Remote Access Trojans (RATs) are malicious programs that run on systems and allow intruders to access and use a system remotely. They mimic the functionality of legitimate remote control programs used for remote administration, but are used for sinister purposes instead of helpful activities. They are developed to allow for stealth installation and operation, and are usually hidden in some type of mobile code, such as Java applets or ActiveX controls, that are downloaded from web sites.

Several RAT programs are available to the hacker (Back Orifice, SubSeven, Netbus, and others). Once it is loaded on the victim’s system, the attacker can download or upload files, send commands, install zombie software, and use the compromised system as he pleases.

Antivirus Software

Traditional antivirus software use signatures to detect malicious code. The signature is a fingerprint created by the antivirus vendor. The signature is a sequence of code that was extracted from the virus itself. Just like our bodies have antibodies that identify and go after a specific type of foreign material, an antivirus software package has an engine that uses these signatures to identify malware. The antivirus software scans files, e-mail messages, and other data passing through specific protocols, and then compares them to its database of signatures. When there is a match, the antivirus software carries out whatever activities it is configured to do, which can be to quarantine the file, attempt to clean the file (remove the virus), provide a warning message dialog box to the user, and/or log the event.

Signature-based detection (also called fingerprint detection) is an effective way to detect malicious software, but there is a delayed response time to new threats. Once a virus is detected, the antivirus vendor must study it, develop and test a new signature, release the signature, and all customers must download it. If the malicious code is just sending out silly pictures to all of your friends, this delay is not so critical. If the malicious software is similar to the Slammer worm, this amount of delay can be devastating.

NOTE

The Slammer worm was released in 2003. It took advantage of a buffer overflow within the Microsoft SQL Server 2000 software and caused excessive Denial-of-Service attacks. Several documented accounts have estimated the resulting damages to industry to be over $1 billion.

Since new malware is released daily, it is hard for antivirus software to keep up. The technique of using signatures means this software can only detect viruses that have been identified and where a signature is created. Since virus writers are prolific and busy beasts and because viruses can morph, it is important that the antivirus software have other tricks up its sleeve to detect malicious code.

Another technique that almost all antivirus software products use is referred to as heuristic detection. This approach analyzes the overall structure of the malicious code, evaluates the coded instructions and logic functions, and looks at the type of data within the virus or worm. So, it collects a bunch of information about this piece of code and
assesses the likelihood of it being malicious in nature. It has a type of “suspiciousness counter,” which is incremented as the program finds more potentially malicious attributes. Once a predefined threshold is met, the code is officially considered dangerous and the antivirus software jumps into action to protect the system. This allows antivirus software to detect unknown malware, instead of just relying on signatures.

As an analogy, let’s say Barney is the town cop who is employed to root out the bad guys and lock them up (quarantine). If Barney was going to use a signature method, he would compare a stack of photographs to each person he sees on the street. When he sees a match, he quickly throws the bad guy into his patrol car and drives off. If he was going to use the heuristic method, he would be watching for suspicious activity. So if someone with a ski mask was standing outside a bank, Barney would assess the likelihood of this being a bank robber against it just being a cold guy in need of some cash.

**CAUTION** Diskless workstations are still vulnerable to viruses, even though they do not have a hard disk and a full operating system. They can still get viruses that load and reside in memory. These systems can be rebooted remotely (remote booting) to bring the memory back to a clean state, which means the virus is “flushed” out of the system.

Some antivirus products create a simulated environment, called a virtual machine or sandbox, and allows some of the logic within the suspected code to execute in the protected environment. This allows the antivirus software to see the code in question in action, which gives it more information as to whether it is malicious or not.

**NOTE** The virtual machine or sandbox is also sometimes referred to as an emulation buffer. They are all the same thing, a piece of memory that is segmented and protected so that if the code is malicious, the system is protected.

Reviewing information about a piece of code is called **static analysis**, while allowing a portion of the code to run in a virtual machine is called **dynamic analysis**. They are both considered heuristic detection methods.

**Immunizers**

*Don’t bother looking over here. We’re already infected.*

Another approach some antivirus software uses is called immunization. Products with this type of functionality would make it look as though a file, program, or disk was already infected. An **immunizer** attaches code to the file or application, which would fool a virus into “thinking” it was already infected. This would cause the virus to not infect this file (or application) and move onto the next file.

Immunizers are usually virus-specific, since a specific virus is going to make a distinct call to a file to uncover if it has been infected. But as the number of viruses (and other malware types) increased and the number of files needed to be protected increased, this approach was quickly overwhelmed. Antivirus vendors do not implement this type of functionality anymore because of this reason.
Now, even though all of these approaches are sophisticated and effective, they are not 100 percent effective because virus writers are crafty. It is a continual cat and mouse game that is carried out each and every day. The antivirus industry comes out with a new way of detecting malware and the very next week the virus writers have a way to get around this approach. This means that antivirus vendors have to continually increase the intelligence of their products and you have to buy a new version each and every year.

The next phase in the antivirus software evolution is referred to as behavior blockers. Antivirus software that carries out behavior blocking actually allows the suspicious code to execute within the operating system unprotected and watches its interactions with the operating system, looking for suspicious activities. The antivirus software would be watching for the following types of actions:

- Writing to startup files or the Run keys in the Registry
- Opening, deleting, or modifying files
- Scripting e-mail messages to send executable code
- Connecting to network shares or resources
- Modifying an executable’s logic
- Creating or modifying macros and scripts
- Formatting a hard drive or writing to the boot sector

If the antivirus program detects some of these potentially malicious activities, it can terminate the software and provide a message to the user. The newer generation behavior blockers actually analyze sequences of these types of operations before determining the system is infected. (The first-generation behavior blockers only looked for individual actions, which resulted in a large number of false positives.) The newer generation software can intercept a dangerous piece of code and not allow it to interact with other running processes. They can also detect rootkits. In addition, some of these antivirus programs can allow the system to roll back to a state before an infection took place so the damages inflicted can be “erased.”

While it sounds like behavior blockers might bring us our well-deserved bliss and utopia, one drawback is that the malicious code must actually execute in real time; otherwise, our systems can be damaged. This type of constant monitoring also requires a high level of system resources. We just can’t seem to win.

**TIP**

Heuristic detection and behavior blocking is considered proactive and can detect new malware, sometimes called “zero day” attacks. Signature-based detection cannot detect new malware.

Most antivirus vendors use a blend of all of these technologies to provide as much protection as possible. The individual anti-malware attack solutions are shown in Figure 11-30.
Spam Detection

We are all pretty tired of receiving e-mails that try to sell us things we don’t need. A great job, a master’s degree that requires no studying, and a great sex life are all just a click away (and only $19.99!)—as promised by this continual stream of messages. These e-
mails have been given the label spam, which is electronic unsolicited junk e-mail. Along with being a nuisance, spam eats up a lot of network bandwidth and can be the source of spreading malware. Many organizations have spam filters on their mail servers and users can configure spam rules within their e-mail clients, but just as virus writers always come up with ways to circumvent antivirus software, spammers come up with clever ways of getting around spam filters.

Detecting spam properly has become a science in itself. One technique used is called Bayesian filtering. Many moons ago, a gentleman named Thomas Bayes (a mathematician) developed a way to actually guess the probability of something being true by using math. Now what is fascinating about this is that in mathematics things are either true or they are not. This is the same in software. Software deals with 1s and 0s, on and off, true and false. Software does not deal with the grays (probabilities) of life too well.

Bayesian logic reviews prior events to predict future events, which is basically quantifying uncertainty. Conceptually, this is not too hard to understand. If you run into a brick wall three times and fall down, you should conclude that your future attempts will result in the same painful outcomes. What is more interesting is when this logic is performed on activities that contain many more variables. For example, how does a spam filter ensure you do not receive e-mails trying to sell you Viagra, but it does allow the e-mails from your friend who is obsessed with Viagra and wants to continue e-mailing you about this drug’s effects and attributes? A Bayesian filter applies statistical modeling to the words that make up an e-mail message. This means the words that make up the message have mathematical formulas performed on them to be able to fully understand their relationship to one another. The Bayesian filter carries out a frequency analysis on each word and then evaluates the message as a whole to determine if it is spam or not.

So this filter is not just looking for “Viagra,” “manhood,” “sex,” and other words that cannot be printed in a wholesome book like this one. It is looking at how often these words are used, and in what order, to make a determination as to whether this message is spam or not. Unfortunately, spammers know how these filters work and manipulate the words in the subject line and message to try and fool the spam filter. This is why you can receive messages with misspelled words or words that use symbols instead of characters. The spammers are very dedicated to getting messages promising utopia to your e-mail box because there is big money to be made that way.

**Anti-Malware Programs**

Detecting and protecting an enterprise from the long list of malware requires more than just rolling out antivirus software. Just as with other pieces of a security program, certain administrative, physical, and technical controls must be deployed and maintained.

The organization should either have an antivirus policy or it should be called out in an existing security policy. There should be standards outlining what type of antivirus software and anti-spyware software should be installed and how they should be configured.
Antivirus information and expected user behaviors should be integrated into the security-awareness program, along with who a user should contact if she discovers a virus. A standard should cover the do’s and don’ts when it comes to malware, which is listed next.

- Every workstation, server, and PDA should have antivirus software installed.
- An automated way of updating antivirus signatures should be deployed on each device.
- Users should not be able to disable antivirus software.
- A preplanned virus eradication process should be developed and a contact person designated in case of an infection.
- All external disks (USB drives, and so on) should be scanned automatically.
- Backup files should be scanned.
- Review antivirus policies and procedures annually.
- Antivirus software should provide boot virus protection.
- Antivirus scanning should happen at a gateway and on each device.
- Virus scans should be automated and scheduled. Do not rely on manual scans.
- Critical systems should be physically protected so malicious software cannot be installed locally.

**TIP**  Antivirus files that contain updates (that is, new signatures) are called DAT files. It is just a data file with the file extension of .dat.

Since malware has cost organizations millions and billions of dollars in operation costs and productivity hits, many have implemented antivirus solutions at network entry points. The scanning software can be integrated into a mail server, proxy server, or firewall. (They are sometimes referred to as virus walls.) This software scans incoming traffic looking for malware so it can be detected and stopped before entering the network. These products can scan SMTP, HTTP, FTP, and possibly other protocols types, but what is important to realize is that the product is only looking at one or two protocols and not all of the incoming traffic. This is the reason each server and workstation should also have antivirus software installed.

**Patch Management**

Vendors are often in too big a hurry to get something released—anything, in fact—that will plug up some hole just found in their product, even if it means breaking a thing or two in the process. That “thing” they might break, or disable, or remove, in the name of counteracting a vulnerability, may bring your business to a standstill. Pick your favorite DBMS or messaging application and take a look at the list of patches, and then the patches-for-the-patches, that were released over the last five years. Ask your DBA how often a patch breaks some obscure function of a critical application. The same obscure function that just happens to be the favorite *function-of-choice* for your developers, and, of course, the one function the critical application can’t live without.
You can take steps to reduce the impact of such patches. The best way is to develop, and mature, a **patch management process**, the goal of which is to introduce patches and hotfixes into the production environment in a controlled fashion and to always have a rollback plan. A good patch management process follows a structured six-step methodology, as described next.

**Step 1: Infrastructure**

You need to put the base **infrastructure** in place for your patch management process. This doesn’t just include the physical infrastructure, like switches, routers, cables, and everything else that allows you to distribute the patch, but **everything** that will allow the **process** to move forward. Create a patch management strategy that will fit your organization and then assemble a team that is responsible for, and that will be held accountable for, the patching of the systems in the organization. Who should this team include? The team members should be representatives from not just the system administration folks, but also the development community and/or the people that support the applications running alongside or on top of the software being patched. And it is software we are talking about here, regardless of whether it gets installed to a hard drive or gets burnt into some flash-ROM, as is the case with “firmware.” Software gets **patched**, while hardware typically gets **replaced or upgraded**. Nevertheless, whether it’s software or hardware, some **process** should be followed that responsibly introduces a change into the environment—something which may take some thought.

**Step 2: Research**

Many a time, the wrong patch gets quickly installed to the wrong system. In a rush to stop the script-kiddie bombardment of a vulnerable server, an administrator misreads the last portion of a patch name, or downloads the patch for a slightly different version of software, and installs it...completely breaking the application in the process. Or even worse, an administrator finds a patch for download on a less than reputable web site and doesn’t bother to check the authenticity or integrity of the file before installing it. This is the quickest way to put Trojan horses and malware onto a system.

The only way to prevent these unfortunate mishaps is to research a patch first, and make sure the source of any downloads is authentic, and that the files are free from corruption by performing some kind of integrity check. Many files are released along with a fingerprint or **digital signature** created by a hash-algorithm like MD5 or SHA1. These signatures provide a way for ensuring the integrity of a file by recalculating the signature and comparing.

**Step 3: Assess and Test**

Before installing on the production systems, it is important to test for any unexpected effects a patch may have. Testing is best done in a test environment, and one that mirrors, as close as possible, the production environment. Testing should be conducted according to a **test plan** that acts as a script designed to walk a system through all the known functions or procedures. A test plan should give a system a thorough “workout” that simulates the activities of the system in production. Finally, any patch management process should fit into the organization’s Change Management Process. Patching
is certainly changing the production environment, and change management is the practice of controlling such changes, reducing the unexpected issues that can arise, and, should issues occur, providing for a rollback strategy.

**Step 4: Mitigation (“Rollback”)**

So even though you researched, tested, and implemented according to change management practices, you may still encounter issues with a particular patch. Your only choice to keep the production systems functioning is to roll back any changes and to reset the production environment to a state that existed before the patch. No test plan can account for everything, and preparing for the unknown ahead of time is the best way to mitigate issues when they arise. The goal of a rollback plan is to research and outline all the steps necessary to get back to the operational state prior to the installation of a patch should you need to.

**Step 5: Deployment (“Rollout”)**

When the time comes to actually deploy the patch, the best thing you can do is run right over to the most critical system first, and “push the go button”…right? Wrong! Most organizations take a phased approach when possible and start with a pilot group of less critical systems. After a period, the next group of more sensitive systems is patched, ending with the most critical of systems. Often, deployment strategies, especially when many systems are concerned, involve the use of automated scripts or deployment tools. These tools help reduce the issues associated with human error by executing a scripted series of steps every time on every system. A common best practice is to conduct patch deployments within a predetermined scheduled patching window outside of peak usage times, and with the Patch Management Team present or available for support.

**Step 6: Validation, Reporting, and Logging**

Finally, in order to keep track of what was done to the environment, there must be some form of auditing of what, where, when, and how patches get deployed. Logs should be kept, and documents and standard builds and configurations should be updated to include the new patches. Once the deployment is complete, confirmation that all systems designated for patching did actually get patched should be implemented. This can be done via manual inspection of the systems, or through use of some sort of scanning tool. Tools such as these may use an agent that is installed locally on the systems being evaluated, or employ remote application calls or query-packets to determine the status of a patch on a system. Once all scans are completed and all data are collected, a report should be generated for review and then archived for historical purposes.

**Limitations to Patching**

Just because you have a mature process that includes all of the six steps listed earlier, it doesn’t guarantee success. Patch management can be restricted by failures in the systems or infrastructure involved in distribution of the patches, or from failures in the patching method itself. Often, patching simply takes longer than planned, and extends beyond a patching window, causing some systems to be patched while others remain
vulnerable. What’s worse, if patched systems become incompatible with unpatched systems, some systems may become inoperable. This sort of issue, where patching takes too long, is often the result of excess network load that starves systems for the precious bandwidth required to deliver patch files.

**Best Practices**

A good patch management practice involves determining the right solution for the given organization and environment. This, after all, is the real trick to patch management. Anyone can buy the latest patch management tool, but finding the right tool and using it in the most efficient way is where the challenges lie.

Some vendors release their own change management and change control processes. Many publish documents that outline a change control methodology, while others merely release patches in accordance with a predictable posted schedule, along with specific procedures for deployment of their patches.

No matter how great you feel your patch management process is, as part of your rollback plan you should always back up systems before patching. That’s not to say you should only back up when you plan on patching. Far from it. Backups should be part of the day-to-day operations and administration of the systems in the organization. However, it is best to have backups that are up-to-date and that provide the best restore options should issues with patches, or any other problems, arise.

Finally, it is good practice to keep an up-to-date inventory of all the software, hardware, and configuration info. This is more challenging than it sounds. Maintaining a library full of installation CD-ROMs, hardware devices, manuals, and update files, can take a lot of work, not to mention a lot of space or storage. In a pinch, such a library is invaluable and almost always outweighs the cost and trouble of maintaining it, however.

**Anything Else?**

Being able to determine exactly which patches are, and are not, installed on any given system is incredibly useful. One way to track this is with a good configuration management process that requires the maintenance of baseline standard configuration documents. If these are up-to-date, you shouldn’t need to run a scan every time you must determine how many systems need patching.

To reduce the risk to systems, be they patched or not, you should surround such systems with compensating controls that reduce risk to the system from exploit. Server hardening means that unnecessary services are disabled and access controls are tightened to provide only the least privileges required to users and/or processes. Furthermore, systems can be protected by firewalls and other forms of end-point security that limit the exposure of exploitable vectors into a system. None of these negate the need for patch management processes, but they do increase their effectiveness and buy an organization more time to get patches implemented.

**Attacks**

This section shows how software weaknesses and bugs are used to provide methods of bringing down systems and/or networks. The weaknesses can reside in the applications, the operating system, protocols, and the network stack.
Denial of Service

The network stack is the portion of the operating system that enables devices to communicate over the network. This is where packets are built and sent over the wire and where packets are received and processed. Different operating systems and vendors interpret the Requests for Comments (RFCs) for networking protocols differently, which end up in slightly different network stacks. These differences can contain their own flaws that can be taken advantage of to produce a Denial-of-Service (DoS) attack. These attacks are performed by sending malformed packets to a system that does not recognize the format and thus does not know how to properly process it. This can cause the system to crash or stop processing other packets (denial of service).

DoS attacks cost businesses millions of dollars each year because of system downtime, lost revenue and productivity, diminished reputation, and the man-hours involved in tracking down the problem and fixing it. DoS attacks can interrupt service or completely deny legitimate users access to needed system resources.

DoS attacks can consume a victim’s bandwidth by flooding the network connection either from an attacker with more bandwidth than the victim or from several attackers working together to saturate the victim’s network and bring it to its knees. If more than one attack is involved, each attacker amplifies the effects of the other attackers by combining their bandwidth capabilities and overwhelming the victim’s network segment.

Another type of DoS attack uses up all of the victim’s resources instead of consuming the network’s bandwidth. The resources can be processes, file system quotas, memory allocation, and CPU utilization. The following sections discuss some of the possible DoS attacks available.

Smurf

Oh, a cute little blue attack.

The Internet Control Message Protocol (ICMP) is the mini-messenger of IP and can be used to find out what systems are up and running (or alive) when being used within the ping utility. ICMP reports status reports and error messages. When a user pings another computer, the ping utility sends an ICMP ECHO REQUEST message, and if the system is up and running, it responds with an ECHO REPLY message. It basically says, “Hello, computer 10.10.10.1, are you up and running?” and that computer answers back, “Yep.”

The smurf attack requires three players: the attacker, the victim, and the amplifying network. The attacker spoofs (changes the source IP address in a packet header) to make an ICMP ECHO REQUEST packet seem as though it originated at the victim’s system. This ICMP ECHO REQUEST message is broadcast to the amplifying network, which replies to the message in full force. The victim system and network are overwhelmed. The ECHO functions of ICMP are there to determine if a computer is up and running and accepting requests. However, this attack takes advantage of the protocol’s lack of certain safety measures, which are built in to protect computers from being overwhelmed with ECHO messages.

Countermeasures

- Disable direct broadcast functionality at border routers to make sure a certain network is not used as an amplifying site.
Configure perimeter routers to reject as incoming messages any packets that contain internal source IP addresses. These packets are spoofed.

Allow only the necessary ICMP traffic into and out of an environment.

Employ a network-based IDS to watch for suspicious activity.

Some systems are more sensitive to certain types of DoS, and patches have already been released. The appropriate patches should be applied.

**Fraggle**

*Fraggle* is an attack that is similar to smurf, but instead of using ICMP, it employs the User Datagram Protocol (UDP) as its weapon of choice. The attacker broadcasts a spoofed UDP packet to the amplifying network, which in turn replies to the victim’s system. The larger the amplifying network, the larger the amount of traffic that is pointed at the victim’s system.

Different ICMP and UDP packets should be restricted from entering a network for many reasons. An attacker often uses these protocols to learn the topology of a network, locate routers, and learn about the types of systems within the network. Because we want to limit the amount of information available to attackers, the following restrictions should take place at the network’s perimeter routers.

**Countermeasures**

- Disable direct broadcast functionality at perimeter routers to make sure a certain network is not used as an amplifying site.
- Packets that contain internal source IP addresses should not be accepted by perimeter routers as incoming messages. These packets are spoofed.
- Allow only the necessary UDP packets into and out of the environment.
- Employ a network-based IDS to watch for suspicious activity.
- Some systems are more sensitive to certain types of DoS, and certain patches may have already been released. The appropriate patches should thus be applied.

**SYN Flood**


*Response: This is looking like a SYN attack.*

Because TCP is a connection-oriented protocol, it must set up a virtual connection between two computers. This virtual connection calls for handshaking, and when using the TCP protocol, this requires a three-way process. If computer Blah would like to communicate with computer Yuck, Blah will send a synchronize (SYN) packet to a specific port on Yuck that is in a LISTEN state. If Yuck is up, running, and accepting calls, it will reply to Blah with a SYN/ACK acknowledgment message. After receiving that message, Blah will send an ACK message to Yuck, and the connection will be established.

Systems, and their network stack, are expected to only have to deal with a certain number of these types of connections, so they have allocated only a certain amount of resources necessary for these types of functions. A quick analogy is in order. If Katie is only
expecting three to five friends to show up at her house for a get-together on Friday night, she will most likely only buy a couple of six packs of beer and munchies. When Friday night comes around and over 100 people show up, the party comes to a standstill when there is no more beer and only a bag of pretzels to go around. The same sort of thing is true within the network stack. Once too many SYN requests are received, the system runs out of resources to process any more requests to set up communications paths.

Attackers can take advantage of this design flaw by continually sending the victim SYN messages with spoofed packets. The victim will commit the necessary resources to set up this communications socket, and it will send its SYN/ACK message, waiting for the ACK message in return. However, the victim will never receive the ACK message, because the packet is spoofed, and the victim system sent the SYN/ACK message to a computer that does not exist. So the victim system receives a SYN message, and it dutifully commits the necessary resources to set up a connection with another computer. This connection is queued waiting for the ACK message, and the attacker sends another SYN message. The victim system does what it is supposed to and commits more resources, sends the SYN/ACK message, and queues this connection. This may only need to happen a dozen times before the victim system no longer has the necessary resources to open up another connection. This makes the victim computer unreachable from legitimate computers, denying other systems service from the victim computer.

The SYN message does not take a lot of bandwidth, and this type of attack can leave the victim computer in this state from about a minute and a half to up to 23 minutes, depending on the TCP/IP stack. Because the SYN packet is spoofed, tracking down the evildoer is more difficult. Vendors have released patches that increase the connection queue and/or decrease the connection establishment timeout period, which enables the system to flush its connection queue.

**Countermeasures**

- Decrease the connection-established timeout period. (This only lessens the effects of a SYN attack.)
- Increase the size of the connection queue in the IP stack.
- Install vendor-specific patches, where available, to deal with SYN attacks.
- Employ a network-based IDS to watch for this type of activity and alert the responsible parties when this type of attack is under way.
- Install a firewall to watch for these types of attacks and alert the administrator or cut off the connection.

**Teardrop**

When packets travel through different networks, they may need to be fragmented and recombined depending on the network technology of each specific network. Each network technology has a maximum transmission unit (MTU), which indicates the largest
packet size it can process. Some systems make sure that packets are not too large, but do not check to see if a packet is too small. The receiving system, the victim, would receive the fragments and attempt to recombine them, but these fragments have been made in such a way by an attacker that they cannot be properly reassembled. Many systems do not know how to deal with this situation. Attackers can take advantage of this design flaw and send very small packets that would cause a system to freeze or reboot.

**Countermeasures**

- Install the necessary patch or upgrade the operating system.
- Disallow malformed fragments of packets to enter the environment.
- Use a router that combines all fragments into a full packet prior to routing it to the destination system.

**Distributed Denial of Service**

A Distributed Denial-of-Service (DDoS) attack is a logical extension of the DoS attack that gets more computers involved in the act. DoS attacks overwhelm computers by one computer sending bad packets or continually requesting services until the system's resources are all tied up and cannot honor any further requests. The DDoS attack uses hundreds or thousands of computers to request services from a server or server farm until the system or web site is no longer functional.

The attack can use other computers that knowingly participate, but most likely are unknowingly used as slaves in the process. The attacker creates master controllers that can in turn control slaves, or zombie machines. The master controllers are systems an attacker has been able to achieve administrative rights to, so that programs can be loaded that will wait and listen for further instructions. The components of the third tier of computers are referred to as zombies because they do not necessarily know they are involved in an attack. Scripts that have been put on their hard drives execute, and together all the zombies work in concert to overwhelm a victim. An example of a DDoS attack is shown in Figure 11-31.

**Countermeasures**

- Use perimeter routers to restrict unnecessary ICMP and UDP traffic.
- Employ a network-based IDS to watch for this type of suspicious activity.
- Disable unused subsystems and services on computers.
- Rename the administrator account and implement strict password management so systems cannot be used unknowingly.
- Configure perimeter routers to reject as incoming messages any packets that contain internal source IP addresses. These packets are spoofed.
Summary

Although functionality is the first concern when developing software, adding security into the mix before the project starts and then integrating it into every step of the development process would be highly beneficial. Although many companies do not view this as the most beneficial approach to software development, they are becoming convinced of it over time as more security patches and fixes must be developed and released, and as their customers continually demand more secure products.

Software development is a complex task, especially as technology changes at the speed of light, environments evolve, and more expectations are placed upon vendors who wish to be the “king of the mountain” within the software market. This complexity also makes implementing effective security more challenging. For years, programmers and developers did not need to consider security issues within their code, but this trend is changing. Education, experience, awareness, enforcement, and the demands of the consumers are all necessary pieces to bring more secure practices and technologies to the program code we all use.

Quick Tips

- Buffer overflows happen when an application does not check the length of data input.
• If an application fails for any reason, it should go directly to a secure state.

• A database management system (DBMS) is the software that controls the access restrictions, data integrity, redundancy, and the different types of manipulation available for a database.

• In relational database terminology, a database row is called a tuple.

• A database primary key is how a specific row is located from other parts of the database.

• A view is an access control mechanism used in databases to ensure that only authorized subjects can access sensitive information.

• A relational database uses two-dimensional tables with rows (tuples) and columns (attributes).

• A hierarchical database uses a tree-like structure to define relationships between data elements, using a parent/child relationship.

• Most databases have a data definition language (DDL), a data manipulation language (DML), a query language (QL), and a report generator.

• A data dictionary is a central repository that describes the data elements within a database and their relationships. A data dictionary contains data about a database, which is called metadata.

• Database integrity is provided by concurrency mechanisms. One concurrency control is locking, which prevents users from accessing and modifying data being used by someone else.

• Entity integrity makes sure that a row, or tuple, is uniquely identified by a primary key, and referential integrity ensures that every foreign key refers to an existing primary key.

• A rollback cancels changes and returns the database to its previous state. This takes place if there is a problem during a transaction.

• A commit statement terminates a transaction and saves all changes to the database.

• A checkpoint is used if there is a system failure or problem during a transaction. The user is then returned to the state of the last checkpoint.

• Aggregation can happen if a user does not have access to a group of elements, but has access to some of the individual elements within the group. Aggregation happens if the user combines the information of these individual elements and figures out the information of the group of data elements, which is at a higher sensitivity level.

• Inference is the capability to derive information that is not explicitly available.

• Common attempts to prevent inference attacks are partitioning the database, cell suppression, and adding noise to the database.
Polyinstantiation is the process of allowing a table to have multiple rows with the same primary key. The different instances can be distinguished by their security levels or classifications.

Polymorphism is when different objects are given the same input and react differently.

The two largest security problems associated with database security are inference and aggregation.

Data warehousing combines data from multiple databases and data sources.

Data mining is the process of massaging data held within a data warehouse to provide more useful information to users.

Data-mining tools produce metadata, which can contain previously unseen relationships and patterns.

Security should be addressed in each phase of system development. It should not be addressed only at the end of development, because of the added cost, time, and effort and the lack of functionality.

Systems and applications can use different development models that utilize different life cycles, but all models contain project initiation, functional design analysis and planning, system design specifications, software development, installation, operations and maintenance, and disposal in some form or fashion.

Risk management and assessments should start at the beginning of a project and continue throughout the lifetime of the product.

If proper design for a product is not put into place in the beginning, more effort will have to take place in the implementation, testing, and maintenance phases.

Separation of duties should be practiced in roles, environments, and functionality pertaining to the development of a product.

A programmer should not have direct access to code in production. This is an example of separation of duties.

Certification deals with testing and assessing the security mechanism in a system, while accreditation pertains to management formally accepting the system and its associated risk.

Change control needs to be put in place at the beginning of a project and must be enforced through each phase.

Changes must be authorized, tested, and recorded. The changes must not affect the security level of the system or its capability to enforce the security policy.

High-level programming languages are translated into machine languages for the system and its processor to understand.

Source code is translated into machine code, or object code, by compilers, assemblers, and interpreters.
Object-oriented programming provides modularity, reusability, and more granular control within the programs themselves.

Objects are members, or instances, of classes. The classes dictate the objects’ data types, structure, and acceptable actions.

Objects communicate with each other through messages.

A method is functionality that an object can carry out.

Data and operations internal to objects are hidden from other objects, which is referred to as data hiding. Each object encapsulates its data and processes.

Objects can communicate properly because they use standard interfaces.

Object-oriented design represents a real-world problem and modularizes the problem into cooperating objects that work together to solve the problem.

If an object does not require much interaction with other modules, it has low coupling.

The best programming design enables objects to be as independent and modular as possible; therefore, the higher the cohesion and the lower the coupling, the better.

An object request broker (ORB) manages communications between objects and enables them to interact in a heterogeneous and distributed environment.

Common Object Request Broker Architecture (CORBA) provides a standardized way for objects within different applications, platforms, and environments to communicate. It accomplishes this by providing standards for interfaces between objects.

Component Object Model (COM) provides an architecture for components to interact on a local system. Distributed COM (DCOM) uses the same interfaces as COM, but enables components to interact over a distributed, or networked, environment.

Open Database Connectivity (ODBC) enables several different applications to communicate with several different types of databases by calling the required driver and passing data through that driver.

Object linking and embedding (OLE) enables a program to call another program (linking) and permits a piece of data to be inserted inside another program or document (embedding).

Dynamic Data Exchange (DDE) enables applications to work in a client/server model by providing the interprocess communication (IPC) mechanism.

Distributed Computing Environment (DCE) provides much of the same functionality as DCOM, which enables different objects to communicate in a networked environment.

DCE uses universal unique identifiers (UUIDs) to keep track of different subjects, objects, and resources.
An expert system uses a knowledge base full of facts, rules of thumb, and expert advice. It also has an inference machine that matches facts against patterns and determines which rules are to be applied.

Expert systems are used to mimic human reasoning and replace human experts.

Expert systems use inference engine processing, automatic logical processing, and general methods of searching for problem solutions.

Artificial neural networks (ANNs) attempt to mimic a brain by using units that react like neurons.

ANNs can learn from experiences and can match patterns that regular programs and systems cannot.

Java security employs a sandbox so the applet is restricted from accessing the user’s hard drive or system resources. Programmers have figured out how to write applets that escape the sandbox.

ActiveX uses a security scheme that includes digital signatures. The browser security settings determine how ActiveX controls are dealt with.

A virus is an application that requires a host application for replication.

Macro viruses are common because the languages used to develop macros are easy to use and they infect Office products, which are everywhere.

A boot sector virus overwrites data in the boot sector and can contain the rest of the virus in a sector it marks as “bad.”

A stealth virus hides its tracks and its actions.

A polymorphic virus tries to escape detection by making copies of itself and modifying the code and attributes of those copies.

Multipart viruses can have one part of the virus in the boot sector and another part of the virus on the hard drive.

A self-garbling virus tries to escape detection by changing, or garbling, its own code.

A worm does not require a host application to replicate. A logic bomb executes a program when a predefined event takes place, or a date and time are met.

A Trojan horse is a program that performs useful functionality and malicious functionality without the user knowing it.

Smurf and Fraggle are two examples of DoS attacks that take advantage of protocol flaws and use amplifying networks.

Questions
Please remember that these questions are formatted and asked in a certain way for a reason. Keep in mind that the CISSP exam is asking questions at a conceptual level. Questions may not always have the perfect answer, and the candidate is advised against
always looking for the perfect answer. Instead, the candidate should look for the best answer in the list.

1. What is the final stage in the change control management process?
   A. Configure the hardware properly.
   B. Update documentation and manuals.
   C. Inform users of the change.
   D. Report the change to management.

2. Which best describes a logic bomb?
   A. It's used to move assets from one computer to another.
   B. It's an action triggered by a specified condition.
   C. It's self-replicating.
   D. It performs both a useful action and a malicious action.

3. An application is downloaded from the Internet to perform disk cleanup and to delete unnecessary temporary files. The application is also recording network login data and sending it to another party. This application is best described as which of the following?
   A. A virus
   B. A Trojan horse
   C. A worm
   D. A logic bomb

4. Why are macro viruses so prevalent?
   A. They replicate quickly.
   B. They infect every platform in production.
   C. The languages used to write macros are very easy to use.
   D. They are activated by events that happen commonly on each system.

5. Which action is not part of configuration management?
   A. Submitting a formal request
   B. Operating system configuration and settings
   C. Hardware configuration
   D. Application settings and configuration

6. Expert systems are used to automate security log review for what purpose?
   A. To develop intrusion prevention
   B. To ensure best access methods
   C. To detect intrusion
   D. To provide statistics that will not be used for baselines
7. Which form of malware is designed to reproduce itself by utilizing system resources?
   A. A worm
   B. A virus
   C. A Trojan horse
   D. A multipart virus

8. Expert systems use each of the following items except for _______________.
   A. Automatic logical processing
   B. General methods of searching for problem solutions
   C. An inference engine
   D. Cycle-based reasoning

9. Which of the following replicates itself by attaching to other programs?
   A. A worm
   B. A virus
   C. A Trojan horse
   D. Malware

10. What is the importance of inference in an expert system?
    A. The knowledge base contains facts, but must also be able to combine facts to derive new information and solutions.
    B. The inference machine is important to fight against multipart viruses.
    C. The knowledge base must work in units to mimic neurons in the brain.
    D. The access must be controlled to prevent unauthorized access.

11. A system has been patched many times and has recently become infected with a dangerous virus. If antivirus software indicates that disinfecting a file may damage it, what is the correct action?
    A. Disinfect the file and contact the vendor.
    B. Back up the data and disinfect the file.
    C. Replace the file with the file saved the day before.
    D. Restore an uninfected version of the patched file from backup media.

12. Which of the following centrally controls the database and manages different aspects of the data?
    A. Data storage
    B. The database
    C. A data dictionary
    D. Access control
13. What is the purpose of polyinstantiation?
   A. To restrict lower-level subjects from accessing low-level information
   B. To make a copy of an object and modify the attributes of the second copy
   C. To create different objects that will react in different ways to the same input
   D. To create different objects that will take on inheritance attributes from their class

14. When a database detects an error, what enables it to start processing at a designated place?
   A. A checkpoint
   B. A data dictionary
   C. Metadata
   D. A data-mining tool

15. Database views provide what type of security control?
   A. Detective
   B. Corrective
   C. Preventive
   D. Administrative

16. If one department can view employees’ work history and another group cannot view their work history, what is this an example of?
   A. Context-dependent access control
   B. Content-dependent access control
   C. Separation of duties
   D. Mandatory access control

17. Which of the following is used to deter database inference attacks?
   A. Partitioning, cell suppression, and noise and perturbation
   B. Controlling access to the data dictionary
   C. Partitioning, cell suppression, and small query sets
   D. Partitioning, noise and perturbation, and small query sets

18. What is a disadvantage of using context-dependent access control on databases?
   A. It can access other memory addresses.
   B. It can cause concurrency problems.
   C. It increases processing and resource overhead.
   D. It can cause deadlock situations.
19. If security was not part of the development of a database, how is it usually handled?
   A. Through cell suppression
   B. By a trusted back end
   C. By a trusted front end
   D. By views

20. What is an advantage of content-dependent access control in databases?
   A. Processing overhead.
   B. It ensures concurrency.
   C. It disallows data locking.
   D. Granular control.

21. Which of the following is used in the Distributed Computing Environment technology?
   A. A globally unique identifier (GUID)
   B. A universal unique identifier (UUID)
   C. A universal global identifier (UGID)
   D. A global universal identifier (GUID)

22. When should security first be addressed in a project?
   A. During requirements development
   B. During integration testing
   C. During design specifications
   D. During implementation

23. Online application systems that detect an invalid transaction should do which of the following?
   A. Roll back and rewrite over original data.
   B. Terminate all transactions until properly addressed.
   C. Write a report to be reviewed.
   D. Checkpoint each data entry.

24. What is the final phase of the system development life cycle?
   A. Certification
   B. Unit testing
   C. Development
   D. Accreditation

25. Which of the following are rows and columns within relational databases?
   A. Rows and tuples
   B. Attributes and rows
C. Keys and views
D. Tuples and attributes

**Answers**

1. **D.** A common CISSP theme is to report to management, get management’s buy in, get management’s approval, and so on. The change must first be approved by the project or program manager. Once the change is completed, it is reported to senior management, usually as a status report in a meeting or a report that addresses several things at one time, not necessarily just this one item.

2. **B.** A logic bomb is a program that has been coded to carry out some type of activity when a certain event takes place, or when a time and date are met. For example, an attacker may have a computer attack another computer on Michelangelo’s birthday, the logic bomb may be set to execute in two weeks and three minutes, or it may initiate after a user strikes specific keys in a certain sequence.

3. **B.** A Trojan horse looks like an innocent and helpful program, but in the background it is carrying out some type of malicious activity unknown to the user. The Trojan horse could be corrupting files, sending the user’s password to an attacker, or attacking another computer.

4. **C.** A macro language is written specifically to allow nonprogrammers to program macros. Macros are sequences of steps that can be executed with one keystroke, and were developed to reduce the repetitive activities of users. The language is very simplistic, which is why macro viruses are so easy to write.

5. **A.** Submitting a formal request would fall under the change control umbrella. Most environments have a change control process that dictates how all changes will be handled, approved, and tested. Once the change is approved, there needs to be something in place to make sure the actual configurations implemented to carry out this change take place properly. This is the job of configuration management.

6. **C.** An IDS can be based on an expert system or have an expert system component. The job of the expert system is to identify patterns that would represent an intrusion or an attack that an IDS without this component may not pick up on. The expert system will look at a history of events and identify a pattern that would be otherwise very hard to uncover.

7. **A.** A worm does not need a host to replicate itself, but it does need an environment, which would be an operating system and its resources. A virus requires a host, which is usually a specific application.

8. **D.** An expert system attempts to reason like a person by using logic that works with the gray areas in life. It does this by using a knowledge base, automatic logical processing components, general methods of searching for solutions, and an inference engine. It carries out its logical processing with rule-based programming.
9. B. As stated in an earlier answer, a virus requires a host to replicate, which is usually a specific application.

10. A. The whole purpose of an expert system is to look at the data it has to work with and what the user presents to it and to come up with new or different solutions. It basically performs data-mining activities, identifies patterns and relationships the user can’t see, and provides solutions. This is the same reason you would go to a human expert. You would give her your information, and she would combine it with the information she knows and give you a solution or advice, which is not necessarily the same data you gave her.

11. D. Some files cannot be properly sanitized by the antivirus software without destroying them or affecting their functionality. So, the administrator must replace such a file with a known uninfected file. Plus, the administrator needs to make sure he has the patched version of the file, or else he could be introducing other problems. Answer C is not the best answer because the administrator may not know the file was clean yesterday, so just restoring yesterday's file may put him right back in the same boat.

12. C. A data dictionary holds the schema information about the database. This schema information is represented as metadata. When the database administrator modifies the database attributes, she is modifying the data dictionary because it is the central component that holds this type of information. When a user attempts to access the database, the data dictionary will be consulted to see if this activity is deemed appropriate.

13. B. Instantiation is what happens when an object is created from a class. Polymorphism is when more than one object is made, and the other copy is modified to have different attributes. This can be done for several reasons. The example given in the chapter was a way to use polymorphism for security purposes, to ensure that a lower-level subject could not access an object at a higher level.

14. A. Savepoints and checkpoints are similar in nature. A savepoint is used to periodically save the state of the application and the user’s information, while a checkpoint saves data held in memory to a temporary file. Both are used so that if the application endures a glitch, it has the necessary tools to bring the user back to his working environment without losing any data. You experience this with a word processor when it asks you if you want to review the recovered version of a file you were working on.

15. C. A database view is put into place to prevent certain users from viewing specific data. This is a preventive measure, because the administrator is preventing the users from seeing data not meant for them. This is one control to prevent inference attacks.

16. B. Content-dependent access control carries out its restrictions based upon the sensitivity of the data. Context-dependent control reviews the previous access requests and makes an access decision based on the previous activities.
17. A. Partitioning means to logically split the database into parts. Views then dictate what users can view specific parts. Cell suppression means that specific cells are not viewable by certain users. And noise and perturbation is when bogus information is inserted into the database to try to give potential attackers incorrect information.

18. C. Relative to other types of access control, context-dependent control requires a lot of overhead and processing, because it makes decisions based on many different variables.

19. C. A trusted front end can be developed to implement more security that the database itself is lacking. It can require a more granular and stringent access control policy by requiring tighter identification and authorization pieces than those inherent in the database. Front ends can also be developed to provide more user friendliness and interoperability with other applications.

20. D. As stated in an earlier answer, content-dependent access control bases its access decision on the sensitivity of the data. This provides more granular control, which almost always means more processing is required.

21. B. A universal unique identifier (UUID) is used by DCE, and a globally unique identifier (GUID) is used by DCOM. DCE and DCOM both need a naming structure to keep track of their individual components, which is what these different naming schemes provide.

22. A. The trick to this question, and any one like it, is that security should be implemented at the first possible phase of a project. Requirements are gathered and developed at the beginning of a project, which is project initiation. The other answers are steps that follow this phase, and security should be integrated right off the bat instead of in the middle or at the end.

23. C. This can seem like a tricky question. It is asking you if the system detected an invalid transaction, which is most likely a user error. This error should be logged so it can be reviewed. After the review, the supervisor, or whoever makes this type of decision, will decide whether or not it was a mistake and investigate it as needed. If the system had a glitch, power fluctuation, hangup, or any other software- or hardware-related error, it would not be an invalid transaction, and in that case the system would carry out a rollback function.

24. D. Out of this list, the last phase is accreditation, which is where management formally approves of the product. The question could have had different answers. For example, if it had listed disposal, that would be the right answer because it would be the last phase listed.

25. D. In a relational database, a row is referred to as a tuple, while a column is referred to as an attribute.