11 Modern Distributed Platform, Web Services and .NET Security

11.1 Introduction

Modern distributed computing platforms that support the current and future applications in the digital age are quite complex. The services provided by these platforms consist of operating system services, system software, middleware services, and application servers. These platforms are a double-edge sword from a security point of view – they

provide security (security software is usually imbedded in one or more of these services) but also themselves need to be protected (protecting the protector!).

Figure 11-1 will serve as a general framework for discussion. This framework, introduced in the first chapter of this book, illustrates the role of the following main IT infrastructure building blocks:

- Networks that provide the network transport between remote parties and are responsible for routing and flow/error control support. The networks may be the private value added networks (VANs), Public Internet, and/or Extranets that utilize the wired or wireless transmission media. We have discussed network services security in the previous chapters.
- Computing platforms that consist of operating systems and computing hardware to provide the basic scheduling and hardware services. The computing platforms also include local system software services such as database managers, transaction managers, language translators, and utilities. We will look at security of these services in Section 11.2.
- Middleware that interconnects remotely located users, databases and applications. Middleware components are *business/industry unaware* software modules that provide a variety of services such as Web services, directory services, email, and remote data access services. We will review the general middleware security issues in Section 11.3.
- Middleware platforms, also known as application servers, represent an interesting trend at present. These platforms package a variety of IT building blocks into "application support environments" that can support the current and future breed of distributed applications. Web Services is a major example of such a movement. A dominant platform that complies to Web Services is the Microsoft's .NET .NET environment. Security issues related to Web Services and .NET are discussed in Sections 11.4 and 11.5, respectively.

The discussion in this chapter may need some extra background on the part of the reader although the general overview presented in Chapter 7 should suffice for most discussions. The goal here is not to discuss the individual technology layers in detail, but instead to concentrate on the security aspects of these technologies. If needed, the following two paperbacks provide additional background:

- Umar, A., “e-Business and Distributed Systems Handbook: Middleware Module”, NGE Solutions, May 2003
- Umar, A., “e-Business and Distributed Systems Handbook: Platforms Module”, NGE Solutions, May 2003

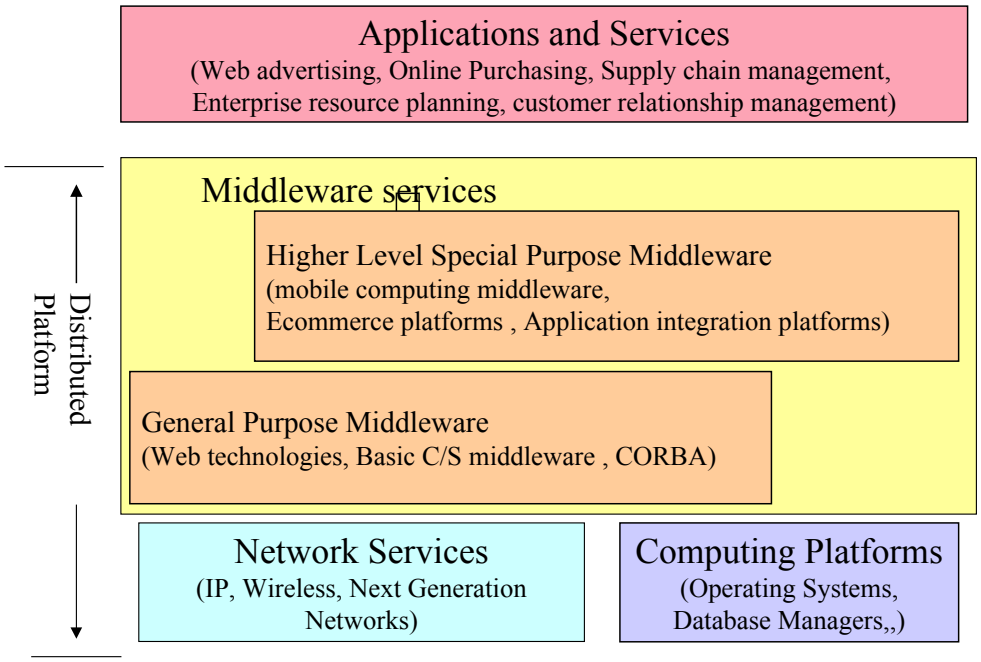

Figure 11-1: Modern IT Infrastructure

Chapter Highlights

- The IT infrastructure (i.e., middleware, networks, operating systems, hardware) supports the modern digital enterprises and needs to be protected against assaults.
- Operating system security has been discussed widely over the years. The main issues to be considered in this area are hardening of operating systems and servers before corporate use.
- Database and transaction security is typically being handled by the DBMS and transaction management vendors.
- Many middleware middleware packages (e.g., CORBA) and middleware platforms (e.g., ecommerce platforms, mobile application platforms) provide a wide range of security services.
- Web Services (WS) security is a relatively new area of work with many open issues.
- SAML (security Assertion Markup Language) is a major component of WS security.
- Microsoft .NET provides many security features, many of them overlap with WS security.
- Microsoft .NET Passport is an interesting initiative for single signon in WS environments.

11.2 Operating System and Local Software Security – A Quick Overview

11.2.1 Operating System (OS) Security

An operating system (OS) is a program, or a collection of programs, which allocates computer resources (memory, CPU, I/O devices, files, etc.) to processes (user commands, jobs, database managers, other operating systems). Operating system security is a vital issue because if the operating system is not secure, then everything that runs under the operating system is also not secure. To understand the role of operating system security, let us look at Figure 11-2 that shows a functional model of operating systems and how they support network services, database and transaction managers, middleware services and the applications and business service.

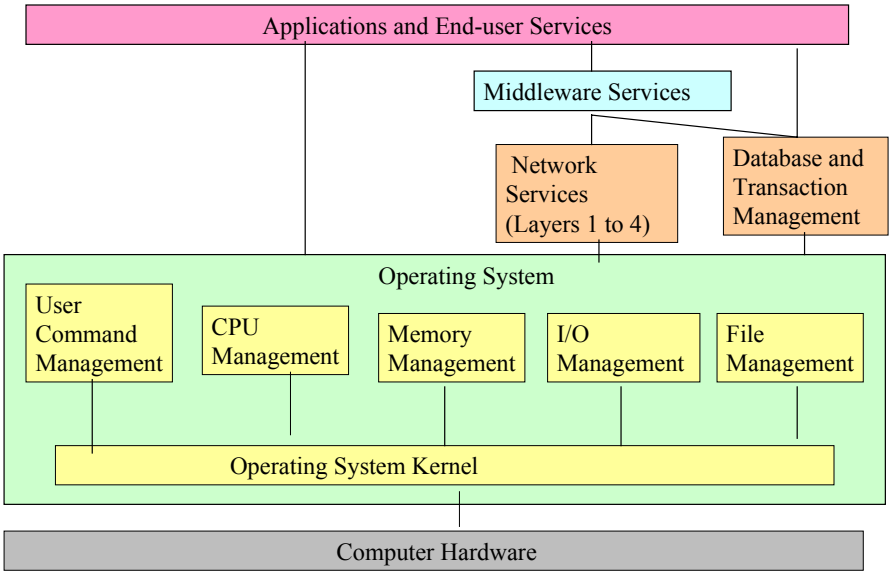

Figure 11-2: Centralized Operating System - Functional View

At the core of the operating system is the OS kernel that coordinates the functional components to run processes. The user command manager parses and executes the user commands. The memory manager allocates the main memory and the hardware registers to various processes. Most memory managers include the capabilities to manage virtual memory and translate between virtual and real memory. The CPU (central processing unit) manager allocates the CPU to the processes. A variety of CPU scheduling schemes (time slicing, interrupt driven) have been employed in the operating systems. The file managers manage the data resources in the system. This normally includes catalogs, file sharing, etc. Database managers and transaction managers rely on the file management capabilities of OSs. The I/O (input/output) managers steward all the local and remote I/O activities. The network services rely on the I/O service. Lower layers of network services (layers 1 and 2) may be included in some operating systems as I/O facilities.

Many operating systems have been developed over the last thirty years. Examples of state of the market operating systems are Windows and its variants (NT, 2000, 2003, XP), Unix, Linux, and MVS. Security services in each operating system are implemented differently, however, the following are the key ideas:

- **Authentication:** To identify and verify users (i.e., make sure that you are who you say you are), all operating systems support IDs and password for authentication.
- **Authorization (Access control):** To determine what access rights that person has (i.e., can you only read given information or can you also update, delete, add information). All operating systems use authority control lists (ACLs) that show who is authorized to do what.
- **Privacy and Confidentiality:** To assure privacy of information (i.e., no one other than the authorized people can see the information) under the control of the operating system, almost all operating systems support encryption in addition to user ID/password.
- **Integrity:** To assure the integrity of information (i.e., no unauthorized modification), all operating systems use ACLs to make sure that only authorized users can update the information.
- **Accountability and Assurance:** To determine who did what when and convince yourself that the system keeps its security promises, all operating systems provide extensive logs and audit trails. This includes non-repudiation (NR) -- the ability to provide proof of the origin or delivery of data. NR, as discussed in Chapter 5, is heavily supported through logs.

The exact implementation of security for different operating systems depends on the operating system itself. For example, Unix Security uses extensive user IDs, passwords and ACLs, for individual users and user groups. Windows NT uses a registry as a central database to store configurations for security and IBM mainframe operating system (MVS) uses RACF (Resource Access Control Facility). It is far beyond the scope of this book to discuss individual operating system security. This subject has been covered extensively in the literature as "host security" (see the sidebar "Sources of Information on Operating System Security").

Although different security approaches are used in different operating systems, ***OS hardening*** is a common approach used to secure most operating systems after installation. The OSs when delivered from the vendors are intended for quick installation and maintenance. This, however, leaves many security holes that need to be filled later as part of the hardening process. In particular, if a firewall is going to reside on a machine, then the host operating system must be hardened for obvious reasons. The main steps of OS hardening are:

- Apply all vendor-recommended security patches.
- Remove all unnecessary services because they may be used by intruders to launch attacks.
- Tighten file and directory permissions.
- Control user access to needed resources by using rigorous ACLs.

Although OS hardening can be a long and laborious task, it is extremely important for system security. A host with "soft" operating system, especially when connected to a network, can be home for launching attacks and thus have disastrous results. Fortunately, many GUIs and scripts commonly accompany OSs to ease the pain of OS hardening.

Sources of Information on Operating System Security

- For ongoing updates on MS Windows security, visit http://www.windowsecurity.com/
- Gollman, D., "Computer Security", Wiley, 2000
- Unix security (http://www.sri.ucl.ac.be/SRI/documents/unix-secure)
- ftp://ftp.win.tue.nl/pub/security/index.html -- papers related to Unix security
- www.linuxsecurity.com for Linux security (naturally!)
- Sutton, S.A., "Windows NT Security Guide", Addison Wesley, 1996
- "Remote Access Control Facility for MVS", available from IBM Library

11.2.2 Server Hardening

In addition to OS hardening, several servers may need to be hardened also by using procedures that are similar to OS hardening. These procedures are customized for the type of server being hardened. Examples of the servers that may need to be hardened for security are:

- LAN servers such as Windows NT Server
- Web servers such as Apache and IIS Servers
- Database servers such as Oracle Server
- E-mail servers such as MS Exchange, Lotus Notes, and Web-mail
- DNS (Domain Name Services) servers

It should be noted that these servers are not hardware machines but are software packages that are installed on various machines.

11.2.3 Database and Transaction Security

The local software in distributed environments provides access and manipulation of data and processes located on networked machines. Examples of the local software are database managers, transaction managers, and utilities such as software development tools and system administration environments. For the purpose of security, we review database and transaction management security.

Database management system (DBMS) security should provide the privacy, integrity, authorization, authentication, accounting, and availability (PIA4) support to databases for on-line and batch users. In a typical database environment, different users can view, access and manipulate the data in a database. Typical DBMSs are designed to a) manage logical views of data so that different users can access and manipulate the data without having to know the physical representation of data, b) manage concurrent access to data by multiple users, enforcing logical isolation of transactions, c) enforce security to allow access to authorized users only, and provide integrity controls and backup/recovery of a database.

Relational database managers such as DB2, Oracle, Sybase, and/or Informix are typically used in many contemporary applications. Older systems use hierarchical database managers such as IMS. Object oriented databases are still in their infancy (see the tutorial on " Databases and SQL" in the Tutorials Module). The AAAIP support for DBMSs is

currently provided by the commercially available DBMS suppliers through database administration (DBA) facilities through IDs, passwords, logs, and directories. Information about specific database management product security can be found in the product documentation such as "Oracle Security Guide" from Oracle Corp.

DBMS security is practically the same as database server security because most DBMSs at present are available as database servers. Some general guidelines for DBMS/database server security are:

- Production databases, especially with sensitive data, should be isolated and not be kept on the "test" machines.
- Operating system level access for developers and testers should not be permitted on machines housing production databases.
- The names and locations of the production databases should not be publicly advertised.
- Production databases should be kept on their own subnets behind corporate firewalls.

Transaction management security, also known as transaction processing monitor (TP monitor), monitors the execution of transactions (sequence of statements that must be executed as a unit). TMs (transaction managers) specialize in managing transactions from their point of origin to their termination (planned or unplanned) and must run in a secure environment. Some TM facilities are integrated with the DBMS facilities to allow database queries from different transactions to access/update one or several data items (IBM's IMS DB/DC is an example). However, some TMs specialize in handling transactions only (CICS, Tuxedo and Encina are examples). Transaction management security is in principle quite similar to database security and in many cases closely integrated with database security.

11.2.4 Host Resident Firewalls

We discussed network firewalls in a previous chapter. These firewalls are used to protect the perimeter of the enterprise. In addition, many host firewalls are currently available for protecting the individual computers. These firewalls are highly recommended for home computing where your computer is directly connected to the Internet through a DSL or cable modem. These firewalls intercept all the inbound and outbound traffic on your computer and generate alerts when appropriate. For example, I have a host resident firewall on my desktop at home and it alerts me whenever someone from outside attempts to read my files and suggests action (permit, deny). This is a very inexpensive and effective way of protecting your computer.

11.3 Middleware Services Security

11.3.1 Introduction

In addition to network services, current and future applications require a variety of COTS middleware that is appearing at a dizzying pace with increasing complexity. Examples of

the COTS middleware include implementations of standards such as Web technologies, CORBA, XML parsers/processors, WAP (Wireless Application Protocol), and Java Messaging Services (JMS).

Security and intrusion tolerance capabilities of emerging COTS middleware require urgent attention because many mission critical applications rely on layers of middleware to operate properly. An intruder can completely destroy the integrity of an application by modifying the middleware being used by the application even when the network is highly secure. An intruder can potentially modify the web server software and remote messaging middleware to redirect the notifications to a different receiver (say an unfriendly site). In this case, the application code itself is not modified, only the middleware used by the application is. However, the integrity of the application is completely destroyed. In addition to application integrity, availability of an application can also be destroyed by attacking the middleware. Basically, unavailability of middleware makes an application unavailable even when the networks and computing platforms are running correctly. For example, a CORBA-based application cannot run correctly if CORBA middleware has been compromised even with perfect network and computing hardware/software.

Many emerging COTS middleware products are weak in terms of intrusion tolerance (IT). Intrusion, in this context, is any undesirable/unauthorized activity that exposes, prevents and/or subverts a legitimate operation. Thus intrusion tolerance includes security as well as fault tolerance. We are potentially facing serious threats such as the following that are typically not discussed in intrusion tolerance literature:

- Directories that contain addresses of participating applications in a weapon deployment system are contaminated, or modified, so that the traffic between the participants stops even though the network, the operating system and computing hardware are fully operational.
- XML Document Translation Definitions (DTDs) used in sensitive XML-based message exchanges are corrupted so that all XML messages become invalid thus halting the message exchanges because every transaction becomes invalid. XML-based messaging is being considered by a diverse array of applications that range from E-commerce to flight control systems to traffic lights.
- EAI (Enterprise Application Integrator) data translation and routing rules between telecommunications provisioning systems, inventory management systems, and billing/payment systems are modified by an intruder thus causing a major crisis that ripples across many industries. Many organizations, such as JBI (Joint Battlespace Infosphere), are beginning to use EAI and publish/subscribe platforms. EAI middleware, currently a more than $10B business, is being used heavily to interconnect different applications of different vintages using different technologies residing on different machines.
- Workflows between participants in a military supply chain are corrupted so that the supply chain is interrupted.
- Someone contaminates EJB (Enterprise Java Bean) container disabling industry segments. This is possible because components (EJBs, CORBA components) are being positioned to develop many applications for industry segments (e.g., financial). Components are “dropped in” to containers that provide security, transaction etc.

- Call agents for VOIP (Voice Over IP) and other "next generation network" applications are corrupted to allow unauthorized routing of calls and eavesdropping with obvious undesirable consequences.

This report highlights key areas of intrusion vulnerability in emerging middleware and identifies future areas of work. We first define the role of middleware in intrusion tolerance and analyze the intrusion threats introduced due to the increased reliance of current and future mission critical applications on emerging middleware.

11.3.2 Middleware and Middleware Platforms – A Revisit

As discussed in Chapter 7, middleware is a set of common business-unaware services that enable applications and end users to interact with each other *across a network.* In essence, middleware is the software that handles distributed communication between applications and resides above the network and below the business-aware application software. This definition excludes "stand alone" system software such as math libraries, OS, and file systems. Current and future e-business applications require a variety of COTS middleware that ranges from simple components such as Sun RPC (remote procedure call) to sophisticated packages such as enterprise application integrators that combine several middleware components into a single framework. At present more than 200 COTS middleware packages of varying complexity from a wide range of suppliers are reportedly available. To develop a systematic and closer look at the emerging middleware, let's consider the following middleware stack (see Figure 11-3):

- Basic Middleware components that support a thin layer of interactions between remotely located applications. Examples of these components include directory services, electronic messaging (e.g., Email), Web services (e.g., Web browsers and servers), remote data services (e.g., bulk data transfer, ODBC/JDBC), and remote application services (e.g., interactions between remote processes through RPCs and message oriented middleware (MOMs), and the basic distributed object services (e.g., the CORBA ORB).
- Special purpose middleware to provide value added features needed by several special purpose applications such as E-commerce and video conferencing. Examples of this middleware for E-commerce include EDI, micropayment support, shopping carts, and middleware to support electronic marketplaces.
- Higher level middleware such as distributed transaction processors, B2B workflow and collaborative systems, message brokers, and middleware to support supply chain management and virtual enterprises.

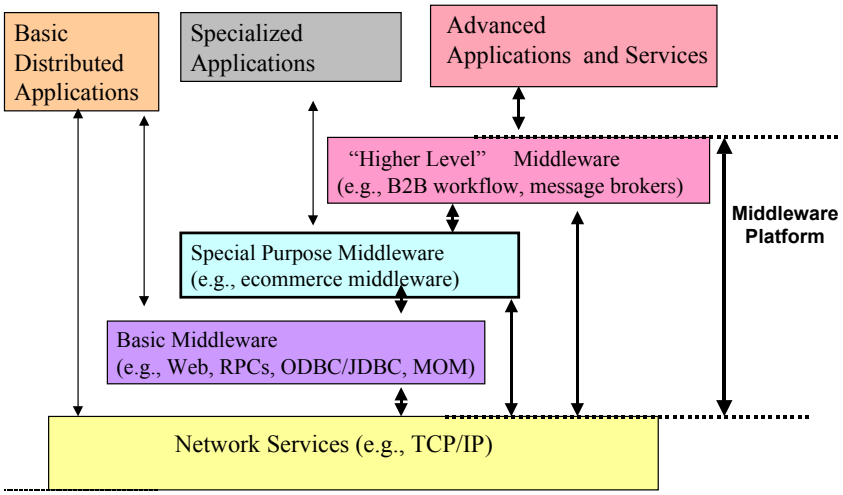

Figure 11-3: Middleware Stack to Support Applications

COTS middleware for a while has followed the "grocery store" model, i.e., pick and choose the middleware components you want and put them together as a service for the applications. For example, to provide e-commerce services, you could choose COTS middleware packages from a variety of suppliers to provide Internet purchasing, cataloging, payment, and order processing services. Since the late 1990s, several middleware components have been packaged together by vendors as ***"middleware platforms"*** for special purposes**.** Examples of middleware platforms are the E-commerce platforms such as WebSphere (www.ibm.com), Broadvision (www.broadvision.com), and OpenMarket (www.openmarket.com) that combine Web, mobile access, cataloging, payment, order processing and other services together for E-commerce. Middleware platforms combine several middleware components of different types (general purpose, special purpose, higher level), in many cases from different suppliers, into a single framework for a specific purpose. Examples of other middleware platforms are:

- Enterprise Application Integration (EAI) platforms such as vitria (www.vitria.com) that combine message transport, message routing, data translation, and workflow into a single framework for application integration.
- XML platforms such as the Tibco XML Suite (www.tibco.com) that combines a variety of XML tools into a set of integrated services.
- CORBA platforms that include a wide range of services (e.g., naming, trading, event, messaging, object life cycle services) in addition to the basic remote object invocation services.
- Wireless platforms such as WAP (Wireless Application Protocol) that combine several networking as well as application development services for mobile applications.
- Emarket platforms such as Ariba and Commerceone that combine a wide range of cataloging, trading, negotiating, and payment services to establish emarkets.
- Supply chain management platforms such as i2 (www.i2.com) that support the setup and monitoring of B2B supply chains.

For an extensive discussion of these and other middleware services, see [Umar 2003-Middleware, Umar 2003-Platform].

Due to the number of middleware packages, it is not possible to analyze the security features of every available middleware package in this chapter. To highlight the main ideas, security features of a few middleware packages are discussed here. The web security issues were discussed in the previous chapter.

11.3.3 Basic Middleware Security and Secure RPCs

Figure 11-4 shows a conceptual view of basic middleware and depicts how a client interacts with a server through the C/S middleware. The C/S middleware is based on a few basic interaction paradigms:

- Remote procedure call (RPC) that accesses a remote program (e.g., a purchasing system at a merchant site).
- Remote data access (RDA), also known as remote SQL, to access remote databases (e.g., an Oracle database at a remote site). ODBC is an example of RDA.
- Message oriented middleware (MOM) that accesses remote systems through queues (also known as queued message processing -QMP).
- Publish/subscribe model that uses channels, very much like TV channels, to push information to subscribers.

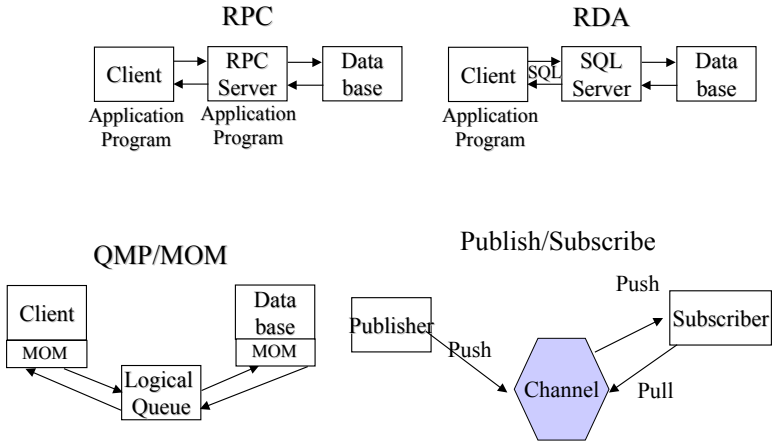

Figure 11-4: Basic Remote Interactions

In all these interactions, the client side of the middleware is typically light-weight (it usually just routes the requests to the servers), but the server side of the middleware is quite sophisticated because it needs to schedule and manage numerous client requests. For example, several clients could request access to the applications, databases, printers and files located at the same server. The C/S middleware relies on network services to transport the messages between sites.

Security of these middleware services depends on the type of service employed and the platform employing it. For example, Network Operating Systems (NOSs) have traditionally provided some of the basic middleware services. An NOS is essentially an operating system that provides the capabilities for transparent access to resources such as printers and files across a network. Historically, NOSs have originated from LANs and have concentrated on directing the user's request to appropriate print and file servers.

Novell Netware is one of the oldest NOSs. Other examples are Windows Servers[1], Banyan Vines, and IBM's LAN Manager. However, with time, NOSs have started providing other services such as RPC and directory services. Security is built into these NOSs.

For standalone security, RPC and its variants are used regularly in accessing remotely located application programs. In ecommerce, for example, customers access purchasing systems at merchant sites by using an RPC type middleware. Examples of RPC-based products are:

- CORBA (Common Object Request Broker Architecture) as well as Microsoft's DCOM uses RPC at its core.
- SUN RPC, Netwise RPC, Novell Netware RPC, and TCP/IP RPC use RPC.

In this paradigm, the client process invokes a remotely located procedure (a server process), the remote procedure executes, and sends the response back to the client. The remote procedure can be simple (e.g., retrieve time of day) or complex (e.g., retrieve all customers from Detroit who have a good credit rating). Each request/response of an RPC is treated as a separate unit of work, thus each request must carry enough information needed by the server process.

A remote procedure call (RPC) facility allows a language level (local) procedure call by the client to be turned into a language level call at the server. In a remote procedure call, a local process invokes a remote process. RPCs have the main advantage that a programmer issues a call to a remote process in a manner that is very similar to the local calls. The RPC software attempts to hide all the network related details from the client-server developers. The client calls are referred to as requests and the server results being returned are referred to as responses. Figure 11-5 shows a conceptual view of a RPC. Note that the RPC procedure can be called by more than one client (i.e., once you have developed an RPC server, it can be invoked by multiple clients residing on multiple machines). In addition, the database access is optional because the RPC server may or may not need to access a database. For example, the RPC server may produce a graph from given inputs or compute averages without accessing a database).

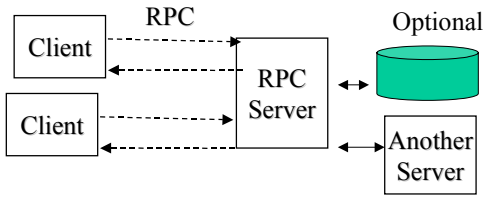

Figure 11-5: Remote Procedure Call (RPC)

Security is one of the main issues in RPCs, i.e., how to authenticate the RPC participants and then to protect the communications between the participants against intruders. ***Secure RPC*** was introduced for exactly this reason. Secure RPC uses a combination of public key cryptography and secret key cryptography. It is based on DES secret key cryptography for encrypting information that is sent over the network -- DES is also used to encrypt the user's secret key that is stored in a central network server. The Diffie-Hellman mechanism is used for key exchange between users. Secure RPC can be

[1] Windows Servers is used as a generic term here to represent the family of Microsft Windows Servers such as Windows NT Server, Windows 2000 Server, Windows XP Server.

combined with higher level protocols, such as NFS (Network File Services), to provide high level of security in client/server environments. Secure RPC assures that the user's password is not transmitted over the network and the secret key is transmitted over the network only when it is encrypted using the user's password.

Despite its many strengths, Secure RPC has not been widely deployed in industrial settings. The reasons range from licensing issues to lack of vendor interest in building and deploying secure RPC. However, the concepts developed in secure RPC have been used in many systems such as CORBA.

11.3.4 CORBA Security

CORBA security is a good example of protection at middleware level. Most CORBA implementations at present use IIOP (Internet Interoperable Protocol) that operates on top of TCP/IP. The OMG (Object Management Group) specifies three levels of security for applications that use CORBA:

- CORBA Security Level 0 provides authentication and session encryption using SSL(Secure Socket Layer).
- CORBA Security Level 1 provides security to applications that are not aware of security. It automatically introduces authentication, session encryption, access control, and simple auditing. CORBA security level 1 uses CORBA interceptors.
- CORBA Security Level 2 provides API to interface to CORBA security objects. Security-aware applications can specify security requirements (e.g. access control) at object and method level.

Most CORBA implementations at present support level 1 security, some support it at a higher level. You need to make sure that the CORBA implementation you purchase provides the level of security you need.

11.3.5 e-Commerce (EC) Platform Security

e-Commerce requires a wide range of IT infrastructure services that include network services, general purpose middleware services such as Web, and EC specific services such as e-payment and EDI. The IT infrastructure services needed for EC are being packaged together as electronic commerce platforms (also known as commerce servers). Many vendors such as IBM, Microsoft, Oracle, Netscape, and Sun are providing such integrated EC platforms that can provide almost all services (both transactional and non-transactional) needed for EC. At a conceptual level, the EC platforms consist of the following building blocks:

- Network and operating system services
- Core middleware (e.g., RPCs, Web)
- EC specific middleware
- Management and support services

We have briefly reviewed the EC platforms in Chapter 7. Let us now briefly review the security issues related with the EC platforms. In general, each EC platform provides its own security. For example, most EC platforms support SSL and Secure Electronic Transaction (SET) payments. In addition, XML security features such as XML encryption and XML signatures are increasingly being supported due to central role of

XML in e-commerce. We will not discuss these issues because they are discussed in other parts of this book.

11.3.6 Mobile Application Server (MAS) Security

Mobile application servers support mobile applications such as mobile commerce by providing middleware services to handle mobile devices over wireless networks. The mobility specific middleware, commonly known as wireless middleware, resides above the network and below the applications to support connectivity of mobile users to web content, databases, and applications. In addition to the wireless middleware services, a MAS typically needs to access back-end systems by using general purpose middleware services and also may use EC/EB middleware services to provide value added features needed by EC/EB applications. These MAS's are commercially available from suppliers such as IBM, Oracle, Nokia, and others (see the sidebar "Examples of Commercially Available Mobile Applications Servers").

Figure 11-6 shows the conceptual components of a Mobile Application Server (MAS). A MAS is part of a multi-tier (mostly three tier) architecture that typically consists of a thin client tier residing on handheld devices, a middle tier consisting of business applications and a set of middleware and network services, and a backend tier consisting of mission-critical databases and applications.

Security is one of the main concerns of MAS's because of the wireless security issues discussed in a previous chapter. Different wireless middleware packages such as WAP and I-mode provide different security features, as discussed in a previous chapter, in terms of authentication, data integrity, and data privacy. We will not repeat these discussions here. It should be sufficient to note that a MAS provides a combination of wireless as well as other security features -- both topics are discussed elsewhere.

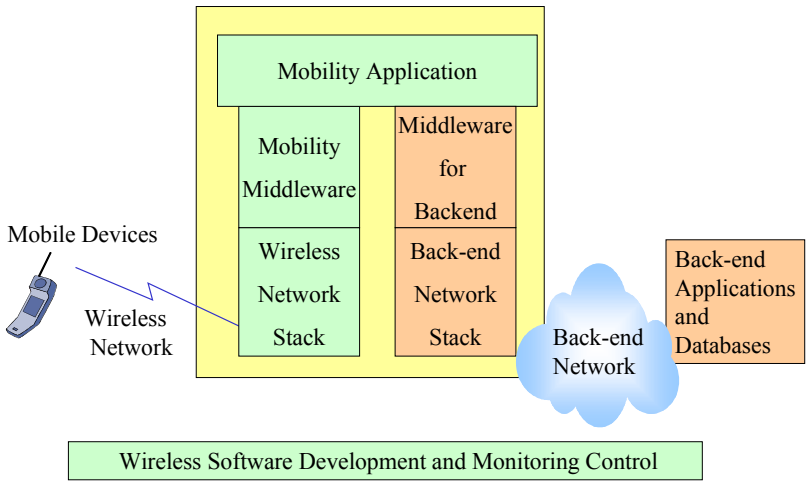

Figure 11-6: Mobile Application Server -- Conceptual View

Examples of Commercially Available Mobile Applications Servers

- Aligo M-1 Mobile Application Server (www.aligo.com) uses the wireless Java, specifically the Sun J2ME (Java 2 Micro Edition) environment to provide the resources and services that developers and architects need to architect, build and deploy Java applications for mobile devices.
- IBM's WebSphere Application Server (www.ibm.com) has the facilities for serious mobile EB application development for large enterprises. However, this is a large and complex system and you must be ready to allocate enough resources to implement solutions based on this MAS.
- Jacada Presentation Server for Palm (www.jacada.com) addresses the needs of developing mission-critical information from enterprise systems to the web through handheld devices.
- Nokia's WAP Application Gateway (www.nokia.com) uses WAP and has an extensive array of facilties for mobile device support with connections to back-end applications.
- Oracle's Mobile Application Server (www.oracle.com) allows access to Oracle databases and application suite from mobile devices.
- Sybase Mobile Application Studio (www.sybase.com) provides a studio of tools for wireless access to Sybase and other databases
- @Hand Mobile Application Server (www.@hand.com) provides core infrastructure, administrative services and a rapid application development environment for creating mobile applications

11.3.7 EAI (Enterprise Application Integration) Security

Enterprise Application Integration (EAI) platforms are used to enable diverse applications to interoperate. EAI platforms consist of a collection of technologies that include message transport mechanisms (e.g. CORBA), adapters, message routers, message/data translators, and process management technologies. EAI platforms also include development and management tools. EAI platforms are used widely to integrate applications. More than a dozen EAI vendors currently exist in the marketplace. Examples of the major EAI vendors are IBM (IBM MQ Integrator), WebMethods (WebMethods-Active Software), Vitria (Vitria Businessware), Tibco (Tibco EAI system), Sun (Sun Fusion) and others. Due to the wide use of EAI platforms, it is important to study intrusion threats. Figure 11-7 shows a conceptual view of EAI platforms (see the "Integration' Module for an extensive discussion of EAI).

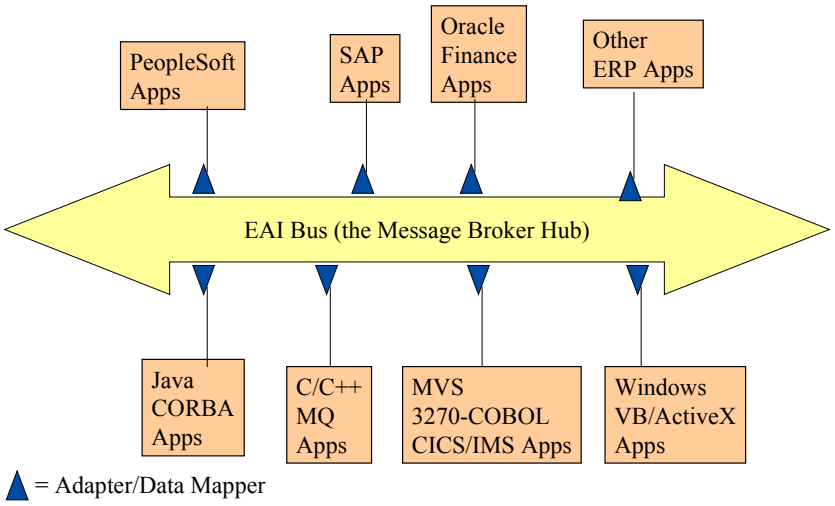

Figure 11-7: Conceptual View of EAI Platforms

EAI components can be attacked at several levels. Examples of the components that can be attacked are EAI publishers and subscribers, EAI Broker, Adapters, Data mappers, Worklflow and Process Management. In particular, intrusion of EAI Broker can have a very high impact on enterprise applications that rely on EAI platforms. Since the Broker serves as the central hub, all publisher and subscriber communication can be completely disrupted and diverted. In addition, unauthorized players can get access to the data.

EAI security is available in different flavors: from the EAI platform vendors, from the individual component providers, or a mixture thereof. For example, Vitria, Tibco, and IBM provide security support for their EAI platforms. The main cautionary suggestion is to thoroughly examine the security features of the EAI platform providers before using them as your enterprise-wide application integration bus.

11.4 Web Services Security and SAML

11.4.1 Overview of Web Services

XML Web Services, commonly known just as "Web Services" (WS), is an area of tremendous activity at present due to the strong support and adoption by the industry (Microsoft, Sun, IBM, Oracle, and numerous others). Naturally, WS security is an area of considerable interest and the interest in this activity is growing with time. For example, IBM and Microsoft have jointly published a white paper on “Security in a Web Services World: A Proposed Architecture and Roadmap” (msdn.microsoft.com/library/en-us/dnwssecur/ html/securitywhitepaper.asp). Most organizations adopting WS are concerned about the security implications and are looking for solutions.

Before analyzing WS security, let us briefly review Web Services. With XML Web Services, also just known as Web Services, any application can be integrated so long as it is Internet-enabled. The core of Web services is XML messages over standard Web

protocols such as HTTP. These industry standards provide a lightweight and widely accepted communication mechanism that any programming language, middleware, or platform can participate in. The main idea of Web Services is that service providers will register themselves in public or private business registries. The registered services will fully describe themselves, including interface structure, business requirements, business processes, and terms and conditions for use. Visit http://www.xmethods.com/ for a listing of some interesting Web services, and links to their accompanying WSDL documents. Consumers of the registered services read these descriptions to understand the abilities of these Web Services and then invoke them. Web Services consist of the following technologies:

- **UDDI** (Universal Description, Discovery and Integration) to provide a directory of services on the Internet. This is conceptually similar to the CORBA/DCOM directory and naming services. More information regarding the UDDI initiative is available at http://www.uddi.org/.
- **WSDL** (XML Web Services Description Language) an XML document that describes the location and interfaces a particular service support. It is conceptually similar to IDL. The Web Services Description Language (WSDL) specification is available at http://www.w3.org/TR/wsdl.
- **SOAP** (Simple Object Access Protocol) through which XML Web Service consumers can send and receive messages using XML. SOAP is conceptually similar to CORBA IIOP. The SOAP specification is available at http://www.w3.org/TR/SOAP/.
- **XML & HTTP** are the core open Internet technologies that are the foundation of XML Web Services, The main source of information for XML and HTTP is www.w3.org.

Web Services are in reality simply XML-based *interfaces* to application and system services and are in fact "Web face lifts" to existing systems. Like many other distributed applications, a Web Service accepts remote service requests, does some processing, and then returns a response. Figure 11-8 shows the steps that take place to develop and use a "loan" service:

1. The provider creates, assembles, and deploys the loan service using the programming language, middleware, and platform of the provider's own choice. It then defines the Web service in WSDL. A WSDL document describes a Web service to others.

2. The provider registers the service in UDDI registries. UDDI enables developers to publish Web services and that enables their software to search for services offered by others.

3. A prospective user finds the service by searching a UDDI registry. The consumer sends a request to UDDI (www.uddi.org) to locate where the loan service is located.

4. UDDI returns the loan service address (www.loan.com/WSDL) that in fact is a link to WSDL.

5. The consumer now issues a call to WSDL to learn how to invoke the loan service (i.e., what method to use, what are the input/output parameters).

6. The WSDL returns a definition of the loan service in WSDL format.

7. The consumer prepares a SOAP message, an XML document, that is sent to www.loan.com over HTTP.

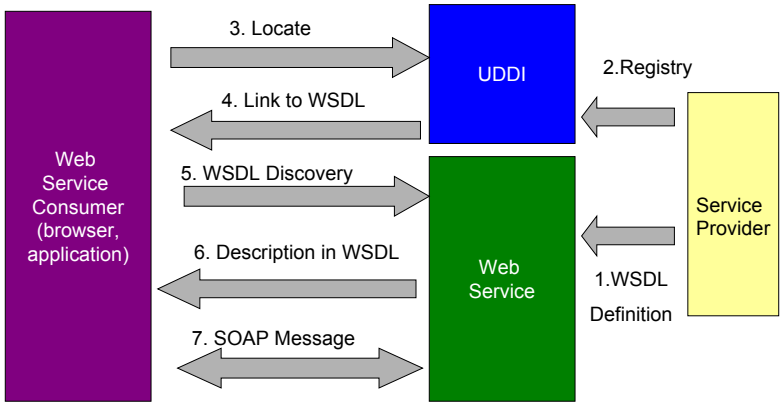

Figure 11-8: XML Web Services in Action

To satisfy this external flow, a number of things happen internally as shown in Figure 11-9. A Web Service contains a Listener that waits for requests. When a request is received, the Listener forwards it on to a component that implements the required business logic. This component might be designed to operate specifically as a Web Service, or it could be some other business component or COM/Java object that the Web Service wants to expose to the outside world. In the latter case, a developer will write some logic that acts as a façade for the Web Service and forwards the request on to the COM/Java object itself. The COM/Java object or other business logic may make use of a database or other data store, accessed using a Data Access layer. As expected, this looks like the classic N-tier architecture in which the Web Service can get its information from a database or another Web Service.

It should be noted that the Web Services core technologies (WSDL, UDDI, SOAP, XML, HTTP) are mainly sufficient for simple Web services. Extended B2B exchanges require an agreed-upon structure for business transactions with multiple steps, backouts, and document flows. These application requirements often stretch the limits of a purely SOAP based implementation. Developments such as **ebXML** (www.ebxml.org) address these issues through a suite of XML specifications and related processes for B2B collaboration and integration. Although there are vendor disagreements on SOAP extensions, ebXML, and service flow descriptions (so what is new!), all major players, including Sun and Microsoft, generally agree that SOAP, WSDL, and UDDI are good things for the future and are working together to establish Web Services standards.

A great deal of information about various aspects of Web Services is available at the Microsoft site (http://msdn.microsoft.com).

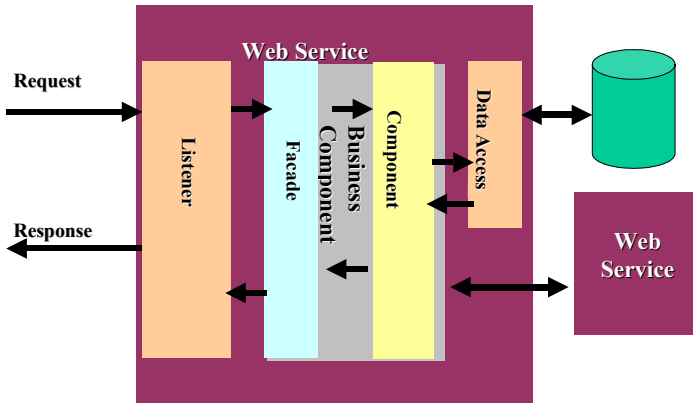

Figure 11-9: Internal Architecture of a Web Service

11.4.2 Issues Unique To Web Services Security

Web Services security needs to be viewed within the context of overall system security. In particular, it is important to interrelate Web Services security issues to the business and technology choices. However, it is also crucial not to re-invent a security solution for every new technology family such as Web Services. For example, security of an online purchasing system must address different security threats than those of a military command control system, even if they both use Web Services. Conversely, a CORBA-based online purchasing system has different security vulnerabilities than that developed by using Web Services or CGI and Javascript. As security issues keep arising in different business contexts and diverse middleware technologies keep appearing at dizzying pace, the separation and interrelationships between the two become increasingly important.

So what are the security issues unique to WS? At a basic level, WS introduce several security exposures due to the following reasons:

- Self-describing nature of XML leaves it open for attacks. Specifically, XML tags can be created to describe a valid document. The validity of the document is specified by an external schema (a DTD or a schema). This creates several security holes.
- If someone modifies XML DTD/Schema, then *all* XML communications can be become invalid. For example, if an intruder changes a DTD so that every name must have a middle name, then all transactions with no middle names would be valid.
- Single signon becomes extremely important because of the distributed nature of WS. SSO is needed so that a single user can access multiple WS suppliers that are widely distributed without having to remember ID and POW for each of the suppliers.
- UDDI as well as WSDL could inadvertently expose some services that the company would rather not publicize. Once a service is defined as WSDL and then further advertised through UDDI, then almost anyone can invoke it. Public UDDI Registries have suffered because many services they contain are not provided or adequately supported by the service providers. Thus a great deal of care is needed before advertising a service.
- Web services interoperability requires a security protocol that should also be interoperable. In particular, integrations based on WS raise serious security problems

because different systems with different security needs can be combined together by using WS.

These and other WS vulnerabilities and threats can be cast in terms of our familiar framework of privacy, integrity, authentication, authorization, accountability, and availability (PIA4):

- Privacy issues: XML messages as well as XML DTDs and schemas need to be kept private, especially for sensitive documents such as medical information and credit card payments. In addition, UDDI as well as WSDL should expose only desired features.
- Integrity issues: XML, DTDs/schemas, WSDL, and UDDI should be protected against modification by intruders. In addition, remotely located WS services could introduce malicious code, so the WS services from unknown or suspicious providers should be carefully watched.
- Authorization and authentication: Widely distributed resources accessible from multiple players expose many issues and require multiple sign-ons. Single sign-on is quite useful in this area.
- Accountabiliy issues: Because of different partners, NR (non-repudiation) is harder. In particular, there is no central log that can be used to keep track of all activities.
- Availability issues. New type of denial of service could be launched against UDDI, thus disabling the providers relying on the attacked UDDI.

11.4.3 Web Services Security Approaches

11.4.3.1 Overview

What are the different approaches to deal with the WS security issues mentioned previously? One of the major approaches is being jointly developed by IBM and Microsoft. This effort has resulted in a jointly published white paper on "Security in a Web Services World: A Proposed Architecture and Roadmap" (msdn.microsoft.com/library/en-us/dnwssecur/ html/securitywhitepaper.asp). This document defines a Web Service security model that supports, integrates and unifies several popular security models, mechanisms, and technologies (including both symmetric and public key technologies) in a platform-neutral manner. The specifications build upon foundational technologies such as SOAP, WSDL, XML Digital Signatures, Kerberos, XML Encryption and SSL/TLS. This allows Web Service providers and requesters to develop solutions that meet the individual security requirements of their applications.

While specifications are being developed, several things can be done to create secure Web Services in the current environments by concentrating on the following:

- Securing the XML documents that are exchanged between WS clients and servers.
- Securing the connection -- the path -- between WS clients and servers.
- Authenticating and authorizing the WS clients and servers.
- Maintaining interoperability between WS security solutions that cross organizational boundaries by using SAML (Security Assertion Markup Language).

11.4.3.2 Securing XML Documents

XML documents, like any other, can be encrypted and exchanged between WS clients and servers. This, however, does not handle situations where different parts of the same document need different treatment. For example, an XML document representing a credit card purchase may need to encrypt the credit card information and the purchase information differently. XML Encryption, discussed in the previous chapter, allows encryption certain segments of XML documents. XML security also allows different parties to digitally sign different segments of an XML document by using XML Signatures. We will not discuss XML Encryption and XML Signatures here because they have been explained in the previous chapter.

11.4.3.3 Securing the Connection

Securing the connection between the Web Services client and the server is usually the first step. As discussed in the previous chapters, Secure Sockets Layer (SSL), Virtual Private Networks (VPN), and firewalls are the most common approaches.

SSL can be used to establish secure connections between WS clients and servers on untrusted networks such as the Public Internet. SSL, as discussed in a previous chapter, encrypts and decrypts messages sent between a client and server. Digital certificates may be used between the WS clients and servers to augment the connection encryption. While SSL is very popular for secure communications in Web environments, SSL does introduce performance overhead that should be taken into account.

A VPN encrypts data between computers on physical links. While similar to SSL, a VPN operates at a lower level, thus all messages (email plus WS) are encrypted on VPN while only the WS messages are encrypted by SSL. VPN is more efficient than SSL because it does not suffer from the session startup/termination for each WS session. However, VPN does require a long-term connection between the computers

Firewall rules can be used to restrict access to computers where the WS applications reside. This is particularly useful for WS applications in a corporate Intranet where you are not concerned about encrypting the messages -- you only want to restrict access to particular apps within a network. Policy-based are especially useful when different clients may have access to different methods of the same Web Services. A wide variety of firewalls are commercially available at present to support these and other features.

In reality, a combination of firewalls, VPN, and SSL can be used to provide rich alternative security designs for WS security. A firewall, for example, can be used to limit access to certain WS methods and to protect the corporate perimeter while SSL and/or VPN can be used to protect the WS traffic over the Public Internet.

11.4.3.4 Authentication

For authentication in Web Services, the authentication features available for HTTP can be used. For example, the following options can be used:

- Basic: User ID and password are sent as encoded text. The server decodes this information and authorizes access to the WS if the ID/PW match a valid account.
- Basic over SSL: Same as Basic authentication except that the communication channel is encrypted, thus protecting the username and password.

- Client certificates over SSL: Requires each client to obtain a digital certificate for additional authentication. This option is only available over SSL connections, thus performance may be a concern.

Other approaches to implementing authentication in WS include using the Microsoft .NET Passport or creating a custom authentication method that includes multi-factor authentication.

11.4.3.5 Authorization

For authorization, access is typically determined by checking Access Control Lists (ACLs). It is possible for WS clients to have different degrees of access. For example, some WS clients may have full access to all operations of a WS service while others may only be allowed to access certain operations. ACLs can be used to specify these restrictions.

11.4.3.6 Interoperability between Web Services

Large scale WS applications can cross trust boundaries and span multiple enterprises. Thus the WS security developed for one application must be able to interoperate with others. Several initiatives for WS security are underway. The Security Assertion Markup Language (SAML), discussed in the next section, is the most comprehensive and is getting the most attention.

11.4.4 SAML (Security Assertion Markup Language)

Security Assertion Markup Language (SAML) is at the core of the WS Security architecture. SAML is an open security specification developed by the Organization for the Advancement of Structured Information Standards (OASIS – www.oasis-open.org). It uses XML to exchange security information in the form of assertions. SAML sponsors include companies such as Boeing, Citrix, HP, IBM, Microsoft, Netscape, and Sun. SAML's stated purpose by OASIS is "to define an XML framework for exchanging authentication and authorization information". SAML concentrates on authentication and authorization, not message integrity and confidentiality. In practical terms, SAML provides a standardized architecture for secure communications, helps secure Web Services, and facilitates Single Sign On (SSO) in WS environments. Since SAML does not directly provide message integrity or confidentiality; it relies on XML Signature to protect integrity and on SSL (Secure Sockets Layer) and TLS (Transport Layer Security) for confidentiality.

The basic SAML model is quite simple (see Figure 11-10). A requesting party submits “assertions” (security credentials) to an issuing authority. The issuing authority looks at the assertions and responds to the assertions based on some verifications. Once verified, the requester then uses these assertions to access protected information.

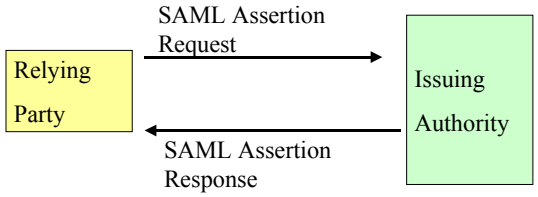

Figure 11-10: Simplified View of SAML

Let us look at SAML assertions in a little more detail. SAML is designed to exchange security information in the form of assertions, stated in XML, about a subject (person or computer that has an identity in the security domain and needs access to a protected resource). The subject, for example, may be a program that needs to read a corporate database. Three types of assertions are supported by SAML:

- an authentication assertion indicates how and when the entity's identity was proved,
- an authorization assertion declares that the entity is (or is not) authorized to access specific resources,
- an attribute assertion provides other information about the entity, such as membership in a group.

Assertions in the SAML token are digitally signed, so the entity can reuse the token for further requests without re-authenticating. This gives end users the benefit of single sign-on (SSO) so that an application that uses numerous WS services has to provide authentication only once, rather than once for each interaction.

Figure 11-11 shows a simplified version of SAML processing when a requesting entity needs access to a protected resource. A policy enforcement engine reads the SAML assertions about authentication, authorization, and attributes about the requesting entity. An authentication server handles the authorization and authentication processing. In addition to assertion, SAML includes a request/response protocol, bindings, and profiles to facilitate processing of assertions. SAML utilizes XML signatures, SSL, TLS, HTTP and SOAP for actual processing, but as pointed out previously, it is not an encryption method.

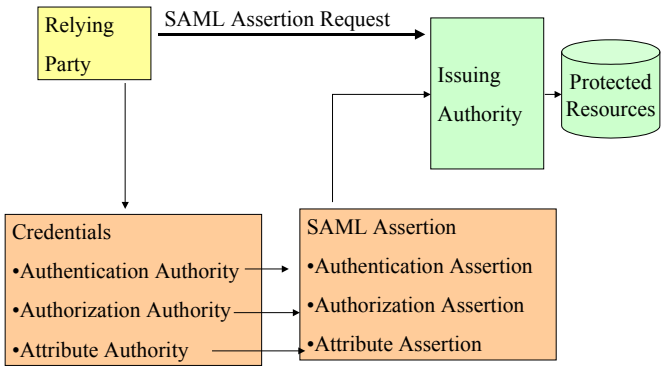

Figure 11-11: SAML Processing

The following is a sample SAML-compliant request that is sent from a relying party ("Umar") requesting password authentication by the issuing party ("nge.com"):

```
<samlp: Request ...>

 <samlp: AttributeQuery>

  <saml: Subject>
   <saml: NameIdentifier
    SecurityDomain="nge.com"
    Name="umar"/>
  </ saml: Subject>

  <saml: AttributeDesignator
   AttributeName="Employee_ ID"
   AttributeNamespace="nge.com">
  </ saml: AttributeDesignator>
 </ samlp: AttributeQuery>
</ samlp: Request>
```

A great deal of information about SAML is becoming available at the time of this writing. The following principal references are suggested for additional information: SAML specification documents (www.oasis-open.org) and "Security Assertion Markup Language (SAML)" (Main reference page --xml. coverpages.com/saml.html). In addition to SAML, the following security technologies need to be considered for a complete WS security solution:

- XML Encryption – Specifies process for representing encrypted XML and other data in XML
- XML Digital Signature – Can sign both XML and non-XML resources (JPG, HTML, XSL)

11.4.5 Web Services Security Summary

The following table summarizes the WS security issues and the possible approaches.

- Privacy issues: XML messages as well as XML DTDs and schemas need to be kept private, especially for sensitive documents such as medical information and credit card payments. In addition, UDDI as well as WSDL should expose only desired features.
- Integrity issues: XML, DTDs/schemas, WSDL, and UDDI should be protected against modification by intruders. In addition, remotely located WS services could introduce malicious code, so the WS services from unknown or suspicious providers should be carefully watched.
- Authroization and authentication: Widely distributed resources accessible from multiple players expose many issues and require multiple sign-ons. Single sign-on is quite useful in this area.
- Accountabiliy issues: Because of different partners, NR (non-repudiation) is harder. In particular, there is no central log that can be used to keep track of all activities. Each server site should keep an audit trail.
- Availability issues. New type of denial of service could be launched against UDDI, thus disabling the providers relying on the attacked UDDI. UDDI should be replicated for improved availability.

Table 11-1: Web Services Security Issues and Approaches

	Privacy	Integrity	Authentication and Authorization	Accountability	Availability
WS Specific Issues	XML and DTD of sensitive information should be kept private	XML, DTDs/schemas, WSDL, and UDDI should be protected against modification	Widely distributed resources accessible from multiple players expose many issues and require multiple sign-ons.	Because of different partners, NR (non-repudiation) is harder.	New type of denial of service could be launched against UDDI
WS Security Solutions	SSL encryption and XML encryption	Message digesting and XML signature (if needed) for integrity	SAML assertions about authorization and authentication	Audit trails at Web servers	Replicated copies of UDDI may be needed

11.5 .NET Security

11.5.1 Overview of .NET

A popular example of component-based platforms is Microsoft's Dot Net, commonly written as .NET. The Microsoft .NET platform includes a family of services and products, built on XML and components. At the core of .NET are ***XML Web services*** — in fact, .NET is Microsoft's platform for XML Web services. XML Web Services, also just known as Web Services, were introduced in the previous section. The .NET platform consists of a new operating system with Windows Server 2003 and Windows XP, and a new approach to writing distributed applications by using WS capabailities. Security on this new platform is paramount because applications no longer run on discrete machines but are dispersed and interconnected through the Internet and HTTP.

Despite its new image, Microsoft.NET is largely a rewrite of Windows DNA (Distributed Network Architecture), which was Microsoft's previous platform for developing enterprise applications. The underlying component model of XML Web Services implemented in .NET is closely related to the COM/DCOM (common object model) model that is part of DNA. DNA includes many proven technologies such as Microsoft Transaction Server (MTS) and COM+, Microsoft Message Queue (MSMQ), and the Microsoft SQL Server database. The new .NET Framework replaces these technologies, and includes a Web Services layer as well as improved language support.

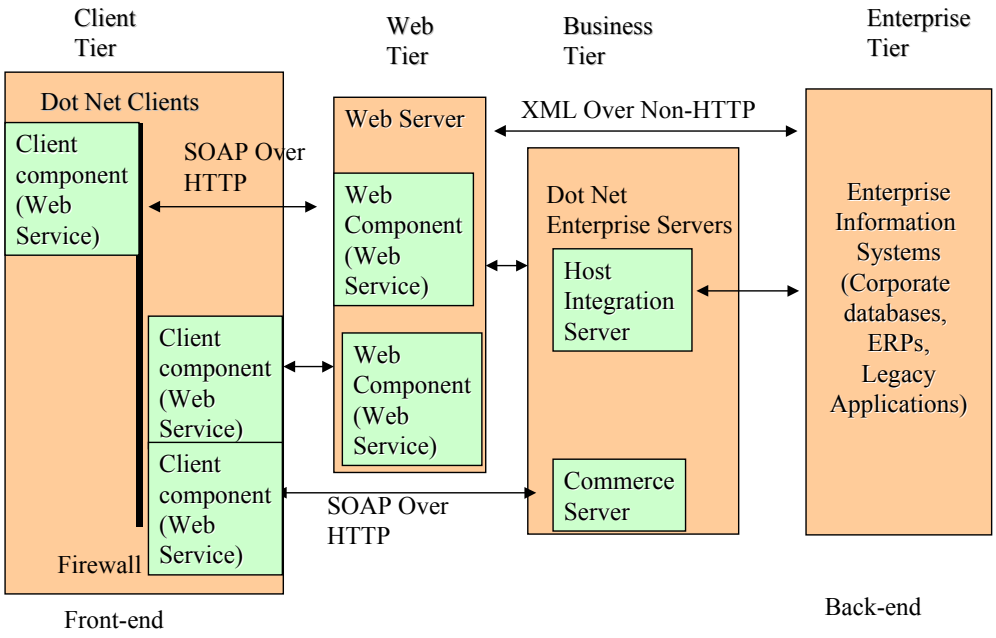

Figure 11-12: Microsoft .NET Architecture

Figure 11-12 shows the overall architecture of .NET (Note that the general architecture of .NET is quite similar to J2EE). It shows how the XML Web Services, shown as Web Services, appear as components in different tiers of the generic component-based architecture. These components communicate with each other by using SOAP (Simplified Object Access Protocol) that can be used to transport XML documents over HTTP. Specifically, Microsoft is building the .NET platform in terms of the following:

.NET Clients: Clients are PCs, laptops, workstations, phones, handheld computers, Tablet PCs, game consoles, and other smart devices that run operating systems such as Windows CE, Windows Embedded, Window 2000, and Windows XP. These clients can house Web Services as components.

.NET Enterprise Servers: The Microsoft .NET Enterprise Servers are intended for enterprise use and have the capabilities to communicate with back-end systems. Examples of these servers are:

- **Microsoft Windows .NET Server** to build, deploy, manage, and run XML-based Web services.
- **Microsoft Application Center 2000** to deploy and manage Web applications.
- **Microsoft BizTalk™ Server 2000** to build XML-based business processes across applications and organizations.
- **Microsoft Commerce Server 2000** for building e-commerce solutions.
- **Microsoft Content Management Server 2001** to manage content for dynamic e-business Web sites.
- **Microsoft Exchange Server 2000** to enable messaging and collaboration.
- **Microsoft Host Integration Server 2000** for bridging to data and applications on legacy systems.
- **Microsoft Internet Security and Acceleration Server 2000** for secure Internet connectivity.

- **Microsoft Mobile Information 2001 Server** to enable application support by mobile devices like cell phones.
- **Microsoft SharePoint™ Portal Server 2001** to find, share, and publish business information.
- **Microsoft SQL Server™ 2000** to store, retrieve, and analyze structured XML data.

Communication Protocols: .NET is built on top of a set of XML-based protocols such as Simple Object Access Protocol (SOAP), Universal Discovery, Description and Integration (UDDI), and Web Service Definition Language (WSDL). These protocols collectively allow Web Services to be described (through WSDL), discovered (through UDDI), and transported as XML messages over HTTP (through SOAP). These services, collectively known as SOAP/XML services, are discussed in Chapter 4 of this Module.

Web Server Sites: Web sites for accessing HTML documents - These sites can also house XML Web Services as components. XML Web Services on Web sites offer a direct means for applications to interact with other applications. XML Web Services applications hosted on Web sites, as well as on remote systems, can communicate via the Internet by using XML and SOAP messages.

Building Block Services: Microsoft is creating a core set of building block services that perform routine tasks and act as the backbone for developers to build upon. The first set of XML Web services are known as .NET My Services (the product formerly code-named "HailStorm"). These services are user-centric -- they use the Microsoft Passport user authentication system to deliver information to users based on preferences they have established.

Tools: Microsoft Visual Studio® .NET and the Microsoft .NET Framework provide a development platform for developers to build, deploy, and run XML Web services. They maximize the performance, reliability, and security of XML Web services.

- **Visual Studio .NET** allows developers to build applications by using Microsoft Visual Basic (of course!), Visual C++, and a new programming language called C# (see below). All languages supported by Visual Studio prior to the release of .NET (except for Java) are still supported by Visual Studio. Visual Studio.NET also provides support for Microsoft's new **C#** language which is semantically equivalent to Java, with just a few minor syntactical differences.
- **The .NET Framework** is an application execution environment that handles essential plumbing chores. The main feature of .NET Framework is language independence (discussed below).

Language Independence Through C#: Microsoft.NET offers language-independence and language-interoperability that is very close to Java concepts. This is a very interesting aspect of the .NET platform. A .NET component can be written, for example, partially in VB.NET, the .NET version of Visual Basic, and C#, Microsoft's new object-oriented programming language. This is how it works. First, the source code is translated into Microsoft Intermediate Language (MSIL or IL). This IL code is language-neutral, and is analogous to Java bytecode. The IL code is then interpreted and translated into a native executable -- the Common Language Runtime (CLR), analogous to the Java Runtime Environment (JRE). The CLR is Microsoft's intermediary between .NET developers' source code and the underlying hardware, and all .NET code ultimately runs within the CLR.

.NET Experience: The .NET experiences are XML Web services built for individuals and for businesses. Some of the products that Microsoft is transitioning into .NET experiences are:

- **Hailstorm** is Microsoft's portfolio of building block Web Services that will be hosted by Microsoft and possibly Microsoft partners. Some Hailstorm Web services will be available on a subscription basis, and others will be free. Information about Hailstorm is available at http://www.microsoft.com/net/hailstorm.asp.
- **Microsoft's Passport** offers an authentication service as a *shared context* so that many other Web Services can depend upon Passport for identity and security credentials. Passport is similar to the Certification Authorities found in PKI (Public Key Infrastructure) systems. Users can store their contextual information (who they are, what they are authorized to do, usernames, passwords, credit card information, etc.) into a single repository that can be shared by multiple Web Services and clients. With shared context, you type this information in once in a passport, and that information is then accessible to all Web services that you choose to give access to that information. The Passport service is now freely available as a Web service-oriented universal identification mechanism. We will take a closer look at .NET Passport in a later section.

The XML Web services, as stated previously, are at the core of .NET. A great deal of information about various aspects of Web Services is available at present over the Web. An example of the source is the Microsoft site (http://msdn.microsoft.com).

11.5.2 .NET Security – Main Feature

The .NET Framework includes a large number of security features that have been packaged under the .NET umbrella. These features include:

- Cryptography for the traditional privacy and authentication support plus XML encryption and Digital signature support.
- Role-based security for managing user identity to support authorization and authentication of principals.
- Evidence-based security for managed code based on different levels of trust and restrictions that can be controlled by the administrator.
- .NET Passport for single-signon across multiple machines.

For cryptography, the .NET framework includes traditional symmetric/asymmetric encryption, digital signatures, hashing, and random number generation. A wide range of algorithms are supported including RSA/DSA for asymmetric encryption, DES/TripleDES/RC2 for symmetric encryption, and MD5/SHA1 for hashing. .NET Framework also has accepted, in principle, the XML Digital Signature and XML Encryption specifications currently under development by the IETF and the W3C. As discussed in a previous chapter, XML signatures and encryption provide an easy way for application programmers to sign and encrypt XML documents and fragments.

Role-based security introduces a model for managing user (or automated agent) identity and roles for authorization. The basic idea is that a principal is identified as the user on whose behalf code is executing. Authentication procedures examine credentials (e.g. name/password and digital certificates, if needed) to establish the identity of the principal. Once identified, a principal may specify roles (e.g., administrator, contractor) to which it belongs. These roles can be used to represent authorizations. The system can

use the identity of the current principal and/or her role as necessary to perform some privileged operation. In the simplest case, the Windows logon identity of users becomes the principal identity, and the groups the user belongs to are the names of the roles assigned. For more advanced situations, a generic principal object is defined that can consult a database to retrieve password and list of roles for a principal.

Evidence-based security is used to support trust of code while role-based security deals with trust of users. As programs can invoke other programs in a .NET application space -- this is part of Web Services features -- evidence is needed that a program can be trusted to perform certain operations and not others. Thus, before any managed code runs, the security policy system determines what permissions to grant it based on evidence that about the code. The evidence can be a valid digital signatures, a URL, or anything else known about the code. After permissions are granted, then the code runs and is limited by what the permissions allow it to do. Security of code is enforced by restricting the code to only use well-defined interfaces. For example, this allows mobile code to be downloaded from unsecured sources and executed safely by restricting it to use only a few well defined interfaces that are safe.

Web applications in .NET use ASP.NET on the server side. ASP.NET supports security through basic cryptography, message digests, Kerberos, SSL/TLS client certificates, and access control lists. In addition, ASP.NET supports Microsoft Passport authentication and cookie authentication. ASP.NET architecture allows built-in support for role-based security with Windows users and groups and also supports application defined roles.

A great deal of information is available on Microsoft .NET security. In addition to the web links available through Microsoft, the following books may be worth a look for the serious .NET security professionals:

- "Programming .NET Security" by Adam Freeman and Allen Jones, O'Reilly & Associates; 1st edition (June 2003)
- ".NET Framework Security" by Brian A. LaMacchia et al, Addison-Wesley, April 24, 2002.
- "Essential .NET Security" from www.develop.com

11.5.3 .NET Passport – A Possible Solution for Single Signon

Microsoft .NET Passport enables participating sites to authenticate a user with a single set of sign-in credentials. Passport was launched in 1999 to provide a single-signon (SSO) facility for the Internet user. By now, it is one of the largest online authentication systems in the world because it eliminates the need for the users to remember numerous passwords and sign-in names. The .NET Passport is designed for users who want to establish their identity once and then move smoothly among various Web sites. It also facilitates the businesses to confirm the consumer's identity as the consumer moves from one site to another.

SSO solutions such as .NET Passport are motivated by the continued increase of Internet users who use multiple sites -- an International Data Corp. (IDC) report indicates that the number of Internet users worldwide will double between 2001 and 2006. In addition, we all forget our passwords due to the multitude of sites we visit -- according to Jupiter Media Metrix, between 10 to 30 percent of customers forget their passwords. In addition,

some sites require less security than others. For example, a site that provides weather information should not need any security while a bank transfer must.

Passport attempts to provide multiple levels of authentication through a single signon. For example, if a site has standard security needs, then the user can use the name and password in Passport. For stronger security needs, the user can be required to enter additional secondary attributes such as a secure ID card number. Still more security can be enforced by adding biometric information such as fingerprints. Thus multiple layers of security can be used to provide a balance between usability and security.

To use the Passport, the consumers go to http://www.passport.com/ and create an account with nothing more than a user name and password. Additional information can be added if needed. On the other side, companies willing to deploy and use Passport on their Web sites sign the nonexclusive .NET Service Agreement. This agreement commits businesses to guidelines that are designed to help protect the personal information of Passport users and help ensure the integrity of the system. For example, the contract states that the participating sites must post a privacy statement online and make it readily accessible to their users. P3P (Platform for Privacy Preferences) from W3c could be used for this purpose. Privacy policies are being posted by different companies. Privacy policies for Microsoft.com can be found at (http://www.microsoft.com/info/privacy.htm) and for ..NET Passport at (http://www.passport.net/consumer/privacypolicy.asp).

As expected, Passport is quite controversial because attempts to provide a balance among many competing priorities such information versus anonymity; security versus usability; and the needs of companies versus those of the consumers they serve. Naturally, different views exist and different approaches are being suggested. For ongoing developments in this area, visit the Web site (http://www.passport.net/).

11.6 Short Case Studies and Examples

11.6.1 Web Services Security by Using a Firewall

A diversified financial services institution attempted to improve its Employee Benefits Administration (EBA) outsourcing business by using XML Web Services (WS). The WS is intended as the integration standard to serve existing and new customers and to offer new products and services such as Long Term Care Insurance.

Security and employee/user privacy emerged as key issues as the company embarked on defining their XML Web Services. Interoperability became a big issue because EBA realized that few of their customers and partners supported the same security standards or used the same XML Web services platforms. A solution was needed that would interoperate across platforms as well as support a variety of security standards and policies.

The EBA evaluated a number of alternatives for security and interoperability. The first option was to build security directly into XML Web Services. This implied that the application developers integrate security infrastructure and interpret standards requiring code audits by security team. This, in turn, made the software developers the long term

managers of security implementation because changes to security policy would require additional software development. The other choice was to re-program existing systems such as single sign-on (SSO), Web Servers and Web Application Firewalls. This implied a significant investment in overcoming technical barriers such as Web server programming to handle business to business transactions with need for atomicity and auditability. In addition, SSO reprogramming to support XML Web services is non-trivial. The third option was to select a package that would provide policy management and integrate existing platforms and infrastructure.

After some analysis, the company chose the Reactivity XML Firewall Appliance (www.reactivity.com) as an implementation choice. EBA deployed the Reactivity XML Firewall in multiple clusters in multiple data centers with remote security management.

Source: http://www.reactivity.com/products/case_studies.html

11.6.2 Microsoft .NET Security at Wakefield Public Schools

Wakefield Public Schools is a Boston suburban district that serves 3,500 students, has a staff of 350 teachers and administrators, and includes eight schools that span elementary to high schools. The district has 700 PCs for instruction and another 75 PCs for administrative use. Until recently, these computers were not connected into a district-wide network. In 2002, the district networked all of its facilities via high-speed fiber. That presented new opportunities for state-of-the-art collaboration, Internet access and more. But it also presented major challenges in securing its servers and desktops from unauthorized internal and external access, and in ensuring that the students would not make improper use of the Internet or the network.

After some analysis, with help from a Microsoft Service Provider (TENCorp), the district deployed a .NET infrastructure based on Windows 2000, ISA Server and the Active Directory service. ISA Server provided a powerful firewall and Active Directory provided a centralized place to manage users and resources while acting as the central authority for network security. This configuration provided different levels and types of security for different users (teachers, administrators and students with broad range of ages and grade) through a variety of security profiles.

To deploy the network, the team installed five separate Windows 2000 servers at the high school to function as a domain controller. At high school it also installed an IIS (Internet Information Server) Web server, an Exchange 2000 Server for email and collaboration, a Main ISA server, a file server and an application server. In each of the other facilities, the team installed a single Windows 2000 server with ISA Server, to support all domain control functions including security. TENCorp also created low-cost "backup servers" from existing equipment to ensure system availability if the local server is unavailable.

The ISA Server works seamlessly with Active Directory to implement group security policies and to filter content. Because of this integration between ISA Server and Active Directory, the district only needs to maintain user permissions in Active Directory and the information is automatically available to ISA Server. The Active Directory and ISA Server solution was extended with the third-party content-filtering solution CyberPatrol -- it gives 38 categories of content filtering. The security and filtering worked well because the district saw and blocked, in the first months after deployment, several

thousand attempts to go to inappropriate Web sites. In cases where a student tried repeatedly to go to an inappropriate site, the IT team notified the student's teacher for a conversation with the student.

The school district is planning refinements to the structure of groups and the policies that apply to them in the future. For example, the district will review and modify the access rights extended to middle school students, based on the student's past use of the Internet. Wakefield is also planning the technology infrastructure for a new elementary school, as well as the continued migration of existing desktop PCs to Windows 2000.

Source:
http://www.microsoft.com/technet/treeview/default.asp?url=/technet/itsolutions/education/case/wakefiel.asp

11.6.3 Microsoft .NET Passport Helps Student Loans

Panhandle-Plains Student Loan Center (PPSLC) is a Texas-based student loan service for partner lending institutions throughout the state of Texas. PPSLC helps students find and obtain financial aid. With increasing use of its Web site for managing account information, PPSLC needed to improve its user sign-on system. The company decided to use Microsoft .NET Passport technology to make access to student loan services more convenient and secure. This was especially convenient for the loan company's many college student account holders who already are using the .NET Passport.

PPSLC's old system utilized borrowers' Social Security numbers and dates of birth as their login names and passwords. However, this information was accessible to family members. For example, the loan center had received complaints from borrowers whose former spouses were accessing their accounts without permission. In addition, the borrowers could not change their passwords (it is tough to change your date of birth, although many of us would like to do that!).

.NET Passport allows the user to set his or her username and password. The PPSLC users enter their Social Security numbers, dates of birth, and zip code only once. This information is stored at the PPSLC site and is mapped to the user's .NET Passport credentials. After the initial sign in, subsequent visits to the site require only the .NET Passport username and password.

Source: http://www.microsoft.com/

11.7 Review Questions

1) Why is it important to consider the security issues related to the modern distributed computing platforms? What type of risks an organization runs into if the security of these platforms is not properly addressed.
2) What are the main approaches to secure an operating system?
3) When and why should host firewalls be used?

4) How will you assure that a database and a transaction manager is secure?

5) Many middleware security issues are discussed in this chapter. In your view, which ones are most important and why?

6) What are the unique issues in WS security and what are the main approaches used to address these issues?

7) What is SAML and what role does it play in WS security?

8) What are the main issues and approaches in .NET security?

9) What is .NET Passport and why it is important?