Eavesdropping

Eavesdropping attacks listen to and decode request messages.

The request parameters and request results that are sent from a client object to a server object or vice versa may include sensitive information. We have seen that the client and server stubs marshal these parameters into a transmittable form. Very often the format that is used for this transmission is well known to an attacker, because it is based on a standard data representation, such as NDR, XDR, or CDR. By listening to or sniffing the network packets that are transmitted when implementing an object request, attackers can therefore obtain and decode the request parameters. We refer to this method of attack as *eavesdropping*.

Eavesdropping is possible because the attacker knows how object requests are encoded. Object requests, however, have to be encoded in a standardized form in order to facilitate interoperability. In order to prevent eavesdropping, we must, therefore, avoid transmitting object request data in an encoding that is known to other people. This is generally achieved by encrypting the messages before they are sent through the network and decrypting the messages immediately after they have been received.

Example 12.7

Eavesdropping Attack on a Bank Account

In the banking application discussed in Example 11.2 on Page ●●●, an object request `get_balance` returns the balance of an account object. If we assume that the object request is executed using CORBA, the floating point number that represents the balance will be encoded using CORBA's CDR representation. Moreover, the IIOP protocol demands that the name of the operation is included in the marshalled representation of the object request, too. The message will also identify whether it is a request or a reply. An attacker who wants to obtain balances of account objects can then eavesdrop and wait for packets that include the string `get_balance`. The requester will then know from the encoding of the type of message whether it is a reply message and can then obtain the balance that is encoded as float in CDR.

Request Tampering

Tampering attacks modify request messages.

Request tampering builds on, and goes beyond, eavesdropping. The idea of request tampering is that request messages are intercepted before they reach the server object. The attacker modifies request parameters and then passes the message on. Using request tampering, an attacker can modify information without having to masquerade some other user's identity. The ease with which request tampering can be used depends on the type of network protocol that is being used. We should however assume that network protocols do not exclude the modification of information between senders and receivers.

Example 12.8

Let us assume that an attacker wants to tamper with a credit request in order to unlawfully increase his or her account balance. Assuming again a CORBA-based distributed object system, the credit request will be implemented using an IIOP request message. The attacker could force the generation of such a message, for example by banking a cheque with a small amount of money. The attacker would then intercept the message and modify the parameters of the `credit` request to a high amount of money before passing the message on to the intended receiver. Instead of generating the request message, the attacker uses a request message generated by an authorized user of the system and just modifies the message to credit a higher amount.

Request tampering can be prevented in the same way as eavesdropping. If a requester cannot decode the request message then he or she can also not modify it. Hence encrypting request and reply messages that implement object requests is the basic mechanism that is used to prevent request tampering.

Replaying

Attackers may compromise the security of a distributed system, even if they are not able to tamper with any request or reply messages. They can intercept and store a request message and then have a server repeatedly execute an operation by *replaying* the message. Note that the attacker does not have to be able to interpret the message and therefore the encryption mechanisms that can be applied to defeat eavesdropping and request tampering cannot prevent replaying.

Replaying denotes the repetition of request messages.

Example 12.9
Replaying Attack on a Bank Account

Consider that an authorized client requests a `credit` operation to an account. In order to prevent eavesdropping and tampering, the designer of the banking application has ensured that the message is encrypted. An attacker who captures a message is not able to interpret the message. However, the attacker can repeatedly send the captured message and thus compromise the integrity of the banking application because the server object will not execute the credit operation once but multiple times. This leads to the balance of the account being unduly increased. An attacker who has other ways than reading the message context to associate the encrypted credit message with a particular account will then be able to increase the balance on that particular account.

Replaying is difficult to prevent. The obvious solution is to tightly control the flow of network packets. This reinforces the concern we raised earlier about separating private from public networks, which is achieved by firewalls that we will discuss below.

12.1.4 Infiltration

For all of the above methods it is necessary that the attacker obtains access to the network to which server objects are connected. There are three ways that an attacker can obtain the necessary network access.

Attacks by Legitimate Users: Many security attacks are, in fact, launched by legitimate users of the system, who attempt to use the system beyond what they are authorized to do. Thus they abuse their existing authorization for attacks. Auditing is an important measure to tackle such infiltration by detecting and recording any illegitimate usage. We will discuss auditing in the sections below.

Obtaining a Legitimate User's Identity: Most operating systems force users to identify themselves with a password during login. The security of the system critically depends on how these passwords are chosen. It is not difficult to obtain the list of legitimate users of a system and then an attacker can write a *password-cracking program*, which tries words from a dictionary for each of the legitimate users. The system administrators of my Computer Science department once found that 40% of the passwords were girls' first names. As those

A password cracker tries to break the login of a legitimate user with a large dictionary of words.

are commonly included in a dictionary, attackers have a significant number of user identities at their disposal. Again these attacks can be identified by auditing. They can be prevented by choosing proper passwords, which is the responsibility of every user. Because this cannot be enforced and validated, organizations often connect the hosts on which their users work to a private network and strictly control the network flow between the private network and public network with a firewall.

Smuggling Client or Server Objects: The most subtle form of infiltration is the smuggling of executable objects that act as servers or clients into a private network. This can be achieved by means of viruses or worms.

A virus is code that travels with an object to overcome a security barrier.

Viruses exploit object mobility. A virus attaches itself to an object and when the object migrates from one host to another, for example as a result of a load-balancing operation, the virus replicates itself by attaching a copy to other objects that it finds. The execution of a virus is often triggered by a particular date.

Worms are programs that use facilities to execute code remotely on other hosts.

The basic mechanisms to write and execute *worms* are provided by the mobility mechanisms of the middleware. The code of an object may be written on some host and then migrated to another host. An attacker can request execution of some operations after it has been migrated to that host. Whether it is actually possible to execute a worm depends on the installation and administration of the factories that are necessary for object migration purposes.

A distributed object virus or worm would typically start by investigating the interface repository or the type library that is available within the domain of an object-oriented middleware in order to discover information about the types of server objects that are known in that domain. It may traverse name servers in order to discover the object references of server objects that are available in a particular domain. It would transmit that information back to the attacker who then has the object references and the type information that is necessary to write more meaningful attacks, which could again be infiltrated using a virus or worm.

12.2 Encription

We have seen above that eavesdropping and message tampering are based on the assumption that the network packets that implement an object request between a client and a server object can be interpreted by an attacker. In order to prevent those methods, we will have to avoid sending object requests in plain messages across the network. Masquerading assumes that an attacker can use another user's identity. To prevent masquerading distributed objects must not trust any object that has not been authenticated, that is the objects must prove that they are who they claim to be. The basic primitives both for avoiding plain messages and authentication are provided by encryption techniques.

Encryption uses an algorithm and a key to convert plain text into cypher text and vice versa.

Encryption is a very old technique. It was used by the Romans to avoid messages carried by a messenger being meaningful if the messenger was captured by an enemy. *Encryption techniques* use an algorithm that encapsulates a large set of encryption functions and the key selects one such function. The encryption functions have to be chosen in such a way that it is difficult to perform the inverse function. That means it has to be computationally difficult to create the plain text from the cypher text without having the key.

Two methods of encryption can be distinguished. The first method uses *secret keys*, which are disclosed to only the two parties wishing to communicate securely. The second method uses *public keys* which each party makes generally available for secure communication with that party. A problem that arises then is how keys are distributed in a trustworthy way between two parties and we discuss protocols for the secure distribution of keys.

12.2.1 Secret Key Encryption

Secret key encryption is based on the idea that every pair of components that wish to exchange messages securely have a key that is not known to any other component; the key is kept secret by these two components. Secret key methods typically use the same key for both encrypting and decrypting messages. This means that the encryption algorithms are chosen in such a way that the same key selects a function and its inverse function. Since the secret keys used for the encryption are not publicly available, the encryption and decryption functions may be public.

Secret keys are known to two parties and not disclosed to any others.

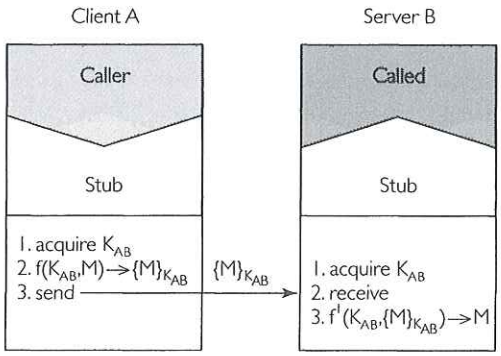

Figure 12.1
Secure Distributed Object Communication with Secret Keys

Figure 12.1 shows how secret key methods work. Sender A acquires key K_{AB}, which has been set up for communication with the receiver B. It then applies the encryption function to the message M that is to be transmitted and parameterizes the encryption with K_{AB}. This results in an encrypted message $M_{K_{AB}}$, which is transmitted to the receiver. The receiver B also has to acquire the same key K_{AB} to decrypt any message received from A and it does so by applying the inverse encryption function f^{-1} to the encrypted message $M_{K_{AB}}$ and the key K_{AB}.

Figure 12.1 also suggests how to employ secret key methods to secure the communication between distributed objects. To implement a request, the client object is the message sender and the server object is the message receiver. The encryption and decryption is performed after the stubs have completed request marshalling and unmarshalling and it has been recognized that the server object is not local. Then the marshalled object request is transmitted in an encrypted form via the network and secured against eavesdropping and message tampering. When receiving the message on the server-side, the message is decrypted and then passed on to the server stub. The same key can then be used for the reply message. We also note that encryption can be kept entirely transparent for client and server