John H. Parodi
Fred W. Burgher

Integrating ObjectBroker and DCE Security

The integration of the ObjectBroker software product with the Distributed Computing Environment (DCE) Security Service makes ObjectBroker the most secure object request broker (ORB) in the industry. ObjectBroker and DCE Security together allow client-to-server, server-to-client, and mutual authentication. The integrated software provides these security functions, as well as message integrity protection, transparently to the applications. Integration has been accomplished in a way that allows plug-in replacement of the ObjectBroker security subsystem by DCE Security, Kerberos, or any third-party software security product that supports the DCE's Generic Security Service Application Programming Interface (GSS-API). This approach supports future GSS-API–compliant third-party security products based on Kerberos and also products that may address other security technologies such as biometrics and smart cards. In addition, the approach places responsibility for compliance with International Traffic in Arms Regulations in the hands of the purveyors and owners of GSS libraries rather than with the ORB vendor. Note that the ObjectBroker product is middleware jointly developed and distributed by DIGITAL and BEA Systems, who have formed a worldwide technology and distribution partnership.

An object request broker (ORB) is a distributed software layer that translates abstract service requests from a client application into requests for specific servers, regardless of where those servers actually reside on the network.[1] In this way, ORBs provide a middle tier in multitiered client-server systems. The ObjectBroker software, developed and distributed by strategic partners DIGITAL and BEA Systems, is an implementation of the Common Object Request Broker Architecture (CORBA) specified by the Object Management Group (OMG).[2]

Security is a growing concern for those who manage distributed computing systems, and the security options available to the CORBA community have been quite limited until recently. In the past year, OMG has adopted a specification for a CORBA Security Service, although few commercially available implementations exist at the time of this writing.

Outside the CORBA community, one widely accepted standard for security in distributed, heterogeneous systems is the Generic Security Service Application Programming Interface (GSS-API),[3,4] as specified by The Open Group (which was formed by the merger of the Open Software Foundation and X/Open Company Ltd.).[5] The GSS-API provides the ability for software entities in a distributed application to authenticate one another and to protect ongoing communication between them. The Distributed Computing Environment (DCE) Security Service provides an implementation of the GSS-API as one way to access its security services.

Plans are under way to implement the CORBA Security Service in the ObjectBroker software, but the implementation specifications were not complete when ObjectBroker version 2.6 was designed. At present, by integrating support for GSS-API implementations, the ObjectBroker software provides its customers state-of-the-art distributed system security with the widest choice of security technologies and products. The first commercially available GSS-API implementation was the Kerberos-based DCE Security Service itself, but other implementations, which use a variety of security technologies and are produced by various independent software vendors, are expected to follow soon.

Security

Ensuring secure communication among entities in a distributed computer system is a challenging task. The term security normally includes three broad classes of system requirements:[6]

1. Secrecy/privacy—the ability to protect information from unauthorized access
2. Integrity—the ability to protect information from unauthorized alteration or destruction
3. Availability—the ability to ensure that valid access to information can be accomplished in a timely manner

Enforcement of a security policy is accomplished by way of the following security functions:

- Authentication—the verification of the identity of a security principal
- Authorization—the determination of which principals can perform which actions
- Access control—the enforcement of the security policy, based on authentication and authorization information, to determine whether to allow or disallow a particular action

The Distributed Computing Environment

The Open Group's Distributed Computing Environment is an integrated, standard set of technologies, tools, and services that enables the development and deployment of distributed applications in a heterogeneous, multivendor computing environment.[7] Typically, system vendors implement the DCE on their own platforms. The DCE has been endorsed by virtually all system vendors, including IBM, HP, DIGITAL, NCR, Stratus, Cray, HAL, Hitachi, Siemens Nixdorf, NEC, Data General, Bull, Tandem, Transarc, SCO, Gradient, Siemens Pyramid, and Olivetti.

The DCE provides the following six technology components:

1. Remote Procedure Call (RPC), which facilitates distributed communication
2. Directory Service, which provides a single naming model throughout the distributed environment
3. Security Service, which provides reliable authentication, authorization, and data protection
4. Distributed Time Service, which synchronizes the network system clocks
5. Distributed File Service, which provides access to networkwide files
6. Threads Service (The DCE uses POSIX threads where available; on operating systems where POSIX is not available, the DCE supplies a threads package that provides the same interface as POSIX threads.)

DCE users can be characterized by their need to deploy and/or integrate large-scale applications on multiple heterogeneous platforms. The most common reasons given for choosing the DCE are its security features, its scalability, and its robustness.

DCE Security provides the following services:

- The DCE Authentication Service allows users and resources to prove their identity to each other. This service is currently based on Kerberos, which requires that all users and resources possess a secret key.
- The DCE Authorization Service verifies operations that users may perform on resources. A DCE Registry contains a list of valid users. An access control list associated with each resource determines valid users and the types of operations a user may perform.
- The DCE Data Integrity Service protects network data from tampering. Automatically generated cryptographic checksums are appended to network transmissions, allowing the DCE to determine if data has been corrupted in transit. The encrypted checksum is a message authentication code (MAC) based on the Data Encryption Standard (DES).

ObjectBroker uses the DCE Authentication and Data Integrity services.

ObjectBroker Security

Although DCE Security provides three basic levels of protection (None, Data Integrity, and Privacy), ObjectBroker uses only the Data Integrity level. This level provides a mechanism that computes an encrypted, time-stamped checksum and attaches it to the message so that any attempt to change or replay the information can be detected. In addition, ObjectBroker uses explicit calls to the DCE Security library's GSS-API to accomplish authentication but maintains its own access control lists and authorization database and mediates access control itself.[8]

Note that within a DCE cell, it is possible to use the DCE RPC with the DCE Security Service to protect communication at the wire protocol level. However, because ObjectBroker does not use the DCE RPC wire protocol, its use of the DCE Security Service is accomplished by means of explicit calls by ObjectBroker to the GSS-API implementation.

ObjectBroker's use of the DCE Security Service provides data integrity protection, authentication of clients to servers and servers to clients, and protection against replay and sequencing attacks. Although encryption is used to create the digital signatures that provide these protections at the network Data Integrity level, ObjectBroker does not directly support the capability to encrypt data, even on nodes that have Privacy-level DCE Security Service support. ObjectBroker provides no protection from denial of service attacks either.

Of course, a customer's use of DCE Security is entirely optional, and the security mechanism used in previous versions of the ObjectBroker software is still supported. With this mechanism, called trusted security, the node/username associated with a request from a remote node is accepted to be as claimed. For trusted security, ObjectBroker uses a proxy approach in which the node/username associated with a remote request is mapped to a proxy identity on the server's system. An access control decision is thus based on the authorization information for the proxy identity. The proxy approach to the trusted security mechanism was necessary because there was no concept of global identity for a user, that is, an identity known to all computer nodes in a distributed system.

To implement DCE Security on a particular platform, a Security Integration Architecture accomplishes the mapping of a globally understood username (e.g., a user or a security principal defined within a DCE cell or a Kerberos realm) to a login of a local user on a particular system. Some implementations of DCE Security and some systems (for example, the OpenVMS operating system) use the notion of integrated or global login, in which a local user login also causes a global user login to be performed. For the OpenVMS system, the global realm is the cluster. For the implementation of DCE Security on the DIGITAL UNIX system, the global realm is the DCE cell.

Because an ObjectBroker configuration can include platforms that have no implementation of the DCE, and because the Security Integration Architecture is different on every DCE platform, there was no common mechanism for ObjectBroker to use to implement an integrated global login across all supported platforms. Thus, ObjectBroker is limited by the integrated login capabilities available on other platforms' implementations of the DCE.

For this reason, ObjectBroker retains a proxy mechanism, even for use by nodes that support the DCE. For authentication between such nodes, a generic remote host definition (called SecGlobalName) is mapped to a local user on the local system. Should a server receive a request that requires authentication from a client node, the server uses SecGlobalName to attempt to match the corresponding global principal name to a local user name.

In other words, because there is no common global identity mechanism, ObjectBroker's proxy implementation maps either a trusted remote user or a global user identity to a local system identity to accomplish a generic mapping between global and local users. Rather than map multiple host/username pairs to the local proxy, the ObjectBroker software maps a single SecGlobalName, known to all nodes in the DCE cell, to that proxy whenever possible.

Mechanism for Global Authentication

The DCE Security Service provides the mechanism for global identity. The mechanism is based on Kerberos encryption, which is a private or symmetric key scheme (as opposed to a public or asymmetric key scheme). A private key scheme requires some trusted third-party node to act as a distribution center for encryption keys or credentials. Each node or user has a key that is known only to the user and the distribution center. In DCE Security, the distribution center is known as a privilege server.[9]

The following is a simplified description of the encryption key protocol between the privilege server and a client. The actual key exchange protocol, which uses three exchanges and conversion keys, results in a Privileged Access Certificate (PAC) in the possession of a client. The PAC, which is appended to each request, contains the authorization information to be compared with the access control information stored with the application server.

When a client wishes to communicate with a server, each must acquire a time-stamped session key for secure communication. The session key is protected in several ways. The time stamp means that the key is only valid for a limited time (the amount of time is configurable), which protects against brute-force attempts to break the key and reuse it. Also, each key is host-specific and can only be used from the node for which it is issued. Finally, the session key is never sent over the network in unencrypted form.

For a user to initiate a DCE_login, the client must enter its DCE_login password. To register as an initiator and acceptor of security contexts, a server uses a SERVTAB key file. This file contains an encrypted key that permits the server to obtain a set of credentials similar to those given to a user. These credentials allow the server to accept security contexts from clients or to initiate requests (that is, become a client) to other servers. The reason for having servers acquire credentials through the SERVTAB mechanism is that servers may be started on demand by the ObjectBroker Agent (the component that locates the appropriate server to satisfy a client request) or by system administrators who do not want to be burdened by having to know a server password.

In either case, the client or the server specifies the principal name to be authenticated. The node sends the specified principal's name to the privilege server. The privilege server returns a session key that is encrypted using the principal's password or SERVTAB key. The DCE run-time software running on the local system decrypts the session key using the password or SERVTAB key. Once the client and the server have decrypted session keys, they can use the keys to initiate secure communication with each other.

Thus, if a server is configured to require authentication, then before invoking a method on that server, a client must successfully perform a DCE_login and obtain the credentials needed to establish a security context with that server. A client may also require authentication from the server to ensure that some malicious software is not masquerading as a real server.

Note that the operations for acquiring credentials are accomplished outside the server executable. The operations are performed by the ObjectBroker run-time software, based on configuration settings in the ObjectBroker Security Registry. The goal is to avoid burdening applications with the knowledge of security mechanisms.

Authentication requirements can apply to the ObjectBroker Agent as well as to clients and servers. The Agent is in fact a separate security principal, and one can require client-to-Agent, Agent-to-client, Agent-to-server, and server-to-Agent authentication in an ObjectBroker configuration—in addition to authentication between the client and the server. The client or the server can independently set these modes, or the ObjectBroker system can require that modes be set nodewide.

Security Design Issues for ObjectBroker

The security issues associated with the design of ObjectBroker versions 2.6, 2.7, and 3.0 were primarily those of increasing the security capabilities and preserving upward compatibility with previous ObjectBroker versions. While compatibility is always a concern when upgrading software, ObjectBroker's requirements in this area are particularly stringent because customers have mission-critical applications running in very large configurations. In some cases, it is difficult or impossible to upgrade all ObjectBroker nodes at one time, so it must be possible to do a rolling upgrade that minimizes the disturbance to the configuration and allows uninterrupted operation of applications.

The need for dynamic, plug-in replaceability of the security subsystem was an important issue for two reasons. First, to provide standards-based solutions to computing problems, the ObjectBroker design had to allow the integration of any security product that implements the GSS-API. The second reason has to do with export controls.

United States government export regulations specify that hardware, software, and documentation for cryptographic products may be exported by license only. Specifically, the Department of State's International Traffic in Arms Regulations (22 Code of Federal Regulations Subchapter M) require that an export license be obtained from the department before any cryptographic hardware, software, or documentation is exported from the United States. An ObjectBroker design goal was not to encumber the product with export restrictions. Therefore, ObjectBroker itself does not include any cryptographic security mechanism. An ObjectBroker customer must provide an appropriate GSS library; whatever package is available on the system is the one ObjectBroker will use.

ObjectBroker Security Features

The security features that have been successfully implemented in the ObjectBroker software include

- Client-to-server, server-to-client, and mutual authentication
- Protection from replay and sequencing attacks and integrity protection
- Fine-grain control over the authentication mechanism (per-host, per-server, or per-method)
- Ability to demand a new security context for an invocation
- Ability to apply new security features to applications without rebuilding them
- Dynamically loadable security libraries

Usage

One of the most important characteristics of a secure ORB is that applications (clients and servers) need not be aware of security operations undertaken on their behalf. For ORBs, as well as for other support software, the goal is to avoid burdening applications with the need to deal with the complexities of a distributed system so that they can concentrate on the application problem at hand.

Therefore, most of ObjectBroker's security-relevant operations are invisible to applications. ObjectBroker's management utilities are used to specify the rules for authenticating clients and servers. These rules are stored in the ObjectBroker Security Registry, and the required authentications are performed automatically.

There are two exceptions to the general rule of keeping security operations invisible to the application. The first is that a client or a server (when acting as a client) can explicitly make a call to an ObjectBroker API to toggle mutual authentication on or off. This operation is allowed as long as it does not diminish the security level specified for the ObjectBroker node as a whole. In other words, a client can demand mutual authentication on a node that does not require such authentication but cannot disable mutual authentication if the node does require it. This feature was implemented to make it possible for clients to enable mutual authentication for specific operations that have security relevance.

The second exception is that a server can demand the creation of a new security context for an invocation, which immediately tests the authentication of the principal making the request. This is important because the GSS-API allows the initiation of a security context that has no expiration. Clearly, if a security context exists for a long enough period, there may be a concern that it is no longer valid. For example, when a user's account is revoked from the DCE Security Registry, it is possible that the user's credentials are still valid in some existing security context. Establishing a new security context forces the DCE run-time software to go back to the security server and verify the validity of the principal.

Figure 1 illustrates the interaction of ObjectBroker and the DCE Security Service components in the establishment of a security context. Once the security context is established, it is used in the verification of MAC-sealed messages between the server and the client. In this illustration, access to the DCE security subsystem is depicted as a local call, though accessing these services could also be done remotely.

The sequence of operations in Figure 1 is as follows:

1. A method invocation (a client request for a remote operation) results in a call to ObjectBroker's security subsystem.
2. The ObjectBroker security subsystem in turn invokes a GSS routine in the DCE Security library. This call determines whether a new security context needs to be established, which can happen for one of two reasons: either it is the first invocation of this server from this client or the context refresh rate has been specified as per-invocation.
3. The DCE Security library executes the call, which sets up the security context. (Note that the process of deleting an existing security context is not shown.)
4. The security subsystem checks the return status of the GSS routine to determine whether the resulting token is to be passed to the invocation layer.
5. If so, the token is passed to the transport layer for marshaling.
6. The client communicates with the server node through the normal ObjectBroker channel.
7. The transport layer in the receiving node unmarshals the message, examines the transport message header, and passes control to a dispatcher in the invocation layer.
8. Depending on the message type, the message may then be passed to a special dispatcher, in this case the security dispatcher in the security subsystem.
9. The security subsystem determines that the message should be handled by the GSS implementation and passes the message there.
10. The DCE Security layer checks the received token and if it is valid, accepts the security context. The routine generates a context establishment token to be passed to the client. The call also returns the server's context handle for the security context the server shares with the client.
11. The security layer passes the token to the invocation layer for marshaling.
12. The invocation layer marshals the information and sends it as an argument to the low-level transport routine call.
13. This message is sent to the client.
14. The data is unmarshaled.
15. The message is sent to the security subsystem.
16. The token is passed to the GSS implementation to initialize the security context, with the server-supplied token as an argument. The routine returns the client's context handle, which is used to sign subsequent messages.

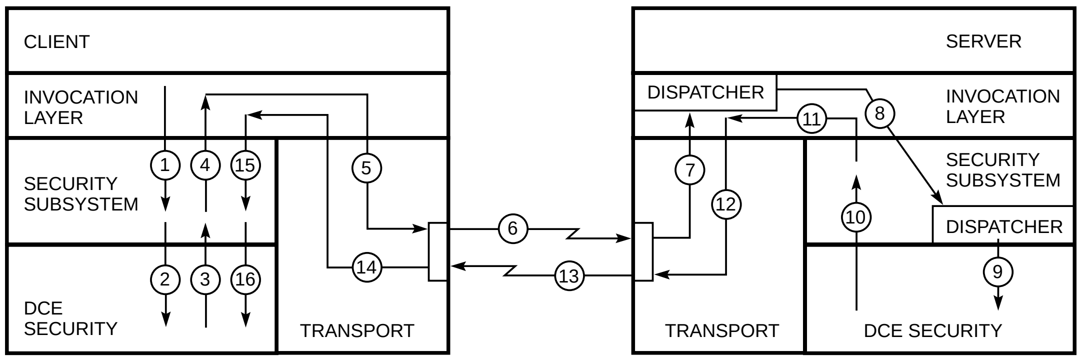

Figure 1
Establishment of a Security Context

Performance Considerations

The benefits of a secure ORB are not free. If authentication is required when a client and server establish a connection through a binding, part of that binding involves the establishment of a security context. Establishment of a security context requires a round-trip on the network, during which a token from the client is passed to the server, and a token is returned from the server to the client in the mutual authentication case.

Once established, the security context is used in subsequent requests (provided that the configuration does not require security context deletion after every method invocation). If the same security context is reused, the only additional overhead considerations are (1) the signing and verification of requests and responses in the client and server, and (2) the security context handle (32 additional bytes of information) appended to each message passed between the client and the server.

The signing and verification of a signature on a request or response is different from the verification of the privileges used when the security context is first set up, in that verification of a signature does not require a network round-trip. In contrast, when you first set up a security context, a network round-trip to the privilege server is required, and its overhead is significantly more costly than that of the verification and signature operations.

Note that when a client has multiple object references to a single method implementation in a server, a single security context can still be used. For example, a derived object reference does not require a new security context. This is both an optimization and a functional requirement, since only one security context is allowed between a client process and a server implementation.

Future Work

The OMG specifies a number of object services in addition to the CORBA specification itself. One of the most important specifications is for the CORBA Security Service. ObjectBroker's integration with DCE Security was designed and implemented before the OMG's CORBA Security Service specification was complete. Thus, even though ObjectBroker is the most secure ORB available today, it is reasonable to ask when and how its security features will be made compliant with the latest specifications from the OMG.

Given sufficient resources, ObjectBroker engineering could investigate supporting CORBA2 interoperability by implementing the OMG's General Inter-ORB Protocol (GIOP). The GIOP architecture supports both the Internet Inter-ORB Protocol (IIOP) and the DCE-based Common Inter-ORB Protocol (DCE-CIOP). Today, ObjectBroker uses a wire protocol based on the CORBA version 1.2 specification.

Security for the IIOP is governed by the Secure Inter-ORB Protocol (SECIOP) specification[10], although few commercially available implementations of the SECIOP are available at the time of this writing. Also, as mentioned previously, security for the DCE-CIOP is accomplished by protecting the RPC connections at the wire protocol level. For the DCE RPC, the DCE does its own authentication for RPC sessions; here the RPC connection between the client and the server, rather than the client and the server themselves, is authenticated. This approach provides the same potential for security management in the ORB configuration; it simply accomplishes the security functions at a level in the protocol stack that does not require the use of the GSS-API. By building in support for the GIOP, ObjectBroker gains the capability to provide the security features for both the IIOP and the DCE-CIOP protocols in future releases.

The SECIOP and the DCE-CIOP both follow the usage model of minimizing the need for applications to be aware of security. In the SECIOP, the OMG has specified APIs for security functions, and these functions are entirely separate from any mechanism that implements them. ORB vendors will be free to provide security features in much the same way that ObjectBroker provides security today, i.e., by working from security-related information kept by the ORB. The SECIOP also provides for administrative objects and operations that perform security management functions by means of APIs.

Conclusion

ObjectBroker provides state-of-the-art distributed system security today. Its security features provide upward compatibility, as well as the least possible disturbance to existing ObjectBroker applications and configurations. In addition, ObjectBroker's implementation of security by means of the DCE's Generic Security Service Application Programming Interface provides the greatest possible choice among security mechanisms and security implementation providers.

References and Notes

1. R. Otte, P. Patrick, and M. Roy, *Understanding CORBA* (Upper Saddle River, N.J.: Prentice Hall, 1996).

2. Information about the Object Management Group is available at http://www.omg.org.

3. J. Linn, *Generic Security Service Application Program Interface,* Internet RFC 1508, 1993.

4. J. Wray, *Generic Security Service API: Overview and C-bindings,* Internet RFC 1509, 1993.

5. Information about The Open Group is available at http://www.opengroup.org.

6. M. Gasser, *Building a Secure Computer System* (New York, N.Y.: Van Nostrand Reinhold, 1988).

7. *X/Open DCE: Authentication and Security Services,* X/Open Preliminary Specification P315, ISBN 1-85912-013-X, electronic version (Reading, U.K.: X/Open Company Limited, 1995).

8. *ObjectBroker—Designing and Building Applications,* Part No. AA-QX1LA-TK (Maynard, Mass.: Digital Equipment Corporation, 1996).

9. S. Miller, B. Neuman, J. Schiller, and J. Saltzer, *Kerberos Authentication and Authorization System* (Cambridge, Mass.: Massachusetts Institute of Technology, Project Athena, 1987).

10. *CORBA Security,* Document Number 95-12-01 (Framingham, Mass.: Object Management Group, 1995). The OMG members who contributed to the document were AT&T Global Information Solutions Co., Digital Equipment Corporation, Expersoft Corporation, Groupe Bull, Hewlett-Packard Company, International Business Machines Corporation (in collaboration with Taligent Inc.), International Computers Limited, Novell Inc., Siemens Nixdorf Informationssysteme AG, Sunsoft Inc., Tandem Computer Incorporated (in collaboration with Odyssey Research Associates Inc.), and Tivoli Systems Inc.

Biographies

John H. Parodi
John Parodi is a consulting technical writer in the Multiplatform Engineering group. His primary work involves customer communications and evangelism for object technology. In earlier work, John provided technical writing support for the Compound Document Architecture group and Architectural Engineering of Systems and Software Technical Office. John joined DIGITAL in 1979 after working in computer operations at Hendrix Electronics and at the University of New Hampshire. He has received two awards from the Society for Technical Communication and has more than 30 publications on various computer science topics, including compound documents, object technology, computer security, and BASIC.

Fred W. Burgher
Principal engineer Fred Burgher is employed by BEA Systems as a member of the ObjectBroker Engineering team. He is currently involved in ObjectBroker IIOP development. Previously, Fred worked at DIGITAL on integrating DCE Security and Naming for the OpenVMS operating system. Earlier in his career, he was employed as a principal engineer at Wang Laboratories, where he worked in the Imaging Engineering Group. Fred studied computer science at Boston University.