SYBASE®

Security Administration and Programming Guide

EAServer

6.0

DOCUMENT ID: DC38035-01-0600-01

LAST REVISED: July 2006

Contents

About This Book

This book describes the features in EAServer with which you can define the security characteristics of client/server communications.

Audience

Use this document if you are responsible for creating or deploying secure components, applications, and Web applications, or for defining secure EAServer listeners with which clients communicate.

How to use this book

Use this document to understand EAServer security.

- Chapter 1, "Security Concepts" – an overview of security terms and concepts and describes how to protect server resources.
- Chapter 2, "Securing Component Access" – how to authenticate base clients, other components, or servlets and JSPs. Also describes how to pass credentials from EJBs and servlets between servers.
- Chapter 3, "Using Web Application Security" – how to secure Web applications and the resources contained within Web applications.
- Chapter 4, "Using SSL in Java Clients" – describes how to use SSL in Java clients.
- Chapter 5, "Using SSL in C++ Clients" – describes how to use SSL in C++ clients.
- Chapter 6, "Using TLS and FIPS" – how to use TLS and FIPS protocols to create secure EAServer connections.
- Chapter 7, "Creating and Using Custom Security Components" – how to create and implement custom role and service components to meet your specific authentication and authorization needs.
- Chapter 8, "Using the JAAS API" – how to implement the Java Authentication and Authorization Support (JAAS) module in clients, EAServer, and as connectors to other servers.
- Chapter 9, "Deploying Applications Around Proxies and Firewalls" – how to deploy applications around firewalls and how to use reverse proxies.

- Chapter 10, "Security Configuration Tasks" – the major security tasks you perform from the Web Management Console's EAServer Manager plug-in, including:
    - Role mapping
    - OS-based authentication
    - Defining security profiles that use SSL
    - Assigning security profiles to EAServer listeners
- Chapter 11, "Managing Keys and Certificates" – how to manage all aspects of SSL keys and certificates.

Related documents

Core EAServer documentation The core EAServer documents are available in HTML and PDF format in your EAServer software installation and on the SyBooks™ CD.

What's New in EAServer 6.0 summarizes new functionality in this version.

The *EAServer API Reference Manual* contains reference pages for proprietary EAServer Java classes and C routines.

The *EAServer Automated Configuration Guide* explains how to use Ant-based configuration scripts to:

- Define and configure entities, such as EJB modules, Web applications, data sources, and servers
- Perform administrative and deployment tasks

The *EAServer CORBA Components Guide* explains how to:

- Create, deploy, and configure CORBA and PowerBuilder™ components and component-based applications
- Use the industry-standard CORBA and Java APIs supported by EAServer

The *EAServer Enterprise JavaBeans User's Guide* describes how to:

- Configure and deploy EJB modules
- Develop EJB clients, and create and configure EJB providers
- Create and configure applications clients
- Run the EJB tutorial

The *EAServer Feature Guide* explains application server concepts and architecture, such as supported component models, network protocols, server-managed transactions, and Web applications.

The *EAServer Java Message Service User's Guide* describes how to create Java Message Service (JMS) clients and components to send, publish, and receive JMS messages.

The *EAServer Migration Guide* contains information about migrating EAServer 5.*x* resources and entities to an EAServer 6.0 installation.

The *EAServer Performance and Tuning Guide* describes how to tune your server and application settings for best performance.

The *EAServer System Administration Guide* explains how to:

- Start the preconfigured server and manage it with the Sybase Management Console
- Create, configure, and start new application servers
- Define database types and data sources
- Create clusters of application servers to host load-balanced and highly available components and Web applications
- Monitor servers and application components
- Automate administration and monitoring tasks with command line tools

The *EAServer Web Application Programming Guide* explains how to create, deploy, and configure Web applications, Java servlets, and JavaServer Pages.

The *EAServer Web Services Toolkit User's Guide* describes Web services support in EAServer, including:

- Support for standard Web services protocols such as Simple Object Access Protocol (SOAP), Web Services Description Language (WSDL), and Uniform Description, Discovery, and Integration (UDDI)
- Administration tools for deployment and creation of new Web services, WSDL document creation, UDDI registration, and SOAP management

The *EAServer Troubleshooting Guide* describes procedures for troubleshooting problems that EAServer users may encounter. This document is available only online; see the EAServer Troubleshooting Guide at http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.help.eas_5.2.eastg/html/eastg/title.htm.

jConnect for JDBC documents EAServer includes the jConnect™ for JDBC™ 6.0.5 driver to allow JDBC access to Sybase database servers and gateways. The *jConnect for JDBC 6.0.5 Programmer's Reference* is available on the Sybase Product Manuals Web site at http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.help.jconnjdbc_6.05.prjdbc/html/prjdbc/title.htm&toc=/com.sybase.help.jconnjdbc_6.05/toc.xml.

Sybase Software Asset Management User's Guide EAServer includes the Sybase Software Asset Management license manager for managing and tracking your Sybase software license deployments. The *Sybase Software Asset Management User's Guide* is available on the Getting Started CD and in the EAServer 6.0 collection on the Sybase Product Manuals Web site at http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.help.eas_6.0/title.htm.

Conventions

The formatting conventions used in this manual are:

Formatting example	To indicate
commands and methods	When used in descriptive text, this font indicates keywords such as: • Command names used in descriptive text • C++ and Java method or class names used in descriptive text • Java package names used in descriptive text • Property names in the raw format, as when using jagtool to configure applications rather than the Web Management Console
variable, *package*, or *component*	Italic font indicates: • Program variables, such as *myCounter* • Parts of input text that must be substituted, for example: `Server.log` • File names • Names of components, EAServer packages, and other entities that are registered in the EAServer naming service
File | Save	Menu names and menu items are displayed in plain text. The vertical bar shows you how to navigate menu selections. For example, File | Save indicates "select Save from the File menu."
`package 1`	Monospace font indicates: • Information that you enter in the Web Management Console, a command line, or as program text • Example program fragments • Example output fragments

Other sources of information

Use the Sybase Getting Started CD, the SyBooks CD, and the Sybase Product Manuals Web site to learn more about your product:

- The Getting Started CD contains release bulletins and installation guides in PDF format, and may also contain other documents or updated information not included on the SyBooks CD. It is included with your software. To read or print documents on the Getting Started CD, you need Adobe Acrobat Reader, which you can download at no charge from the Adobe Web site using a link provided on the CD.
- The SyBooks CD contains product manuals and is included with your software. The Eclipse-based SyBooks browser allows you to access the manuals in an easy-to-use, HTML-based format.

  Some documentation may be provided in PDF format, which you can access through the PDF directory on the SyBooks CD. To read or print the PDF files, you need Adobe Acrobat Reader.

  Refer to the *SyBooks Installation Guide* on the Getting Started CD, or the *README.txt* file on the SyBooks CD for instructions on installing and starting SyBooks.
- The Sybase Product Manuals Web site is an online version of the SyBooks CD that you can access using a standard Web browser. In addition to product manuals, you will find links to EBFs/Maintenance, Technical Documents, Case Management, Solved Cases, newsgroups, and the Sybase Developer Network.

  To access the Sybase Product Manuals Web site, go to Product Manuals at http://sybooks.sybase.com/nav/base.do.

Sybase certifications on the Web

Technical documentation at the Sybase Web site is updated frequently.

❖ Finding the latest information on product certifications

1. Point your Web browser to Technical Documents at http://www.sybase.com/support/techdocs/.
2. Select Products from the navigation bar on the left.
3. Select a product name from the product list and click Go.
4. Select the Certification Report filter, specify a time frame, and click Go.
5. Click a Certification Report title to display the report.

❖ Creating a personalized view of the Sybase Web site (including support pages)

Set up a MySybase profile. MySybase is a free service that allows you to create a personalized view of Sybase Web pages.

1 Point your Web browser to Technical Documents at http://www.sybase.com/support/techdocs/.

2 Click MySybase and create a MySybase profile.

Sybase EBFs and software maintenance

❖ Finding the latest information on EBFs and software maintenance

1 Point your Web browser to the Sybase Support Page at http://www.sybase.com/support.

2 Select EBFs/Maintenance. If prompted, enter your MySybase user name and password.

3 Select a product.

4 Specify a time frame and click Go. A list of EBF/Maintenance releases is displayed.

   Padlock icons indicate that you do not have download authorization for certain EBF/Maintenance releases because you are not registered as a Technical Support Contact. If you have not registered, but have valid information provided by your Sybase representative or through your support contract, click Edit Roles to add the “Technical Support Contact” role to your MySybase profile.

5 Click the Info icon to display the EBF/Maintenance report, or click the product description to download the software.

Accessibility features

EAServer has been tested for compliance with U.S. government Section 508 Accessibility requirements. The online help for this product is also provided in Eclipse help formats, which you can navigate using a screen reader.

The Web Management Console supports working without a mouse. For more information, see “Keyboard navigation” in Chapter 2, “Management Console Overview,” in the *EAServer System Administration Guide*.

The Web Services Toolkit plug-in for Eclipse supports accessibility features for those that cannot use a mouse, are visually impaired, or have other special needs. For information about these features see the Eclipse help:

1 Start Eclipse.

2 Select Help | Help Contents.

3 Enter `Accessibility` in the Search dialog box.

4 Select Accessible User Interfaces or Accessibility Features for Eclipse.

Note You may need to configure your accessibility tool for optimal use. Some screen readers pronounce text based on its case; for example, they pronounce ALL UPPERCASE TEXT as initials, and MixedCase Text as words. You might find it helpful to configure your tool to announce syntax conventions. Consult the documentation for your tool.

For additional information about how Sybase supports accessibility, see Sybase Accessibility at http://www.sybase.com/accessibility. The Sybase Accessibility site includes links to information on Section 508 and W3C standards.

If you need help

Each Sybase installation that has purchased a support contract has one or more designated people who are authorized to contact Sybase Technical Support. If you cannot resolve a problem using the manuals or online help, please have the designated person contact Sybase Technical Support or the Sybase subsidiary in your area.

CHAPTER 1 Security Concepts

Keeping resources secure is an ongoing challenge. As defenses are implemented, new methods are devised to circumvent them. The following is a short list of the many excellent sites that contain security related information:

- A good source of cryptographic related information is available from RSA laboratories at ftp://ftp.rsasecurity.com/pub/labsfaq/labsfaq4.pdf.
- The Java security page at http://java.sun.com/security/

Authentication and authorization

Authentication means that a person, client, or server has been verified to either a server or a client. In contrast, authorization means that a person, client, or server has permission to use a resource or file. An entity must be authenticated before it can be authorized to use a resource or file. This book describes authentication and authorization services provided for:

- Components and packages
- Web clients
- Java, C++, and PB clients
- Web applications
- Client/application server connections

Public-key cryptography

To maintain secure communications between a client and host, public-key cryptography techniques are used for:

- Authentication – verifying the identity of both the client and the server; Public-key cryptography techniques use digitally signed certificates that identify network entities.
- Encryption – modifying data so that it can be read only by the party for whom it is intended. When used with a user's private key, certificates encrypt and decrypt messages.

Unencrypted messages are known as *plain text*. Encoding the contents of a message is called *encryption*. This encrypted message is the *cipher text*. *Decryption* is the process of retrieving the plain text from the cipher text. A key is usually required to perform encryption and decryption. A *cipher suite* defines the parameters and methods supported by both the client and server that perform the encryption and decryption.

Public-key encryption uses a pair of keys for encryption and decryption. One key is secret (the private key) and the other is distributed (the public key). You send your digitally signed public key (certificate) to anyone you want to communicate with.

Messages that are sent to you are encrypted with your distributed public key and decrypted by your private key, while messages sent by you are encrypted with your private key and decrypted with your distributed public key. *RSA encryption* is a widely used public-key encryption system.

For more information on RSA and public-key encryption, see the RSA Web site at http://www.rsa.com .

Public-key certificates

Public-key certificates provide a method to identify and authenticate clients and servers on the Internet. Public-key certificates are administered and issued by a third party known as a *certification authority* (CA). A *subject* (individual, system, or other entity on the network) uses a program to generate a key pair and submits the public key to the CA along with identifying information (such as name, organization, e-mail address, and so on). This is known as a *certificate request*. The CA issues a digitally signed certificate. A *digital signature* is a block of data that is created using a private key.

The CA ties the certificate owner to the public key within the certificate. The subject then uses the certificate, along with his private key to establish his identity. Once this is done, whomever the subject is communicating with knows that a third party has vouched for his identity.

The process requires these steps:

1. Use the set-certificate script located in the *bin* subdirectory of your EAServer installation to map a user name to a certificate. These are the only client certificates that EAServer 6.0 trusts.
2. The client supplies its certificate and negotiates a secure connection with the server.

SSL, HTTPS, and IIOPS

SSL provides security for network connections. Specifically, SSL uses public-key encryption to provide:

- Client and server authentication using certificates
- Encryption, which prevents third parties from understanding transmitted data
- Integrity checking, which detects whether transmitted data has been altered

Packets for other protocols can be embedded inside of SSL packets. A connection in which the application protocol is embedded inside of SSL is an *SSL-tunnelled* connection.

Both IIOP and HTTP can be tunnelled inside SSL, which means that these protocols take advantage of SSL security features. For example, HTTPS connections embed HTTP packets inside of SSL packets. Your Web browser creates a secure HTTP connection any time you load a page from a URL that begins with “https:”

See Chapter 11, “Managing Keys and Certificates”

TLS and FIPS

TLS (Transport Layer Security) is a protocol from the IETF based on SSL and provides similar services as SSL. FIPS (Federal Information Processing Standard) are the standards and guidelines for information processing developed by NIST and approved by the Secretary of Commerce as requirements for the Federal Government for information assurance and interoperability.

See Chapter 6, “Using TLS and FIPS,” for complete information.

Proxies and firewalls

A *firewall* is a system that enforces an access control policy between networks. Located on a gateway into the network, the firewall blocks traffic that does not have permission to access the network. An organization establishes a firewall so that it can control access to resources. For example, an organization that allows intranet users access to the Internet installs a firewall to prevent external users from accessing internal resources.

Proxy servers are typically used to constrain and secure connections from an organization’s computers to sites that require connecting across the Internet. To enhance security, some network configurations require all Internet connections to go through a proxy server, including IIOP connections to an application server.

See Chapter 9, “Deploying Applications Around Proxies and Firewalls” for more information.

Lines of defense

This section describes types of attacks and some strategies for defending against them.

Types of attacks

There are several ways in which data can be tampered with, compromised, and stolen. In addition, systems can be overwhelmed with traffic to the point that they are rendered useless.

Integrity attacks Data integrity is a measure of the quality of the information stored and transmitted on a system. Types of attacks on data integrity include deleting or modifying files or information on the file system or over a network.

Spoofing IP spoofing occurs when an intruder attempts to deceive the target system into accepting packets that appear to the target as coming from someone other than the intruder. If the target system already has an authenticated TCP session with another system and mistakenly accepts spoofed IP packets, the intruder can access sensitive information and lead the target to execute commands in that packet, as though they came from the authenticated connection.

Availability attacks Availability attacks occur when a resource such as a Web site or HTTP port becomes unavailable due to a high volume of traffic. Someone can use a program to generate thousands of simultaneous requests aimed at the same site, which then is unable to respond to legitimate requests.

Capture-and-replay Capture-and-replay refers to an intruder capturing data as it moves from one system to another. User names, passwords, authentication information, and so on, can be tampered with or used by the intruder to gain access to protected resources.

There are a variety of ways and tools that intruders use to gain access to system resources. Some of these attacks may go undetected, while others destroy or alter information. Following is a few examples of how an intruder gains access to system resources:

- A *brute force* attack involves using many combinations until the right key/password is located.
- A *trojan horse* attack occurs when an intruder secretly inserts a program or file that either steals or destroys information, such as a virus. Another simple example would be for someone to place a bogus program on your system that prompts for a user name and password. The program simply logs the user name and password information. The intruder accesses this information and can then use your user name and password to access resources to which you are permitted.
- A *person-in-the-middle* attack intercepts communication between two parties without their knowledge.

Defense against attacks

This section discusses some of the methods by which you can protect data and restrict access to resources.

Protecting ports and listeners You can provide various levels of security to EAServer listeners by assigning security profiles to HTTPS and IIOPS listeners. See Chapter 10, “Security Configuration Tasks” for more information.

Protecting application server resources and securing clients EAServer provides several methods to protect server resources and secure client/server connections:

- Set authentication and authorization levels using the Web Management Console. See Chapter 3, “Using Web Application Security.”
- Create custom authentication and authorization components. See Chapter 7, “Creating and Using Custom Security Components.”
- Use the Java authentication and authorization service (JAAS). See Chapter 8, “Using the JAAS API.”
- Establish minimum levels of protection for listeners using quality of protection (QOP). See Chapter 2, “Securing Component Access.”
- Propagate client principal information from one server to another and use run-as support so an EJB can perform method invocations on other EJBs using a different identity. See Chapter 2, “Securing Component Access.”

Protecting data Use public-key certificates when exchanging sensitive data over a network to protect it from being viewed by intruders. See Chapter 11, “Managing Keys and Certificates” for more information.

CHAPTER 2 Securing Component Access

Components can be invoked by clients, other components, or servlets and JSPs. This chapter describes the various methods used to authenticate and authorize each type of client.

Topic	Page
Client authentication	7
Intercomponent authentication	9
Intercomponent authentication for EJBs and servlets	12
Client authorization	15

Client authentication

Users are authenticated when a client application creates a proxy or stub object (a connection is made when the application creates the first proxy or stub; other proxies or stubs may use the same connections or allocate new connections as needed).

Authentication options for base clients include:

- **Authentication methods** There are nine in total; cts-auth, ftp, http, jaas, jndi, local-hash (the default), none, and os-auth. See “Login methods” on page 81 of Chapter 10, “Security Configuration Tasks” for a description of each.

- **SSL certificate-based authentication** Alternatively, you can map a client certificate to a user using the *bin/set-certificate* script located in the EAServer directory. Then, if the client connects to an IIOPS or HTTPS listener configured for mutual authentication (for Web applications, the Web application must also be configured for CLIENT-CERT authentication) the certificate is mapped to a user name. In the IIOP case, if you use a certificate and provide a password, the password must also be correct. However you do not need to provide a password.

  Note Using "keytool" to add client certificates is not sufficient, only the *bin/set-certificate* script configures the necessary certificate to user mapping (which is stored in the "default" Security Domain properties file). See Chapter 11, "Managing Keys and Certificates" for information about using keytool commands and set-certificate.

  You can configure a secure IIOP port that requires mutual (client and server) authentication. Clients must have a valid SSL certificate to connect to this port, and the certificate must be issued by a certificate authority that is trusted by EAServer.

  When clients connect with an SSL client-side certificate, the client also supplies an EAServer user name and password for the connection in addition to the certificate. EAServer performs authorization checking based on the EAServer user name. The SSL user name and certificate information are available through the built-in CtsSecurity/SessionInfo component.

  EAServer provides native SSL support without the use of proxies. On the client side, the EAServer Java ORB supports SSL. Java applets and Java applications, C++, and PowerBuilder can use SSL natively. Other types of clients require the use of an SSL proxy.

  C++ and PowerBuilder™ clients require that a public-key infrastructure (PKI) system be available on the client to manage those certificates. Use sctool.exe to manage the certificate database.

  See "Configuring security profiles" on page 90 for information about the various authentication levels you can establish for a client-EAServer connection.

- **Custom authentication** You can code a service component to perform your own authentication checks. For example, you can retrieve the client user name and password and check to see if they allow a login to a remote database server. See "User installed authentication services" on page 9 for more information.

EAServer provides a special user name, admin@system, for the EAServer Administrator login. Administrator authentication is performed independently of the authentication option you configure. By default, the admin@system user name has no password. Use the set-admin-password batch file located in the EAServer *bin* directory to establish the admin@system password.

Set the administrator password for new servers Immediately after you create a new server, you must secure access to the server by defining the admin@system password and configuring the authentication mechanism of your choice. See "Administration password and OS authentication" in the *EAServer System Administration Guide* for more information.

For backwards compatibility, you can use "jagadmin" as an alias for "admin@system". However by default you cannot have an empty password. To allow an empty password (which is not recommended) you must set the "minimum password length" property of the "default" Security Domain to "0".

By default, the admin@system user name has no password. Use the set-admin-password batch file located in the EAServer *bin* subdirectory to establish the admin@system password, otherwise the server will not start.

The "admin@system" password must be the same for all members of a cluster.

User installed authentication services You can install your own service component to authenticate clients for any EAServer. For example, if you require the client user name to match a remote database user name, you can code the component to retrieve the client user name and password and attempt to log in to the remote database. For more information, see Chapter 7, "Creating and Using Custom Security Components."

Intercomponent authentication

This section describes various security features that are available ton components, including:

- Retrieving SSL information
- Restricting access to EJBs
- Authenticating non-EJB components within a server and for standalone clients
- Issuing intercomponent calls using SSL

Accessing SSL information

Clients can connect to a secure IIOP port using an SSL client certificate. You can issue intercomponent calls to the built-in *CtsSecurity/SessionInfo* component to retrieve the client certificate data, including the:

- Distinguished SSL user name
- Client certificate fingerprint (MD5 message digest)
- Client certificate data
- Chain of issuing certificates

This component implements the CtsSecurity::SessionInfo IDL interface. HTML documentation is available for the interface in the *html/ir* subdirectory of your EAServer installation. View it by loading the main EAServer HTML page, then clicking the “Interface Repository” link.

CtsSecurity::UserCredentials interface has been replaced The CtsSecurity::UserCredentials interface, which is implemented by the *CtsSecurity/UserCredentials* component, has been replaced by the CtsSecurity::SessionInfo interface, which provides additional functionality such as certificate parsing. EAServer supports the CtsSecurity::UserCredentials interface for backward compatibility. Use the CtsSecurity::SessionInfo interface if you are developing new components.

Non-EJB components

For non-EJB CORBA components, use the following mechanism for authentication within a server and for standalone clients:

Embed the user name and password in the URL when creating a component instance. For example:

```
Module::Interface_var compInstance = Module::Interface::narrow(
"iiop[s]://user:password:host:port/EAServerPackage/EAServerComponent");
```

Preferred API usage patterns

The preferred access pattern for CORBA clients (using Java example code) is:

```
Manager manager = ManagerHelper.narrow(orb.string_to_object(url));
Session session = manager.createSession(username, password);
Factory factory =
FactoryHelper.narrow(session.lookup("MyPackage/MyComp"));
MyComp comp = MyCompHelper.narrow(factory.create());
```

The last two lines can alternatively be replaced with:

```
MyCompHome home =
MyCompHomeHelper.narrow(session.loookup("MyPackage/MyComp"));
MyComp come = home.create();
```

The second style is recommended because it allows you to pass one or more parameters to the called EJB stateful session's home.create(...) method. This is why the factory.create(username, password) API was deprecated, since it conflicts with the home.create methods for stateful session beans with initialization parameters.

CosNaming APIs

Explicit use of CosNaming APIs in CORBA clients is not recommended, because it might lead you to use the deprecated (or in the non-Java case, unsupported) factory.create(username, password) API.

For inter-component calls, you can use the shortcut:

```
MyComp comp =
MyCompHelper.narrow(orb.string_to_object("MyPackage/MyComp"))
```

Or you can use an URL of "iiop://0:0" with the full style using explicit Manager, Session, and so on.

PowerBuilder clients and components

PowerBuilder clients can use the "Connection.CreateInstance" method, and can also use the explicit CORBA style shown above.

PowerBuilder components doing intercomponent calls can use the "TransactionServer.CreateInstance" method, and can also use the explicit CORBA style shown above.

C++ components

C++ components (and PowerBuilder NVOs) can make intercomponent calls across different servers using SSL in much the same way as any other C++ client. However, be aware that the SSLServiceProvider interface is not available to components. Instead, set ORB-level SSL properties to initiate server-to-server intercomponent calls using SSL.

For information about developing C++ components and clients, see these chapters in the *EAServer CORBA Component's Guide*:

- Chapter 8, "Developing CORBA/C++ Components"
- Chapter 9, "Developing CORBA/C++ Clients"

Intercomponent authentication for EJBs and servlets

EAServer implements J2EE version 1.3 security requirements, including Java and C++ ORB support and CORBA Secure Interoperable version 2 protocol (CSIv2). CSIv2 is part of EJB version 2.0 interoperability requirements, and supports:

- EJB 2.0 security features including:
    - Caller propagation on remote servers from EJB 2.0 clients using RMI/IIOP
    - Run-as support
    - Trust identities
- Servlet 2.3 security enhancements including:
    - Caller propagation on remote servers
    - Run-as support
- Java Authentication and Authorization Service (JAAS) – see Chapter 8, “Using the JAAS API” for more information.

Other references

For more information about J2EE, see the Java Web site at http://java.sun.com/j2ee.

For more information about servlet technology, see the Java Web site at http://java.sun.com/products/servlet/index.html.

For more information about CSIv2, see the OMG Web site at http://www.omg.org/technology/documents/formal/omg_security.htm

Intercomponent authentication for EJB 2.0 components

EJB 2.0 components use caller propagation to pass client information between servers for authentication, whereas run-as support allows EJB 2.0 components to perform method invocations on other components using a different identity.

Caller propagation

Caller propagation allows an EJB 2.0 RMI/IIOP client to pass principal information to a server and have that information propagated to other servers. In other words, EAServer can pass a client's user name or X.509 certificate information from an EJB on one server, to an EJB on a different server. For example:

1. The client passes principal information to EAServer1, where the information is authenticated.
2. EAServer1 retrieves the remote client's authentication information by calling getCallerPrincipal().
3. EJBA, on EAServer1, makes a call to another bean, which resides on EAServer2.
4. The propagated caller information is retrieved on EAServer2 using the getCallerPrincipal() method.

To enable caller propagation for EJB component calls made in servlet or component code when communicating with a third-party EJB server, specify a `corbaname` URL in the EJB Reference properties for the EJB component, servlet, or JSP that issues the call.

EAServer supports RMI/IIOP interoperability with other EAServer 6.0 servers, even when using "iiop:..." format URLs.

Caller principal propagation for inter-component calls (on the same server) is automatic (if you use ejb-refs or ejb-local-refs with no special configuration).

For information on interoperable naming URLs, see Chapter 5, "Interoperability," in the *EAServer EJB User's Guide*.

Run-as support

Normally, when a component calls another component, the invocation uses the client's credentials. You can use *identities* to specify alternate credentials for intercomponent calls. Identities map logical identity names to a user name, password, and required SSL session characteristics. The identity names are used in the run-as mode settings for components and component methods.

Run-as support enables an EJB 2.0 component to perform method invocations on other components using a specified identity. This identity can be configured at deployment time. In the standard EJB 2.0 deployment descriptor, the run-as property is expressed in terms of a role. The role is a name of a security-role element defined in the same deployment descriptor. It is expected that at deployment time, or when configuring a new EJB, the role name should be defined. Further, the deployer selects an identity that is expected to be present in this role. This identity is used while invoking another EJB. The run-as feature can be enabled in the Web Management Console.

To enable use of the run-as identity for EJB component calls made in component code, specify `corbaname` URLs in the EJB Reference properties for the EJB component that issues the call. For information on interoperable naming URLs, see Chapter 5, "Interoperability," in the *EAServer EJB User's Guide*.

❖ **Configuring an EJB 2.0 component to run as a different identity**

1 Modify the *config/ejbjar-name.xml* file (where *name* is the name of the EJB) to use <`run-as`>. If you do not set this property, intercomponent calls use the client identity. For example:

2 Recompile the component using the recompile batch file located in the EAServer *bin* subdirectory.

You can configure a run-as identity application-or server-wide. This provides a convenient way to globally set the run-as identity for all of the EJBs in an application or server.

Authentication of component invocation from servlets

This section describes how to propagate servlet credentials between servers and how to use identities to map logical identity names to a user name, password, and required SSL session characteristics. The identity names are used in the run-as mode settings on beans called from servlets.

Basically EAServer receives an HTTP request targeting a certain servlet which in turn invokes another EJB. The user credentials (either a user name/password pair, or a X.509 certificate) is passed to the invoked EJB. EAServer, which is the EJB container, authorizes the user's credentials.

Caller propagation for servlets on remote servers

A servlet's or JSP's HTTP client credentials are propagated when EJBs are invoked on remote servers. Earlier versions of EAServer propagated user name/password or digital IDs, and only within the same server.

Run-as support

Run-as support for servlets is similar to run-as support for EJBs:

- is defined on a per-servlet basis.
- applies to all method invocations on beans called from the servlet.

To enable run-as support for servlets or JSPs, the servlet or JSP must be installed in a Web application. Additionally, you must specify `corbaname` URLs in the EJB Reference properties for the servlet or JSP that issues the call. For information on interoperable naming URLs, see Chapter 5, "Interoperability," in the *EAServer EJB User's Guide*.

Configure the run-as identity as follows:

1. Define the run-as identity using the <run-as> property in the *config/webapp-name.xml* file. If you do not set this property, intercomponent calls use the client identity.
2. Recompile the servlet using the recompile batch file located in the EAServer *bin* subdirectory.

Client authorization

EAServer provides component authorization through both roles and custom components:

Roles EAServer authorization model is based on roles. Use the Sybase Management Console to define roles. Each role can include and exclude specific user names and digital certificates. If you use native operating system authentication, you can also include and exclude operating system group names; all users in the specified group are affected.

See "Configuring roles" on page 86 for more information on defining roles.

Custom components EAServer provides role and authorization service components with which you can create and install your own component to authorize clients to access resources on EAServer. See Chapter 7, “Creating and Using Custom Security Components.”

Enterprise JavaBeans

Your EJBs are ready for use after deployment, but if you want to modify access control properties, you have two options:

1. Modify the JAR or War file that contains the EJB and redeploy.
2. After deploying, modify the EJBs corresponding XML file located in the *config* subdirectory of your EAServer installation. For example, originally ejb.BeanRemote contains a method called foo() that is only accessible to roles “role1”, if you want to grant access to “role2”:

   a. locate the *ejbjar-ejb.BeanRemote.xml* file in the *config* subdirectory.

   b. Search for the line:

      ```
      <permitAccess method="foo()" roles="role1"/>
      ```

   c. In the section defining `BeanRemote interface`, modify the line to:

      ```
      <permitAccess method="foo()" roles="role1, role2"/>
      ```

   d. Go to the *bin* subdirectory of your EAServer installation and recompile the entity using the recompile command script, and refresh the module.

CHAPTER 3 Using Web Application Security

This chapter discusses how to establish authentication and authorization levels for your Web application elements using declarative security provided by the Sybase Management Console.

Introduction

A Web container holds your Web application elements, including components, servlets, JSPs, HTML pages, and so on. The Web application's deployment descriptor describes how a Web application is deployed, including the level of security for the various elements of your Web application. For example, your Web application may include an HTML page that is available to all visitors to your site, while other HTML pages, servlets, and JSPs are restricted to existing or preferred customers.

Web client security requires that Web content be deployed in Web applications:

- There is no way to secure files deployed in the EAServer HTML root directory.
- Do not put sensitive information such as passwords in files that can be downloaded by Web clients.
- Do not put files containing sensitive information in locations that allow download by Web clients.

Configuring security properties of your Web application

Define the authentication method of your Web application and security constraints on the various elements within your Web application and redeploy the Web application.

Refer to the user documentation of your specific development tool for information about setting Web application security. You can also refer to the Java Servlet 2.4 specification at http://java.coe.psu.ac.th/J2EE/Servlet2.4/servlet-2_4-fr-spec.pdf for additional information.

Authentication

The types of Web application authentication methods available include:

- **None** No authentication is required.
- **Basic** The server asks the client for its user name and password. You also provide a Realm name. The realm adds additional information to the client who is logging in to your site. For example, if you do not provide a realm name when a client visits your site, the browser displays a message to the client that states "The server at *host*:*port* wants you to log in." If you enter a realm name of "Human Resources Web site," the browser displays "The server at Human Resources Web site at *host*:*port* wants you to log in."

  When an HTTP client sends the HTTP basic authentication header:

  - The server authenticates the client using the server-defined authentication scheme and invokes any defined customized authentication component.
  - If the authentication fails, the request fails and an error message is sent back to the client. If the request is intended for a Web application, the Web application manages error handling.
  - If the request is intended for a regular static page, the request is denied, and HTTP status code 401 (Unauthorized) is sent back to the client.
- **Form** The Web application developer creates an HTML login page, where the client enters a user name and password. The entire HTML page is sent to the server. You also create an error page that is returned to the client in the event of a server error.

- Login page – enter the location of the login page that is supplied to the client at login. For example, */login.jsp* might be your login page.
- Error page – enter the location of the error page that the client is directed to should a server error occur during login. For example, */error.jsp* might be your error page.

Login and error pages can vary from a simple HTML page to a complex page that includes servlets and JSPs.

The location of the error and login pages is relative to the *WebApp* directory whether or not a "/" is used. For example, if you specify */error.jsp* or *error.jsp* as the location of your error page, the servlet engine assumes that it is contained in the *WebApp* context.

Below is an example of a form login and error page. The action of the form login page must always be j_security_check. The user name and password fields should be j_username and j_password respectively.

Form login page:

```
<html>
<body>
<h1>Login page</h1>

<form method="POST" action="j_security_check" >
<input type="text" name="j_username">
<input type="password" name="j_password">

<input type="submit" name="j_security_check">
</form>

</body>
</html>
```

Form error page:

```
<html>
<head>
<title>Login Error</title>
</head>
<body> Login error -- please try <a
href="login.html">again</a>.
</body>
</html>
```

These examples assume that *login.html* is the login page, and that the error page and login page are in the same directory.

- Client-cert – the client connects to the server using SSL tunneled within HTTP. The client must provide a certificate that the server accepts and authenticates. For more information about SSL, see Chapter 10, “Security Configuration Tasks,” and Chapter 11, “Managing Keys and Certificates.”

  Note You cannot use both “client-cert” and “OS authentication” as Web application security mechanisms at the same time. If you do, clients will not connect to the Web application. See “Configuring OS authentication” on page 89.

Note EAServer does not support HTTP digest authentication. If you specify digest authentication, the default, Basic, is used instead.

EAServer supports *lazy authentication*, which means that the server attempts to identify a client only when the client attempts to access a restricted resource. As long as the client accesses only resources that do not require authorization, the server does not attempt to authenticate the client.

When a server authenticates a client, the client is authenticated for all applications and references on the server. You can implement authentication of a client for an entire server by using cookies or rewriting the URL. A reference to the client’s security credentials is saved in a cookie or encoded in the URL.

Form login requirements in a Web application when using HTTPS (SSL)

To use the form login mechanism in your Web application, the client must support cookies. The client can be a browser or a standalone HTTP client. To convert your Web application, which uses the form login mechanism in conjunction with HTTPS, the transport guarantee for the form login page and the pages that require authorization must be identical. Otherwise, the client receives multiple HTTP redirects to the same page, resulting in an error. See “Defining a security constraint” on page 22 for information about configuring transport guarantee.

Here are the steps required to enable HTTPS for the eStore application, which is a large, comprehensive sample application developed by Sun Microsystems to run on J2EE-compliant servers. eStore simulates an online pet store implemented with JavaServer Pages, Java servlets, and Enterprise Java Beans. You can download eStore as part of the Sun Microsystems J2EE Blueprints at http://java.sun.com/j2ee/blueprints/.

1. Change the transport guarantee for the existing two security constraints from None to Confidentiality or Integrity.
2. Add a new security constraint. Set the transport guarantee for the new security constraint to the same value as the existing two security constraints.
3. Add a Web resource collection to the new security constraint. Define a Web resource, and set the URL pattern to "*/login.jsp*", which is the URL of the form login page.
4. Redeploy the eStore application. Connect to the eStore application from your browser. The form login and subsequent communication occurs using HTTPS.

Authorization

Security constraints enable you to set various levels of authorization within the elements of your Web application. You create J2EE roles and map them to EAServer roles, then limit access to JSPs, servlets, and HTML pages to entities that belong to an authorized J2EE role. In addition, you can define which HTTP methods have access to which URLs, and establish levels of transport guarantee.

For example, you could create a security constraint that blocks access to all users at the Web application level. You could then grant access to resources (HTML pages, JSPs, servlets) within the Web application to authorized users. To do this, you need at least two security constraints:

1. Create a top-level security constraint and assign to it a Web resource collection with a URL pattern set to "/*".

   Establish an authorized role for the security constraint that contains no users. For example, you could create the role of "None" and assign it to the security constraint.

2. Create another security constraint and assign to it a Web resource collection with a URL pattern set to the URL locations for which you are providing access.

   Establish an authorized role that contains the users that are allowed access to the Web resources protected by this security constraint.

3. Create additional security constraints and allow access to other Web resources as needed.

Use this same approach to define security constraints that require specific levels of transport guarantee.

❖ Defining a security constraint

1. Defining security constraints for your Web application includes:

   - A Web resource collection – a list of URL patterns and HTTP methods available for those URLs.

     The URL pattern can have two forms:

     - */url_name* – an individual URL.
     - */url_location/** – all of the URLs located in the *url_location* directory.

   - The HTTP operations that are allowed access to the defined URL patterns. HTTP operations include:

     - GET – the most common method used by browsers. GET receives its input through a query string.
     - POST – similar to a GET except that the input data is sent through standard input instead of using the query string. The POST method is normally used for an HTML form.
     - PUT – same as POST except PUT usually implies that the operation take effect immediately whereas POSTs action may be delayed.
     - OPTIONS – determines what HTTP options are supported.
     - DELETE – removes some entity.
     - TRACE – causes a response with a message containing all of the headers sent in the trace request.

   - Authorized roles – the authorized roles that have access to the HTTP methods for the URLs defined for this security constraint.

- Transport guarantee – establish a level of transport security for each security constraint appropriate for the Web resources you are protecting. If you use basic or form-based authentication, passwords and other sensitive information is not protected for confidentiality. If you have sensitive information that you want to protect, establish a security constraint that uses a greater level of protection. Supported transport guarantee levels are:
    - None – uses insecure HTTP. Using SSL-protected sessions has more overhead than insecure HTTP sessions. Use None for transport guarantee if you do not need the added confidentiality of SSL.
    - Integral – uses an SSL-protected session that checks for data integrity.
    - Confidential – uses an SSL-protected session to ensure that all message content, including the client authenticators, are protected for confidentiality as well as data integrity. A Confidential transport guarantee has more overhead than None.

2 Redeploy the Web application.

Sample web.xml configuration

```
<security-constraint>
    <web-resource-collection>
      <web-resource-name>securityZone</web-resource-name>
      <url-pattern>/*</url-pattern>
      <http-method>GET</http-method>
      <http-method>DELETE</http-method>
      <http-method>TRACE</http-method>
      <http-method>PUT</http-method>
      <http-method>POST</http-method>
      <http-method>OPTIONS</http-method>
    </web-resource-collection>
    <auth-constraint>
      <description>null</description>
      <role-name>admin-role</role-name>
      <role-name>Console_ReadOnly</role-name>
    </auth-constraint>
    <user-data-constraint>
      <description>userdata</description>
      <transport-guarantee>NONE</transport-guarantee>
    </user-data-constraint>
  </security-constraint>
```

CHAPTER 4

Using SSL in Java Clients

EAServer supports SSL connections from Java applets and applications. In deployment scenarios where clients connect to EAServer over the Internet, SSL can protect sensitive data transmitted over the network. For more information about SSL, see Chapter 10, “Security Configuration Tasks,” and Chapter 11, “Managing Keys and Certificates.”

Using SSL in Java applets

Java applet clients can use the Web browser’s Java SSL implementation to create SSL connections to an EAServer. SSL connections from a Java applet require that:

- You connect to a server listener that supports the desired level of security. You do this by specifying the address as an IOR string as described in “Creating a Manager instance” in Chapter 10, “Developing CORBA Java Clients,” of the *EAServer Corba Clients Guide*.
- Your Web browser recognizes and accepts the server listener’s SSL certificate.
- If using mutual authentication, you have a personal certificate installed in the Web browser’s certificate database, signed by a certificate authority that is recognized and trusted by the EAServer.

Note For more information about security, including managing client side certificates, see Chapter 11, “Managing Keys and Certificates.”

Using SSL in Java applications

Java application clients create SSL connections using the same native implementation used by C++ clients.

Requirements

Make sure you select the *Client Runtime* option when installing EAServer. SSL support in Java applications requires the files installed by these options.

The following environment variable settings are required at runtime:

- JAGUAR_CLIENT_ROOT must specify the full path to the EAServer client runtime installation directory.
- On Windows platforms, PATH must include the EAServer client *dll* subdirectory.
- On UNIX platforms, the system's shared library search path (LD_LIBRARY_PATH on Solaris) must include the EAServer client *lib* subdirectory.
- CLASSPATH must include the EAServer client runtime classes. Specify the full path to the *easclient.jar* and *easj2ee.jar* files, or include the classes in these JAR files in a JAR that you build yourself.

Establishing a secure session

To ensure a secure session between your Java application and EAServer, you must configure SSL settings before using one of the standard techniques to instantiate proxies for the EAServer components.

You can configure the settings required for SSL connections using two techniques:

1. **By setting ORB properties** The required SSL settings must be known in advance, and your application can connect only to servers that use certificates issued by a known, trusted certificate authority.

2. **By using the SSLServiceProvider interface** The SSLServiceProvider interface allows your application to determine what options are available at runtime. In addition, you can supply a callback class with methods that supply settings as needed and respond to exceptional cases. For example, the client ORB invokes callback methods if the application specified an invalid certificate password or if a connection is made to a server that uses certificate issued by an unknown certificate authority.

Applications that run without user interaction typically configure SSL settings with the ORB properties. Interactive applications typically use the SSLServiceProvider interface and install a callback. When a callback is installed, you can rely on user interaction in the callback methods to configure necessary settings. For example, if the certificate password has not been supplied, the ORB invokes the getPin callback method.

Once you have correctly configured the required SSL settings, use the standard technique to instantiate proxies, as described in Chapter 10, "Developing CORBA Java Clients," of the *EAServer Corba Components Guide*. Proxies are created in a secure session as long as the server supports the requested level of security.

Using the SSLServiceProvider interface

The CtsSecurity.SSLServiceProvider interface provides setGlobalProperty and getGlobalProperty methods to set and retrieve the SSL properties listed in Table 4-1 on page 34. After initializing an ORB instance, you can instantiate a proxy for the SSLServiceProvider interface with the ORB.resolve_initial_references method, as shown below:

```
import CtsSecurity.*;

SSLServiceProvider sslServProv =
       SSLServiceProviderHelper.narrow
       (orb.resolve_initial_references
       ("SSLServiceProvider"));
```

You can then call the setGlobalProperty method to set properties, as in the example below:

```
prov.setGlobalProperty("qop", "sybpks_intl");
```

Properties set with the SSLServiceProvider interface affect all ORB instances used by the application. However, if an equivalent property has been set for an ORB instance, the ORB property value takes precedence.

You can retrieve property values using the getGlobalProperty method. For example:

```
String availQop[] = prov.getGlobalProperty("availableQop");
String qopDesc[] = prov.getGlobalProperty("availableQopDesc");
```

getGlobalPropertyMethod returns an array of strings. When retrieving properties that take a single value, the value is returned in an array of length 1.

These methods are also documented in the generated Interface Repository documentation for the CtsSecurity::SSLCallback interface. The generated documentation is linked to your EAServer's main HTML page.

SSL properties

Refer to the Interface Repository Documentation at http://*hostname*:*portnumber*/ir/CtsSecurity__SSLSessionInfo.html#JaguarSecuritySSLProperties for a description of the ORB and SSLServiceProvider properties that govern the use of SSL, where *hostname* is the host on which EAServer is running, and *portnumber* is the HTTP listener number (8000 for example). In addition, you need to connect to a server address that can support your chosen level of security, as described in "Secure server addresses" on page 29.

Some properties, if not set or set incorrectly, cause the ORB to invoke an SSL callback method. If you do not install an SSL callback, the default callback implementation aborts the connection attempt.

Choosing a security characteristic

To use SSL, you must specify a value for the qop property in ORB properties or by using the SSLServiceProvider interface. Specify the name of an available security characteristic. The characteristic describes the CipherSuites the client uses when negotiating an SSL connection. When connecting, the client sends the list of CipherSuites that it uses to the server, and the server selects a cipher suite from that list. The server chooses the first cipher suite in the list that it can use. If the server cannot use any of the available CipherSuites, the connection fails.

"Configuring security profiles" on page 90 describes the security characteristics that are provided with EAServer. At runtime, you can retrieve a list of characteristics and their descriptions by retrieving the availableQop and availableQopDesc properties.

Set the qop property to `sybpks_none` to prevent any use of SSL on a connection. This setting can be useful if you have set the property globally for all ORBs using the SSLServiceProvider interface, and you want to override the setting for an individual ORB instance.

Secure server addresses

The client ORB connects only to a server listener that uses an equivalent or greater level of security as requested in the qop setting. If you use the CosNaming or JNDI interfaces to instantiate proxies, the name service URL cannot specify a server address that uses a higher level of security than specified by the qop property. For example, if your server uses the typical port configuration, you can specify port 9000 (no SSL) in the name service URL if the qop specifies mutual authentication. However, you cannot specify port 9002 (mutual authentication) in the name service URL and set the qop to request server-only authentication. When you use ORB.string_to_object to instantiate a SessionManager::Manager proxy, the listener specified by the server address must use a security profile that matches the client's qop setting.

For more information on instantiating proxies, see Chapter 10, "Developing CORBA Java Clients," of the *EAServer Corba Components Guide*.

Implementing an SSL callback

An SSL callback class must implement the CtsSecurity.SSLCallback interface (described in the Interface Repository Documentation at http://*hostname*:*portnumber*/ir/CtsSecurity__SSLCallback.html, where *hostname* is the host on which EAServer is running, and *portnumber* is the HTTP listener number (8000 for example). The ORB invokes callback methods when required SSL settings have not been configured or a setting has an incorrect value. To install the callback, call SSLServiceProvider.setGlobalProperty to set the callbackImpl property, as in the example below:

```
sslprov.setGlobalProperty("callbackImpl",
     "Sample.ClientSSL.SSLCallbackExample.SSLCallbackExampleImpl");
```

The SSLCallback method is trustVerify, which is called when the correct PIN for the certificate database has not been set, or if the server has presented a questionable certificate. The callback response determines whether the connection is allowed and, optionally, whether the certificate should be added to the local EAServer client certificate database.

You must implement this method in your class. If your implementation of a method does not process the request, throw an org.omg.CORBA.NO_IMPLEMENT exception so that the ORB uses the default response.

For more information about the callback method, see the documentation for the CtsSecurity::SSLCallback interface in the generated Interface Repository documentation.

Retrieving session security information

The CtsSecurity.SSLSession and CtsSecurity.SSLSessionInfo classes allow you to determine whether SSL is used on connections from a proxy to the server, and if so, retrieve the SSL session settings. The code below illustrates the sequence of calls:

```
... code to set ORB ssl properties, create session,
     instantiate proxy myComp ...
SSLSession sslSession =
  SSLSessionHelper.narrow(myComp);
try {
  SSLSessionInfo sslSessionInfo =
    sslSession.getSessionInfo();
} catch (CtsSecurity.SSLNotEnabledError e) {
     ... this means the proxy does not use SSL ...
}
```

You can call SSLSessionHelper.narrow to obtain the session information associated with any CORBA object.

The SSLSessionInfo methods allow you to determine the SSL session properties, such as the server's address, the client certificate in use, the server certificate in use, and so forth. For more information, see the Interface Repository documentation for the CtsSecurity::SSLSessionInfo interface.

Creating HTTP and HTTPS connections in Java applications

You can create HTTP connections in Java applications using the HTTP protocol handling code built in to the Java Developer's Kit, and HTTPS connections using the HTTPS protocol handler provided with EAServer.

HTTP connections

The standard Java virtual machine provides HTTP connectivity with these classes in java.net package:

- URL allows you to use Uniform Resource Locator strings for HTTP connections and other protocol connections that can be represented by URLs.
- URLConnection represents a connection to a server and resource indicated by a URL.
- HttpURLConnection extends URL with additional methods that are specific to the HTTP protocol.

For details on these classes, see the JDK documentation. The following code shows a typical example. This code opens a connection, retrieves the data (text is assumed), and prints it:

```
URL url = new URL("http://www.sybase.com/");
URLConnection conn = url.openConnection();
conn.connect();
InputStreamReader content
    = new InputStreamReader(conn.getInputStream());
for (int i=0; i != -1; i = content.read())
{
    System.out.print((char) i);
}
```

HTTPS connections

The procedure for creating HTTPS connections is similar to that for HTTP connections, except that you must install EAServer's HTTPS protocol handler in the Java virtual machine and configure SSL parameters before opening a connection.

System requirements EAServer's HTTPS protocol handler uses the same SSL implementation as used by Java and C++ IIOP clients and requires a full client runtime install. For information on system requirements, see "Requirements" on page 26.

Installing the HTTPS protocol handler

The EAServer HTTPS protocol handler can be installed two ways:

- By configuring the `java.protocol.handler.pkgs` Java system property, making it the default handler for all HTTPS URLs. This is the recommended approach if you do not need to use another vendor's HTTPS protocol handler in addition to the EAServer implementation.
- By calling one of the java.net.URL constructors that takes a java.net.URLStreamHandler as a parameter. This approach must be used if you must use more than one HTTPS protocol handler in one EAServer or in one client application.

Specifying protocol handlers at runtime

If you must use more than one HTTPS protocol handler in one EAServer or in one client application, you must call one of the java.net.URL constructors that takes a java.net.URLStreamHandler as a parameter. The specified java.net.URLStreamHandler instance overrides the default handler for the protocol specified by the URL. For example, to specify the EAServer HTTPS handler, use code like this:

```
import java.net.*;
import com.sybase.jaguar.net.JagURLStreamHandlerFactory;
import com.sybase.jaguar.net.HttpsURLConnection;

....

String url_string = "https://localhost:8081/index.html";

// The URL stream handler factory is required to create a stream
// handler.
JagURLStreamHandlerFactory fact = new JagURLStreamHandlerFactory();

// Extract the protocol from the front of the URL string
String protocol = url_string.substring(0, url_string.indexOf(":"));

// If the protocol is HTTPS, use the EAServer HTTPS handler. Otherwise,
// use the default handler
java.net.URL url;
if (protocol.equals("https"))
{
    url = new URL((URL)null, url_string,
        fact.createURLStreamHandler(protocol));
} else
{
    url = new URL(url_string);
}
```

❖ **Creating HTTPS connections**

1. Configure or install the EAServer HTTPS protocol handler as described in "Installing the HTTPS protocol handler" on page 31.
2. Create URL and URLConnection instances. If connecting to an EAServer, specify the address of an HTTPS listener that supports the desired level of security. For example:

   ```
   URL url = new URL("https://myhost:8081/index.html");
   URLConnection conn = url.openConnection();
   ```

3. Verify that the object returned by URL.openConnection is of class com.sybase.jaguar.net.HttpsURLConnection, then set SSL properties for the connection. "SSL properties" on page 34 describes the SSL properties that can be set. At a minimum, you must specify the qop and pin properties, as well as the certificateLabel property if using mutual authentication. For example:

   ```
   if (conn instanceof HttpsURLConnection)
   {
     HttpsURLConnection https_conn =
       (HttpsURLConnection) conn;
     try
     {
       https_conn.setSSLProperty( "qop","sybpks_intl"
   );
       https_conn.setSSLProperty( "pin", "secret");
       https_conn.setSSLProperty(
              "certificateLabel", "John Smith");
     }
     catch ( CtsSecurity.InvalidPropertyException ipe )
     {
       System.err.println( ipe );
     }
     catch ( CtsSecurity.InvalidValueException ive )
     {
       System.err.println( ive );
     }
   ```

4. Open the connection, for example:

   ```
   conn.connect();
   ```

Once the connection is open, you can perform any valid operation for a connection that uses java.net.HTTPUrlConnection. You can also call the getSessionInfo method to retrieve a CtsSecurity.SSLSessionInfo instance that allows you to verify the SSL connection parameters. For example:

```
java.net.URLConnection conn;
... deleted code that constructed URLConnection ...
if (conn instanceof HttpsURLConnection)
{
  HttpsURLConnection https_conn =
    (HttpsURLConnection) conn;
  CtsSecurity.SSLSessionInfo sessInfo =
    https_conn.getSessionInfo();
```

The SSLSessionInfo methods allow you to determine the SSL session properties, such as the server's address, the client certificate in use, the server certificate in use, and so forth. For more information, see the Interface Repository documentation for the CtsSecurity::SSLSessionInfo interface.

SSL properties

Table 4-1 lists the properties that can be set and retrieved with the HttpsURLConnection getSSLProperty, getGlobalProperty, setSSLProperty, and setGlobalProperty methods. Global properties are set and read with the getGlobalProperty and setGlobalProperty methods. Global properties affect all HTTPS connections, not just the HttpsUrlConnection instance on which they are set. The right column in Table 4-1 lists which methods are valid for each property.

Some properties, if not set or set incorrectly, cause the connection to invoke an SSL callback method. You can install a callback to respond to these cases with the callbackImpl global property. If you do not install an SSL callback, the default callback implementation aborts the connection attempt.

Table 4-1: HTTPS Properties

Property name	Description	Valid for methods
pin	Always required when using SSL. Specifies the PKCS #11 token PIN. This is required for logging in to a PKCS #11 token for client authentication and for retrieving trust information. This property cannot be retrieved. If not set, set to “any”, or set incorrectly, the connection invokes the getPin callback method.	setSSLProperty setGlobalProperty

Property name	Description	Valid for methods
certificateLabel	Required when using mutual authentication. Specifies the client certificate to use if the connection requires mutual authentication. The label is a simple name that identifies an X.509 certificate/private key in a PKCS #11 token. If the property is not set and the connection requires mutual authentication, the connection invokes the getCertificateLabel callback method, passing an array of available certificate names as an input parameter.	setSSLProperty getSSLProperty setGlobalProperty getGlobalProperty
qop	Always required when using SSL. Specifies the name of a security characteristic to use. See “Choosing a security characteristic” on page 36 for more information.	setSSLProperty getSSLProperty setGlobalProperty getGlobalProperty
userData	Specifies user data (String datatype). This is an optional property. Client code can set user data during connection initialization and access it using SSLSessionInfo::getProperty method in the SSL callback implementation. This may be useful as a mechanism to store connection-level context information that is otherwise not available through the SSLSessionInfo interface.	setSSLProperty getSSLProperty setGlobalProperty getGlobalProperty
callbackImpl	Specifies the name of a Java class that implements the CtsSecurity.SSLCallbackIntf interface. For example: `com.acme.AcmeSSLCallback` See “Implementing an SSL callback” on page 29 for more information.	setGlobalProperty getGlobalProperty
availableQop	Retrieve only. A list of available security characteristics. The qop property can be set only to values that appear in this list.	getGlobalProperty
availableQopDesc	Retrieve only. A list of descriptions for the available security characteristics, in the same order as listed in the value of the availableQop property.	getGlobalProperty

Choosing a security characteristic

To use SSL, you must specify the name of an available security characteristic as the value for the qop property. The characteristic describes the CipherSuites the client uses when negotiating an SSL connection. When connecting, the client sends the list of CipherSuites that it uses to the server, and the server selects a cipher suite from that list. The server chooses the first cipher suite in the list that it can use. If the server cannot use any of the available CipherSuites, the connection fails.

Chapter 10, "Security Configuration Tasks" describes the security characteristics that are provided with EAServer. At runtime, you can retrieve a list of characteristics and their descriptions by retrieving the availableQop and availableQopDesc properties.

Using Java Secure Socket Extension classes

The Java Secure Socket Extension (JSSE) is a set of Java packages that implements SSL and Transport Layer Security, which enables data encryption, server authentication, message integrity, and client authentication. JSSE is a client-side feature, which can be used with EAServer when it has been configured for SSL communication. For more information on SSL, see Chapter 11, "Managing Keys and Certificates.".

Note JSSE does not contain any actual cryptographic logic. You must obtain an API package that performs the cryptographic functions, such as Bouncy Castle or Cryptix, which are available free over the Internet.

❖ **Setting up your JSSE environment**

1. Download and install the JSSE according to the documentation on the Java Web page at http://java.sun.com/products/jsse. The basic steps are:
   - Copy the JSSE JAR files to the *jre/lib/ext* directory in your JDK installation.
   - Edit the *jre/lib/security/java.security* file in your JDK installation, and add this line:

```
security.provider.2=com.sun.net.ssl.internal.ssl.Provider
```

2. Download and install the Java Plug-in HTML Converter, either version 1.3.1 or 1.4.

If you install version 1.3.1:

a Download and install JSSE 1.0.2 in the JDK 1.3.1 *jre/lib/ext* subdirectory of the JDK installation.

b Set up *jre/lib/security/java.security* according to the JSSE 1.0.2 directions.

3 The JSSE Samples Web page at http://java.sun.com/j2se/1.4/docs/guide/security/jsse/samples/index.html includes samples that create clients using JSSE. Verify that the samples compile and run with your JDK. You must be able to use the Java samples to request the secure VeriSign Web page at https://www.verisign.com.

4 Using the Java keytool, import the *eas.crt* file; for example:

```
keytool -import -file eas.crt -keystore
DJC_HOME/_JDK13/jre/lib/security/
[cacerts | jssecacerts] -trustcacerts
```

To simplify things, use the default certificate store cacerts; the password is “changeit”.

5 To run a JSSE client application; for example, *ClientApp*:

a Create a *ClientApp.bat* file with these lines:

```
set CLASSPATH=%DJC_HOME%\lib\eas-client-14.jar;%CLASSPATH%
java -Djava.protocol.handler.pkgs=
   com.sun.net.ssl.internal.www.protocol ClientApp
```

If using JDK 1.5 rather than 1.4, change the reference to *eas-client-14.jar* to refer to *eas-client-15.jar*.

b Run *ClientApp.bat*.

If you do not have a Web proxy, remove the Web proxy settings from your client, and enter the server information; for example:

```
iiops://localhost:9001, or
iiops://<host_name>:9001
```

Note The following steps apply only to HTML applets.

6 Remove these client ORB properties from your HTML applet client, if appropriate:

- com.sybase.CORBA.WebProxyHost=localhost
- com.sybase.CORBA.WebProxyPort=80

- com.sybase.CORBA.LogFile=.\iiop.log

7 To access your Web page from a Web browser, enter:

```
http://<host_name>:8080/jssehtml/yourAppClient.html
```

Where *yourAppClient.html* is your HTML applet client.

8 In the applet, enter `iiops://<host_name>:9001` as the connection parameter, and click Connect.

Note Sybase recommends using a Web browser that supports the Java Plug-in 1.3.1 or higher and the Java Plug-in Converter 1.3.1 or higher

---

Configuring ORB settings

Direct IIOP connections using JSSE are not supported.

❖ Tunnelling IIOP through HTTPS (JSSE socket) using HTTP GET requests

IIOP is contained within the HTTP packets.

1 Set the client URL to iiops://<*host_name*>:9001.

2 Set the following client ORB properties. To enable the EAServer message service to access the ORB properties, set the properties using the JMS property names; otherwise, use the CORBA property names.

JMS property	CORBA property	Vale
org.omg.CORBA.ORBClass	org.omg.CORBA.ORBClass	com.sybase.CORBA.ORB
com.sybase.jms.https	com.sybase.CORBA.https	true
com.sybase.jms.useJSSE	com.sybase.CORBA.useJSSE	true
com.sybase.jms.forceSSL	com.sybase.CORBA.forceSSL	true

❖ Tunnelling IIOP through HTTPS (JSSE socket) using HTTP POST requests

IIOP is contained within the HTTP packets.

1 Set the client URL to iiops://<*host_name*>:9001

2 Set the following client ORB properties. To enable the EAServer message service to access the ORB properties, set the properties using the JMS property names; otherwise, use the CORBA property names.

JMS property	CORBA property	Vale
org.omg.CORBA.ORBClass	org.omg.CORBA.ORBClass	com.sybase.CORBA.ORB
com.sybase.jms.https	com.sybase.CORBA.https	true
com.sybase.jms.useJSSE	com.sybase.CORBA.useJSSE	true

JMS property	CORBA property	Vale
com.sybase.jms.forceSSL	com.sybase.CORBA.forceSSL	true
com.sybase.jms.HttpUsePost	com.sybase.CORBA.HttpUsePost	true

❖ **Tunnelling IIOP through an HTTPS connect (JSSE socket) using HTTP GET requests**

IIOP is contained within the HTTP packets.

1 Set the client URL to iiops://<*host_name*>:9001.

2 Set the following client ORB properties. To enable the EAServer message service to access the ORB properties, set the properties using the JMS property names; otherwise, use the CORBA property names.

JMS property	CORBA property	Vale
org.omg.CORBA.ORBClass	org.omg.CORBA.ORBClass	com.sybase.CORBA.ORB
com.sybase.jms.https	com.sybase.CORBA.https	true
com.sybase.jms.WebProxyHost	com.sybase.CORBA.WebProxyHost	<*web_proxy_host_name*>
com.sybase.jms.WebProxyPort	com.sybase.CORBA.WebProxyPort	<*web_proxy_port*>
com.sybase.jms.useJSSE	com.sybase.CORBA.useJSSE	true
com.sybase.jms.forceSSL	com.sybase.CORBA.forceSSL	true

❖ **Tunnelling IIOP through an HTTPS connect (JSSE socket) using HTTP POST requests**

IIOP is contained within the HTTP packets.

1 Set the client URL to `iiops://<`***`host_name`***`>:9001`.

2 Set the following client ORB properties. To enable the EAServer message service to access the ORB properties, set the properties using the JMS property names; otherwise, use the CORBA property names.

JMS property	CORBA property	Vale
org.omg.CORBA.ORBClass	org.omg.CORBA.ORBClass	com.sybase.CORBA.ORB
com.sybase.jms.https	com.sybase.CORBA.https	true
com.sybase.jms.HttpUsePost	com.sybase.CORBA.HttpUsePost	true
com.sybase.jms.WebProxyHost	com.sybase.CORBA.WebProxyHost	<*web_proxy_host_name*>
com.sybase.jms.WebProxyPort	com.sybase.CORBA.WebProxyPort	<*web_proxy_port*>
com.sybase.jms.useJSSE	com.sybase.CORBA.useJSSE	true
com.sybase.jms.forceSSL	com.sybase.CORBA.forceSSL	true

Note The first time you connect may take a while because JSSE goes through an SSL authentication process.

Using an unsigned JAR

To improve performance when using an unsigned JAR, you can edit Java's default security policy file using the instructions in Sun's security documentation. To enable EAServer's ORB to work in an unsigned environment:

- You must grant the ORB permission to read the proxy host settings, using one of these methods:

```
permission java.util.PropertyPermission "*", "read"
```

  or

```
permission java.util.PropertyPermission "javaplugin.proxy.config.*",
"read"
```

- The ORB may require socket connect permissions to connect to a proxy server.
- If you are using the sample test certificate generated by EAServer, the EAServer certificate authority must be installed. You can do this in either the *cacerts* or the *jssecacerts* keystore using this syntax:

  keytool -import -file <*file_name*> -keystore [cacerts | jssecacerts]

  The password for the cacerts keystore is "changeit".

Note With a signed applet, you do not need to set permissions at the plug-in level. A signed JAR file describes the type of permissions it requires.

Sample security file

You can find a sample JDK security file in the JDK installation, in file *jre/lib/security/java.security*.

Possible solutions for JSEE issues

Cannot load applet

If you cannot load an HTML applet from your Web browser:

1 In the Tools | Internet Options dialog box:

  - On the Connections tab, select Settings, then deselect Use a Proxy Server. Or, if you use a proxy server, verify the information is valid.

- On the Advanced tab, under Browsing, select Browse in a New Process.

2 In the Control Panel, double-click Java Plug-in 1.3.1_02. In the Java Plug-in Control Panel:

- On the Basic tab, select Enable Java plug-in and Show Java Console.
- On the Proxies tab, either select Use Browser Settings, or verify the Proxy Settings.
- Verify settings on the other tabs.

3 Shut down all Web browser sessions.

4 Close all Java console sessions; for example, from the Java Plug-in.

5 Restart your Web browser.

6 Delete all your temporary and cache files.

7 Reload the HTML applet page.

Debugging

If necessary, use the Java Plug-in console for debugging; set to debug level 5. If you reset the debug level, refresh the HTML applet.

CHAPTER 5

Using SSL in C++ Clients

Introduction

A C++ client can use IIOP tunnelled within SSL (also called IIOPS) to establish a secure session with EAServer.

Note For more information about security, including managing client side certificates, see Chapter 11, "Managing Keys and Certificates."

To establish a secure session with EAServer, follow these steps:

Step	What it does	Detailed explanation
1	Initialize the SSL security service as an ORB.	"Initializing the SSL security service" on page 44
2	Initialize the client ORB and create an *ORB* reference.	"ORB properties for secure sessions" on page 45
3	Use the *ORB* reference to create a *Manager* instance for the server.	"ORB properties for secure sessions" on page 45
4	Use the *Session* instance to create stub component instances. This step is the same regardless of whether the application uses SSL.	Chapter 6, "Developing CORBA/C++ Clients," of the *EAServer Corba Components Guide*.
5	Optionally, you can retrieve security information about the session.	"Retrieving session security information" on page 47

Initializing the SSL security service

To initialize the SSL security service, you must retrieve the SSL security service context and set the quality of security services as well as any global properties for that context.

You must decide if you want to respond to any authentication request by the server.

Retrieve the SSL security service context

In this example, you use CORBA::ORB_init to initialize the ORB as an instance, *orb1*.

```
CORBA::ORB_var orb1 =
CORBA::ORB_init(argc,argv, "");
```

Use resolve_initial_references to obtain the initial context from the SSL security service URL string (`SSLServiceProvider`) as an object reference, *object*, on *orb1*. You must use `SSLServiceProvider` as the URL string. You use CtsSecurity::SSLServiceProvider::_narrow to convert *object* to the *sslServProv* instance (an instance of the SSLServiceProvider interface).

```
object = orb1->resolve_initial_references
    ("SSLServiceProvider");
sslServProv = CtsSecurity::SSLServiceProvider
    ::_narrow(object);
```

Set the quality of security services and global properties

To return the available qualities of security services from the availableQop property, call getGlobalProperty on the sslServProv instance. The qualities of security services refer to the security profile characteristic, which specifies the supported CipherSuites.

```
// query Available quality of services and set
// whatever we want.
CtsSecurity::StringSeq_var * availQop =
  sslServProv->getGlobalProperty("availableQop");
```

At this time, you can also set any global properties, such as the callback component with the callbackImpl property. You specify the callback component using the setGlobalProperty method. The setGlobalProperty method takes the name of the global property, callbackImpl, and the name of the callback component. The name of the component is the DLL or shared library name (without the file extension) followed by a forward slash, and the package and component name separated by forward slashes as shown in this example:

```
// Set callbacks.

sslServProv->setGlobalProperty
    ("callbackImpl", "myDLL/myPackage/myComponent");
```

Enable client authentication

To respond to a server's request for client authentication, you can:

- Use the setGlobalProperty method to set the *certificateLabel* property to the client certificate to use when the server asks for one, or
- Use the *callback* interface to provide a dialog (GUI- or text-based) where the user can enter a certificate to be sent back to the server.

ORB properties for secure sessions

You must set the ORBqop property when initializing the client ORB in order to use one of the available security profile characteristics. The security profile characteristic lists the CipherSuites the client uses when negotiating an SSL connection. The client sends the list of CipherSuites that it uses to the server, and the server selects a cipher suite from that list. The server must choose the first cipher suite in the list that it can use.

In this example, the ORBqop property is specified as sybpks_strong (strong 128-bit encryption) and the ORBuserdata property is specified as `myUserData`. The CORBA::ORB_init method initializes the client ORB (*orb2*) with these properties.

```
// Now configure a specific ORB instance,
// overriding the default Quality of
// service. Might want to connect to a server
// only using 128bit encryption.
Properties props(argc, argv);
props.put("ORBqop", "sybpks_strong");
props.put("ORBuserData", myUserData);
orb2 = CORBA::ORB_init(props.argc(),
      props.argv(), "");
```

You can also set these properties when initializing the client ORB:

- **ORBcertificateLabel** Specifies the client certificate to use, if the server requests mutual authentication. The label is a simple name that identifies an X.509 certificate/private key in a PKCS #11 token. You must set this property if the server will request the client's certificate. If this property is not set and the server requests client authentication, credentialCallback is invoked. If you set this property to "any", then the getCertificateLabel method in the SSLCallback interface is invoked. If client authentication is requested and neither the certificateLabel property nor the credentialCallback is set, the SSL session fails.

- **ORBpin** Specifies the PKCS #11 token PIN. This is required for logging in to a PKCS #11 token for client authentication and for retrieving trust information. If this property is not set and the server requests client authentication, the Login callback implementation is invoked to get the PKCS #11 PIN. If this property is set to the value `any`, then the getPin method in SSLCallback interface is invoked. If a PKCS #11 token login is required and neither the Login callback property nor the PIN property are set, the SSL session fails. This property can be set application-wide using the SSLServiceProvider context. This property cannot be retrieved once it has been set.
- **ORBuserData** Specifies user data (string datatype). This is an optional property. Client code can set user data during ORB initialization and access it using SSLSessionInfo::getProperty method in the SSL callback implementation. This may be useful as a mechanism to store ORB-level context information that is otherwise not available through the SSLSessionInfo interface.

Creating a manager instance

Creating the manager instance for an SSL session is exactly like creating a manager instance for a non-SSL session, except that instead of specifying an IIOP port for the manager session in the string_to_object method, you specify the secure IIOP (specify `iiops`) port (the IIOPS default port number for mutual client-server authentication is `9002`). You must specify a port that supports the at least the level of security specified by the QOP setting.

Retrieving session security information

To retrieve security information about the session, narrow the component object reference, *typeObj*, to an SSL session, *sslSession*. Then use the getSessionInfo method to retrieve the session information from the SSL session and create an object reference for the session information. Use individual get methods to retrieve information about each SSL session property.

Note You cannot use the getName, getPassword, getAuthenticationStatus, getListener, getPeerAddress, getHostName. These methods are inherited from the SessionInfo interface.

```
// Obtain SSLSession information from
// typesObj.
CtsSecurity::SSLSession_var sslSession =
    CtsSecurity::SSLSession::
    _narrow(typesObj);
CtsSecurity::SSLSessionInfo_var
    sslSessionInfo = sslSession->getSessionInfo();

// Obtain user data (similar usage in user's
// SSL callback implementation)
String_var currentUserData =
    sslSessionInfo->getProperty("userData");

// Obtain details about server's certificate.
CtsSecurity::X509Certificate_var serverX509=
    sslSessionInfo->getPeerCertificate();
cout << "Connected to server: <<
    serverX509->getSubjectDN() << endl";

// get details about my certificate
CtsSecurity::X509Certificate_var clientX509=
    sslSessionInfo->getCertificate();
```

Creating an SSL callback component

An SSL callback component is a component that the client uses to execute callback methods. A callback method is a method that responds to SSL requests from EAServer. An SSL callback component resides on the client machine. To create an SSL callback, you must create a component DLL or shared library and deploy it on the client machine in a directory specified by the *PATH* environment variable. You can create the component in the same manner that you would create any other server-side component—using EAServer Manager and a C++ IDE.

You must specify the component DLL or shared library by using the setGlobalProperty method in the CtsSecurity::SSLServiceProvider interface to set the *callbackImpl* global property. For information, see "Set the quality of security services and global properties" on page 44.

Implementing callback methods

Although default implementations of the following callback methods are included with the EAServer client ORB, you can implement your own logic for these callback methods. To implement the default response for callback methods, code them to return the CORBA::NO_IMPLEMENT exception.

- **getCertificateLabel** The user is prompted with the available certificate labels and asked to choose one of them for client authentication.
- **getCredentialAttribute** The EAServer SSL client runtime engine retrieves credential attributes from the user on request.
- **getPin** The user is prompted with the PKCS #11 token password information and asked to provide a PIN for logging into the PKCS #11 token.
- **trustVerify** The user is prompted with server certificate information and asked to determine if the server certificate chain can be trusted and if the SSL session can proceed.

For more information about these callback methods, see the CtsSecurity::SSLCallback interface in the interface repository documentation. The interface repository documentation can be viewed in a Web browser by connecting to your server with this URL:

```
http://yourhost:yourport/ir/
```

where *yourhost* is the EAServer's host name and *yourport* is the HTTP port number.

CHAPTER 6 Using TLS and FIPS

This chapter describes TLS and FIPS protocols and how to use them to create secure EAServer connections.

Introduction

The National Institute of Standards and Technology (NIST) develops standards and guidelines for security and interoperability for federal computer systems. These guidelines are called the Federal Information Processing Standards (FIPS).

EAServer uses a cryptographic module to perform encryption and decryption, signing and verification, computing a checksum (or MAC) of data, and protecting security-sensitive data. These operations are invoked by the Transport Layer Security (TLS) runtime, a software implementation of a PKCS #12 interface, and key management utility routines.

EAServer utilizes a FIPS 140-2-certified cryptographic module provided by Certicom Cryptographic libraries.

For more information, see these Web sites:

- Cryptographic Module Validation Program Web site at http://csrc.nist.gov/cryptval/ – describes the FIPS standards, contains related documents and specifications, and answers commonly asked questions.

- Certicom Security Builder Government Standard Edition (GSE) Web site at http://www.certicom.com/index.php?action=product,sbgse – describes the FIPS 140-2 certified cryptographic module that is integrated into EAServer.
- Certificate and cryptographic module information at http://csrc.nist.gov/cryptval/140-1/1401val2003.htm#351 – contains a list of certificates issued by NIST, including a copy of the Certicom certificate.

TLS is a protocol based on Secure Sockets Layer (SSL) that is used to establish secure connections between a client and server. TLS can authenticate both the client and the server, and create an encrypted connection between the two.

The TLS protocol addresses some of the security concerns of SSL v3. FIPS requires TLS for use with a FIPS cryptographic module.

See RFC 2246 at http://www.faqs.org/rfcs/rfc2246.html for a complete description of TLS.

SSL/TLS and FIPS support

EAServer security includes support for SSL, certificate handling, and TLS, including:

1. keytool – a command line tool used to configure and modify the certificate database, generate certificate requests, and so on. See “Managing keys and certificates on EAServer” on page 102 for more information.
2. set-certificate – a command line script used to map certificates to users. See “Set-certificate” on page 102 for more information.
3. Two implementations for handling SSL/TLS protocol and related ciphersuites:
    - JSSE – JDK1.4 includes support for JSSE (Java Secure Socket Extension), allowing you to use existing applications with little modification.
    - Certicom SSL-J – includes a Java implementation of the SSL/TLS protocol that supports more ciphersuites than JSSE and is certified against FIPS140-2.

      Set the ORB option com.sybase.ejb.useJSSE to choose between JSSE and Certicom implementations on the client side. Certicom is the default if the variable is not set.

By default, the TLS version 1 protocol is enabled on the server and client.

❖ Enabling FIPS using the Certicom Java libraries

FIPS mode requires Certicom Java 1.4 libraries to be installed and running.

When FIPS mode is enabled, any SSL listener not using a FIPS-supported security profile is considered invalid, and does not start. See "Security characteristics" on page 55 for a list.

1. On the server – from the Web Management Console, select FIPS Mode Enabled for the server on which you are enabling FIPS. See "JSSE configuration" on page 53 for instructions.

   If you set the "-fips false" option, the server is started using the JSSE library.

   With FIPS enabled, you can use only certain algorithms (security characteristics) for quality of protection (QOP). See "Security characteristics" on page 55 for a list.

2. On the client – set the `com.sybase.ejb.fips` connection property to specify FIPS usage. Additional client-side properties added to support FIPS include:

   - `com.sybase.ejb.keystore`
   - `com.sybase.ejb.truststore`
   - `com.sybase.ejb.keystoreType`
   - `com.sybase.ejb.truststoreType`
   - `com.sybase.ejb.keystorePassword`
   - `com.sybase.ejb.truststorePassword`
   - `com.sybase.ejb.qop`
   - `com.sybase.ejb.useJSSE`
   - `com.sybase.ejb.certificateLabel`
   - `com.sybase.ejb.pin`

     `com.sybase.ejb.pin` is used to support backward compatibility. When this property is set and `keystorePassword` and `truststorePassword` are not specified, the `pin` property is used.

   With FIPS enabled, typical client code used to access an EJB on the server could look like:

```
Properties props = new Properties();
props.put(Context.INITIAL_CONTEXT_FACTORY,
"com.sybase.ejb.InitialContextFactory");
props.put(Context.PROVIDER_URL, "iiops://" + "localhost" + ":" + port1);
props.put(Context.SECURITY_PRINCIPAL, "admin@system");
props.put(Context.SECURITY_CREDENTIALS, "sybase1");
props.put("com.sybase.ejb.qop", "intl");
Context ctx = new InitialContext(props);
Object obj = ctx.lookup("ejb");
```

JMS clients
For JMS clients, set the `com.sybase.jms.fips` connection property to specify FIPS usage. Additional client-side properties added to support FIPS are the same as mentioned above for EJBs, except the property names use `jms` instead of `ejb`. For example, `com.sybase.ejb.keystore` becomes `com.sybase.jms.keystore`. You also need the proper Java libraries in the classpath, namely under the *lib/fips/* subdirectory.

If FIPS mode is enabled, EAServer logs the message `FIPS 140-2 mode enabled` to the console. If the mode is not set, no message is logged.

Enabling FIPS has the following effect on EAServer:

- Permits TLS protocol only by the SSL/TLS runtime engine.
- Permits the use of cipher suites and security characteristics listed in Table 6-1 on page 55.
- Accepts X.509 certificates signed using a SHA1WithRsa algorithm. Certificates signed with any other algorithm are not accepted and generate an error.
- Other cryptographic functionality that normally employ a non-FIPS approved algorithm now fail. For example, a PKCS #12 certificate containing a private key shrouded (signed) with a pbeWithSHA1And40bitRc4 algorithm fails to import, since RC4 is not a FIPS 140-2-approved algorithm. The private key and public keys must be shrouded using pbeWithSHA1And3KeyTripleDescbc.

For FIPS certification:

- When EAServer runs with JDK 1.4, it is FIPS certified.
- When EAServer runs with JDK 1.5, it is not FIPS certified.

JSSE configuration

The Java Secure Socket Extension (JSSE) is a set of Java packages that implements SSL and Transport Layer Security, which enables data encryption, server authentication, message integrity, and client authentication. JSSE is a client-side feature, which can be used with EAServer when it has been configured for SSL communication. See "Configuring JSSE" on page 96 for more information.

FIPS mode for Java-side cryptography usage

On the server, you must use the Certicom library to enable FIPS mode. The Java client uses the Certicom library by default to make SSL/TLS connections. To disable the Certicom Java library and use JSSE instead, set -Djsse=true or use the com.sybase.ejb.UseJSSE property in the ORB.

C/C++ clients use native calls into the SSL handler.

Java clients using JSSE to establish an SSL connection should use a FIPS-approved JSSE provider. EAServer SSL client runtime is not a JSSE provider.

Java clients and components that utilize a Java Cryptography Extension (JCE) provider should install a FIPS-enabled JCE provider to operate in a FIPS 140 mode. Further, Java components hosted in EAServer that also use a JCE provider should install a FIPS-enabled FIPS provider.

Compatibility with earlier versions

There is a new class in EAServer 6.0 called com.sybase.djc.security.SSLGlobalProperties, which can be invoked in two ways:

```
SSLGlobalProperties.callBackImpl=foo
```

In this case, foo is the fully qualified name for the class implementing the SSLCallBack interface, or

```
SSLGlobalProperties.callBackImpl=foo/bar
```

In this case, foo/bar refers to a component residing in the server.

You must use the Certicom library to use this SSLCallBack.

The EAServer SSL client runtime earlier than version 5.2 offers a mechanism to query the user and obtain the PKCS#11 pin, determine whether to trust the server certificate and set a client-side certificate to use for the SSL connection (if necessary).

The method getCertificateLabel() in the SSLCallback interface queries the user to determine which client-side certificate to use during the SSL handshake. The server asks the client to authenticate itself during mutual authentication, by sending a message as part of the SSL handshake requesting a client certificate. When this request arrives at the client, and the client has not set its client certificate, the SSL client runtime queries the user through the getCertificateLabel() method. For more details on this, see the corresponding IDL documentation.

The post-5.1 EAServer SSL client runtime does not invoke the getCertificateLabel() callback. If your client application relies on the getCertificateLabel() method being invoked/executed during mutual authentication using SSL, do the following after installing EAServer 5.2 or later:

1. In the shell used to run the client program, set the environment variable JAGSSL to true. This has the effect of using the earlier version of the EAServer SSL client runtime. The earlier version of the SSL runtime does not support TLS.

   PowerBuilder client applications that rely on mutual authentication should also set JAGSSL to true before running the application, but for a Java client you must set -DuseJSSE to true.

2. If you do require TLS support now, you must recode your client application. For example, if this is an EJB client application, set the client certificate label using the com.sybase.ejb.certificateLabel ORB property.

Enabling TLS-secure listeners

Define security profiles in EAServer using the Sybase Management Console. Associate the profile with a server listener and determine the SSL characteristics of the listener. The profile is used on the client side to set the SSL connection parameters. Follow the same procedures to assign a profile containing TLS characteristics to a listener. A profile has a security characteristic, which is a combination of the following properties:

- SSL or TLS cipher suite

- Authentication mode – server only, mutual, or none

Table 6-1 displays a list of the security characteristics and cipher suites used to support TLS. A characteristic that includes _mutual_ specifies:

- For a client – the client wants to authenticate to the server, or
- For a server – the client's certificate is necessary.

Table 6-1 lists the name, the level of authentication, the supported cipher suites for each TLS security characteristic, and if it supports FIPS.

FIPS-supported cipher suites for each TLS security characteristic are enabled when a server or client is operating in a FIPS 140-2 mode.

When EAServer or a client is operating in a FIPS-compliant mode, only the TLS protocol should be used. FIPS 140-2 has an approved list of algorithms. Due to this requirement, not all cipher suites are available while operating in a FIPS mode.

❖ Modifying security profile properties

1. From the Web Management Console, expand the Security folder.
2. Expand the Profiles folder.
3. Highlight the security profile whose properties you want to modify. The General Properties pane appears, from which you can define or modify these general profile properties:
   1. Certificate Label – the name of the certificate label for this profile. The certificate label identifies the certificate used for authentication.
   2. Security Characteristic – select a security characteristic to use for the security profile from the drop-down list. The characteristic defines the required level of security, including authentication. Table 6-1 lists the characteristics, cipher suite support, and authentication level.

Table 6-1: Security characteristics

Name of characteristic	Authenticates	Cipher suites	Supports FIPS?
domestic	server	RSA_WITH_3DES_EDE_CBC_SHA RSA_WITH_RC4_128_MD5 RSA_WITH_RC4_128_SHA RSA_WITH_DES_CBC_SHA RSA_EXPORT_WITH_RC4_40_MD5 RSA_EXPORT_WITH_DES40_CBC_SHA TLS_RSA_WITH_NULL_MD5 TLS_RSA_WITH_NULL_SHA	Yes

Name of characteristic	Authenticates	Cipher suites	Supports FIPS?
domestic_anon_tls	neither	DH_anon_WITH_3DES_EDE_CBC_SHA DH_anon_WITH_RC4_128_MD5 DH_anon_WITH_DES_CBC_SHA DH_anon_EXPORT_WITH_RC4_40_MD5 DH_anon_EXPORT_WITH_DES40_CBC_SHA The _anon profiles are used for anonymous Diffie-Hellman communications. Neither the client nor the server is authenticated.	No
domestic_mutual	client/server	RSA_WITH_3DES_EDE_CBC_SHA RSA_WITH_RC4_128_MD5 RSA_WITH_RC4_128_SHA RSA_WITH_DES_CBC_SHA RSA_EXPORT_WITH_RC4_40_MD5 RSA_EXPORT_WITH_DES40_CBC_SHA RSA_WITH_NULL_MD5 RSA_WITH_NULL_SHA	Yes
domestic_mutual_tls	client/server	RSA_WITH_3DES_EDE_CBC_SHA RSA_WITH_RC4_128_MD5 RSA_WITH_RC4_128_SHA RSA_WITH_DES_CBC_SHA RSA_EXPORT_WITH_RC4_40_MD5 RSA_EXPORT_WITH_DES40_CBC_SHA RSA_WITH_NULL_MD5 RSA_WITH_NULL_SHA	Yes
domestic_tls	server	RSA_WITH_3DES_EDE_CBC_SHA RSA_WITH_RC4_128_MD5 RSA_WITH_RC4_128_SHA RSA_WITH_DES_CBC_SHA RSA_EXPORT_WITH_RC4_40_MD5 RSA_EXPORT_WITH_DES40_CBC_SHA TLS_RSA_WITH_NULL_MD5 TLS_RSA_WITH_NULL_SHA	Yes
intl_mutual_tls	client/server	RSA_EXPORT_WITH_RC4_40_MD5 RSA_EXPORT_WITH_DES40_CBC_SHA	No
intl_tls	server	RSA_EXPORT_WITH_RC4_40_MD5 RSA_EXPORT_WITH_DES40_CBC_SHA	No
simple_tls	server	RSA_WITH_NULL_MD5 RSA_WITH_NULL_SHA	No
simple_mutual_tls	client/server	RSA_WITH_NULL_MD5 RSA_WITH_NULL_SHA	No

Name of characteristic	Authenticates	Cipher suites	Supports FIPS?
strong	server	RSA_WITH_3DES_EDE_CBC_SHA RSA_WITH_RC4_128_MD5 RSA_WITH_RC4_128_SHA	Yes
strong_mutual	client/server	RSA_WITH_3DES_EDE_CBC_SHA RSA_WITH_RC4_128_MD5 RSA_WITH_RC4_128_SHA	Yes
strong_tls	server	RSA_WITH_3DES_EDE_CBC_SHA RSA_WITH_RC4_128_MD5 RSA_WITH_RC4_128_SHA	Yes
strong_mutual_tls	client/server	RSA_WITH_3DES_EDE_CBC_SHA RSA_WITH_RC4_128_MD5 RSA_WITH_RC4_128_SHA	Yes
tls_rsa_with_3des_ede_cbc_sha	server	rsa_with_3des_ede_cbc_sha	Yes
tls_rsa_with_3des_ede_cbc_sha_mutual	client/server	rsa_with_3des_ede_cbc_sha	Yes
tls_rsa_with_aes_256_cbc_sha	server	rsa_with_aes_256_cbc_sha	Yes
tls_rsa_with_aes_256_cbc_sha_mutual	client/server	rsa_with_aes_256_cbc_sha	Yes
tls_rsa_with_aes_128_cbc_sha	server	rsa_with_aes_128_cbc_sha	Yes
tls_rsa_with_aes_128_cbc_sha_mutual	client/server	rsa_with_aes_128_cbc_sha	Yes
tls_rsa_with_des_cbc_sha	server	rsa_with_des_cbc_sha	Yes
tls_rsa_with_des_cbc_sha_mutual	client/server	rsa_with_des_cbc_sha	Yes
tls_rsa_with_rc4_128_sha	server	rsa_with_rc4_128_sha	Yes
tls_rsa_with_rc4_128_sha_mutual	client/server	rsa_with_rc4_128_sha	Yes
tls_rsa_export_with_rc4_40_md5	server	rsa_export_with_rc4_40_md5	No
tls_rsa_export_with_rc4_40_md5_mutual	client/server	rsa_export_with_rc4_40_md5	No

CHAPTER 7

Creating and Using Custom Security Components

This chapter describes how to use custom components to perform security tasks such as user authentication and authorization. These features allow you to create your own components to customize EAServer security and to integrate with third-party enterprise security software such as Netegrity SiteMinder.

Topic	Page
Introduction	59
Using a custom authentication service	60
Using a custom role service	61
Using a custom authorization service	62

Introduction

Custom authentication service components, authorization service components, and role service components are security-domain-based. You can specify a specific component name or the full Java class name in the corresponding property for the security domain.

For example, you can name an authorization service component authservicecomponent-foo.bar, which is an authorization service component for the package foo, and component bar.

See "Configuring domains" on page 77 for information about defining custom service components used to enforce domain security.

Using a custom authentication service

You can specify different authentication service components for individual security domains in EAServer 6.0 using a custom authentication service component. For example, to configure the security domain named bar to use a custom authentication service component, choose an authentication method of "cts-auth", see "Login methods" on page 81, and enter the name of the customized service component. All users registered under the bar domain are authenticated by this component.

As an example, if you require the client user name to match a remote database user name, code the component to retrieve the client user name and password and attempt to log in to the remote database.

The security domain delegates authentication requests to this component or class:

- If using a CORBA component, specify its name in the form "MyPackage/MyComp". The component must implement the CtsSecurity::AuthService IDL interface.

  This interface contains the method checkSession. Your code for this method can check the client's user name and password and the status of other authentication checks, that is, whether the client's credentials have passed OS authentication or SSL authentication checks. Your code can perform additional authentication checks and auditing. For more information, see the documentation for the CtsSecurity::AuthService IDL interface.

- You can also use a Java class with the simplified authenticate API, which is recommended if your implementation is in Java, rather than using a component. If you use a Java class, specify its fully qualified class name. The class must contain a method with the signature:

```
public boolean authenticate(String username, String password)
{
...
}
```

Maintaining authenticated sessions

EAServer provides a mechanism by which applications can extend and maintain the authenticated session beyond the lifetime enforced by EAServer. This mechanism uses the methods CtsSecurity::SessionInfo::setName and CtsSecurity::SessionInfo::getCallerPrincipal.

If these methods are implemented, you must also handle the authorization of the user by either implementing a role service or authorization service. The internal role checking performed by EAServer does not work unless the alternate user name is added to the authorized user's list for the role. As the alternate user name that is set using the setName API can be dynamic, the role service or authorization service should work in tandem with the authentication service to authorize the user.

CtsSecurity::SessionInfo::setName is a method that can be called only when your custom authentication component is running. When this method is called from the custom authentication component, the server sets the reference to the authenticated security credentials. When the client needs to be authenticated again, the custom authentication component returns the original principal name by calling CtsSecurity::SessionInfo::getCallerPrincipal(string alternate_name).

The CtsSecurity::SessionInfo::setName method has no effect if clients obtain component instances using CSIv2. If you are using CSIv2, you must use a JAAS module in addition to an authentication or other component. See Chapter 8, "Using the JAAS API."

Retrieving HTTP session information

In a custom authentication component implemented in Java, you can call the com.sybase.jaguar.server.Jaguar.getHttpServletRequest() method to retrieve the HTTP servlet request (if any) that triggered the authentication event. This method returns null if the authentication event is not associated with an HTTP request (for example, if the authentication is for a component invocation).

Using a custom role service

You can install your own component that performs access control based on role membership. The component must implement the CtsSecurity::RoleService IDL interface. Your custom role service evaluates user membership in EAServer roles, so authorization in your application is still dependent on the role names associated with a package, component, method, or Web resource collection. Using a role service eliminates the need to define role memberships in the Sybase Management Console. For example, you might code your component to retrieve role membership information from a database.

You can also use a Java class with the simplified isUSerInRole API. This is recommended if the implementation is in Java, rather than using a component. If using a Java class, specify its fully qualified class name. The class must contain a method with the signature:

```
public boolean isUserInRole(String user, String role)
{
...
}
```

Creating a role service

The role service must be a stateless component that implements the CtsSecurity::RoleService IDL interface:

```
interface RoleService {
     boolean isMember(
          in CtsSecurity::SessionInfo sessionInfo,
          in string role);
 };
```

isMember checks if the authenticated client is a member of the role. The client's credentials are obtained from sessionInfo. The server first checks if the role is defined in the repository. If the role is defined, membership checks are performed. If the role is not defined, the server assumes that the user is not a member of the role, and the role service is invoked. The result from this method is cached by the server, where it can be referenced for the same client/role combination, provided the internal cache has the relevant information.

Using a custom authorization service

You can create and install your own component to authorize clients to access resources (packages, Web applications, or applications) on any EAServer.

Deciding whether to use the authorization services or role service

Using an authorization service offers greater control than using a role service, but the API is more complicated than the role service API.

The role service acts server-wide, and evaluates user membership in declared EAServer roles associated with a resource (package, component, method, or Web resource collection).

An authorization service can control access to all resources on a server, or only those in a particular application, Web application, or package. With the authorization service, you can allow or deny access to resources with no dependencies on roles configured in EAServer.

You can use both a role service and an authorization service. For example, you could use a role service to configure role-based resource permissions in the Web Management Console, but use an authentication service to create audit logs to track user access to resources.

Creating the authorization service

An authorization service component must implement the CtsSecurity::AuthorizationService IDL interface, and be stateless to support refresh. It must be one of:

- Java CORBA
- PowerBuilder Non-Visual User Object (NVO)
- C++ CORBA
- Stateless Component Object Model (COM)

Usage

```
interface AuthorizationService {
     boolean isAuthorized(
             in CtsSecurity::SessionInfo sessionInfo,
             in StringSeq resource,
             in StringSeq roles,
             in boolean isMember,
              in long permTimeDelta);
```

isAuthorized checks if the client is authorized to access a resource. The client’s credentials can be obtained from sessionInfo.

resource is the entity the client is trying to access. The resource is represented as an ordered array of strings, and each string represents a scoped entity. A string starts with one of these prefixes:

- A: – application
- WA: – Web application

- P: – package
- C: – component
- M: – method
- S: – servlet
- HM: – HTTP method (GET, PUT, POST, and so on)
- URL: – complete URL being accessed

For example, if the resource being accessed is a servlet or a JSP that belongs to a Web application, which belongs to an application, then the array might contain the following string sequence:

```
A:ApplicationName; WA:WebApplicationName; S:servletName; HM:httpMethod;
```

roles lists all the roles associated with the resource (if any). The server first checks if the role is defined in the repository. If the role is defined, then membership checks are performed and if the user is in at least one of the roles, the authorization check succeeds. isAuthorized is still invoked, and the caller can audit the resource access. isMember is set to AUTH_OK to indicate that the authorization succeeded. If a role is not defined, it is assumed that the user is not a member of the role.

If the user is not a member of all the roles, then isMember is set to AUTH_FAILED. isAuthorized then determines whether to authorize the client. isAuthorized returns true if the user is allowed access to the resource, and returns false otherwise.

permTimeDelta is the time difference in seconds, since the last time isAuthorized was invoked for this particular user and resource combination. This value can be used by the authorization component logic to determine whether to audit the event. A value of zero (0) implies that the isMember was not determined from the internal permission cache. A positive value indicates that the isMember was determined from the internal permission cache. permTimeDelta is always less than or equal to the server-wide authorization permission cache timeout value (see the Permission Cache Timeout property described in Table 10-1 on page 78).

CHAPTER 8 Using the JAAS API

Introduction

The Java Authentication and Authorization Service (JAAS) provides a framework and standard programming interface for authenticating users and assigning privileges. JAAS is based on the Pluggable Authentication Module (PAM) standard, which extends the access-control architecture of the Java 2 platform to support user-based authentication and authorization.

JAAS support is provided in EAServer as an alternative authentication mechanism. EAServer supports user-name and password-based JAAS authentication. The code-level authorization component of JAAS is not supported in this version of EAServer.

EAServer 6.0 does not support the use of JAAS in clients. The security principal and credentials for EJB or JMS InitialContext are used exclusively. EAServer 6.0 support for JAAS with EJB, JMS and Web clients is limited to the JAAS loginMethod method for a security domain. See “Login methods” on page 81 for more information.

See the Java software Web site at http://www.java.sun.com/products/jaas/ for more information about JAAS.

There are several new terms that are used throughout this chapter:

Principal represents a user identity that is used to gain access to a computing service. Typically, a user’s login name or public key.

Credentials represents a security attribute of a principal. Typically, a user's password or public-key certificate. The credential is set in the subject when a principal is authenticated successfully.

Subject is an entity that has one or more principals and corresponding security attributes.

A *login context* is a JAAS framework for developing applications independent of underlying authentication technology.

A *login module* is an authentication module that can be plugged in under a Java application using JAAS framework. The module implements the JAAS `javax.security.auth.spi.LoginModule` interface. It performs any authentication either on its own or by interacting with any external authentication service such as Kerberos.

A *callback* is a mechanism by which a login module retrieves authentication parameter values needed for authentication from the Java application. The callback is implemented in a Java application to pass required parameters to the login module. It implements the `javax.security.auth.callback` interface.

The *JAAS configuration file* (*$EAServer/config/eas_jaas.cfg*) specifies:

- One or more authentication modules for an application
- The order in which authentication modules are invoked
- Other parameters and options

This is the interaction between an application, login module, and the JAAS configuration file:

1. The Java application program instantiates a login context that consults the JAAS configuration file to load all of the login modules configured for that application.
2. The login module requests the Java program to provide the user name and password using the JAAS callback mechanism.
3. The login module executes custom code to authenticate the user and set up the subject with valid principals and credentials if successfully authenticated.

The subject can then be used to gain access to controlled resources or to perform privileged actions.

JAAS in EAServer

Over time, you may need to modify or replace authentication infrastructure due to deficiencies, enhancements, or applications requiring a different security policy. EAServer support for JAAS login modules simplifies replacement and modification of the underlying authentication mechanism.

Configure domain-wide login modules that are used to authenticate clients trying to gain access to applications, Web applications, and servlets/JSPs. The JAAS login method (see “Login methods” on page 81) points to the JAAS configuration file, which determines the login module to use for a specific domain.

Based on the contents of the configuration file, EAServer invokes any specified login modules. If a login module is not defined, JAAS is bypassed and the server uses the regular mechanism, if any, for authentication. For example, if credentials are passed to a server and no login module is defined, the server uses operating system authentication, if enabled.

If a login module is defined, it overrides any other authentication service that may be installed, and passes the request for authentication to the login module.

Enabling JAAS for a domain

To enable JAAS, specify JAAS as the login method for the domain (see “Configuring domains” on page 77), from which you specify the JAAS configuration file and section name in the server properties. EAServer uses the login module in that section for authentication.

Your EAServer installation contains a JAAS configuration file, *eas_jaas.cfg*, in the *config* subdirectory.

JAAS for connectors

The J2EE connector architecture enables you to write portable Java applications that can access multiple transactional enterprise information systems. A resource adapter is a specialized connection factory that provides connections for EJBs, Java servlets, JSPs, and CORBA-Java components.

Each resource adapter has a set of managed connection factories with their own property files. The Java Connection Manager (JCM) classes create the connection factories and manage a pool of connections for a resource adapter. You can use JAAS to authenticate a resource adapter, resource principal, or the application component’s caller principal when accessing enterprise information systems.

See “Configuring connectors” Chapter 4, “Database Access” in the *EAServer System Administration Guide* for more information about connectors.

Enabling JAAS-based authentication for connectors

To use JAAS for authentication, you must enable JAAS on the EAServer where the resource adapter is located. See “JAAS in EAServer” on page 67.

An entry in the login configuration file is identified by the name of the resource adapter for which JAAS is used. If container-managed authentication is set, any component that tries to obtain a connection from resource adapter’s connection factory is authenticated by the login module defined by the configuration file entry.

CHAPTER 9

Deploying Applications Around Proxies and Firewalls

Proxy servers are typically used to isolate computers on a corporate network from the Internet. Connections to and from the Internet must go through the organization’s proxy servers. EAServer does not include proxy server software, but supports client connections through proxy servers to EAServer.

This chapter describes security features for clients that connect over the Internet or through proxy servers. You should understand the basics of your client model before reading this chapter.

Connecting through proxy servers

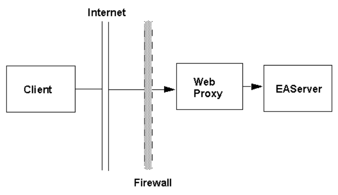
Proxy servers are typically used to constrain and secure connections from an organization’s computers to sites that require connecting across the Internet. To enhance security, some network configurations require all Internet connections to go through a proxy server, including IIOP connections to EAServer.

EAServer supports two types of proxy servers for clients, Web proxies and reverse proxies.

Using Web proxies

Web proxies typically act as a gateway for outgoing connections from a group of workstations. Web proxies can be used to enhance network security, for example, a proxy may constrain which servers clients can connect to and which protocols may be used, and log outgoing connections. Web proxies may also be used to improve network performance, by caching the results of frequently executed Web requests. Web proxies are also referred to as HTTP-connect-based proxies. Figure 9-1 illustrates how clients connect to servers through a Web proxy.

Figure 9-1: Connecting through a Web proxy



Clients connect to EAServer through a Web proxy as follows:

1. Using the HTTP protocol, the client connects over the Internet to the Web proxy, embedding the destination server address inside a specially formatted HTTP connect request.
2. The Web proxy connects to the host and port indicated in the initial HTTP connect request.
3. Subsequent traffic is forwarded unchanged between the client and server until the connection is closed.

Java applets can use the built-in proxy configuration provided by Web browsers such as Netscape Navigator. See your Web browser's documentation for information on configuring proxy addresses. For applets running in a Web browser, HTTP and HTTPS-tunnelled IIOP connections automatically use the browser's proxy connection settings. HTTP-tunnelled IIOP connections go through the browser's configured HTTP proxy. HTTPS-tunnelled IIOP connections go through the browser's configured secure proxy.

Other applications must specify the Web proxy address by setting the Web proxy host and port in the properties described below.

Properties that affect Web proxy use

Table 9-1 describes the client properties that configure connections that must be opened through a Web proxy. You must set these properties in addition to any properties that you would set to connect directly to EAServer. For JMS clients, see the *JMS User's Guide*.

Table 9-1: Properties that affect Web proxy use

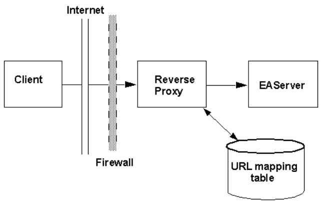
C++/ PowerBuilder property	CORBA property	EJB property	Specifies
`ORBWebProxyHost` or environment variable `JAG_WEBPROXYHOST`	`com.sybase.CORBA.WebProxyHost`	`com.sybase.ejb.WebProxyHost`	Specifies the host name or IP address of the Web proxy server. Does not apply to Java applets running in a Web browser, which use the proxy address specified by the browser's proxy configuration. There is no default for this property, and you must specify both the host name and port number properties.
`ORBWebProxyPort` or environment variable `JAG_WEBPROXYPORT`	`com.sybase.CORBA.WebProxyPort`	`com.sybase.ejb.WebProxyPort`	Specifies the port number at which the Web proxy server accepts connections. Does not apply to Java applets running in a Web browser, which use the proxy address specified by the browser's proxy configuration. There is no default for this property, and you must specify both the host name and port properties.
`ORBHttp` or environment variable `JAG_HTTP`	`com.sybase.CORBA.http`	`com.sybase.ejb.http`	By default, the client ORB attempts to open IIOP connections, then attempts an HTTP-tunnelled connection if plain IIOP fails. Since Web proxy connections require HTTP tunnelling, set this to true to eliminate the performance overhead of trying plain IIOP connections before trying HTTP-tunnelled IIOP.

C++/ PowerBuilder property	**CORBA property**	**EJB property**	**Specifies**
`ORBHttpExtraHeader` or environment variable `JAG_HTTPEXTRAHEADER`	`com.sybase.CORBA.HttpExtraHeader`	`com.sybase.ejb.HttpExtraHeader`	An optional setting to specify what extra information is appended to the header of each HTTP packet sent to the Web proxy server. There is no need to set this property unless your HTTP proxy server has special protocol requirements. By default, the following line is appended to each packet: `User-agent: Jaguar/`*`major.minor`* where *major* and *minor* are the major and minor version numbers of your EAServer client software, respectively. You can set this property to specify text to be included at the end of each HTTP header. If multiple lines are included in the setting, they must be separated by carriage return and line feed characters. If the setting does not include a "User-agent:" line, then the default setting above is included in the HTTP header.
N/A	`com.sybase.CORBA.useJSSE`	`com.sybase.ejb.useJSSE`	Use the Java Secure Sockets Extension (JSSE) classes for secure HTTP-tunnelled (HTTPS protocol) connections. JSSE provides an alternative to the built-in SSL implementations when secure connections are needed from an applet running in a Web browser. Additional configuration may be required to use this option.

Using reverse proxies

Reverse proxies typically act as a gateway for incoming connections to an organization's network servers, preventing direct connections from clients outside the firewall to servers inside the firewall. The reverse proxy can enhance security, by restricting protocols and logging connection activity. Reverse proxies may also act as caches to respond to common requests. In some cases, multiple reverse proxies may be deployed to cache results from one server, as a form of load balancing. Figure 9-2 shows how clients connect through a reverse proxy.

Figure 9-2: Connecting through a reverse proxy



Clients connect to EAServer through a reverse proxy as follows:

1 The client connects to the reverse proxy, and sends each IIOP packet tunnelled inside an HTTP or HTTPS packet. The destination server address is encoded in the HTTP packet header as:

```
GET /host/port/HIOP/1.0/...
```

Where *host* is the target EAServer host name, and *port* is the target EAServer port number.

2 The reverse proxy uses its URL mapping configuration (shown as a database in the figure) to determine the destination server address.

3 The reverse proxy opens a connection to the destination server, or reuses an existing connection, and forwards the request to the server, then forwards the response to the client.

Reverse-proxy configuration

For use with EAServer, you must configure your reverse proxy server's URL mapping table to recognize the EAServer addresses embedded in the HTTP requests sent by the client runtime. For each EAServer that clients can connect to through the server, configure a mapping for the following URL prefix:

```
GET /host/port/HIOP/1.0/
```

Where *host* is the target EAServer listener host name, and *port* is the target EAServer listener port number. For each EAServer that you deploy behind the reverse proxy, add a mapping for each IIOP, IIOPS, and Message Service listener address. If you deploy an EAServer cluster behind a reverse proxy, add mappings for each server in the cluster.

Properties that affect reverse proxy use

To connect through a reverse-proxy server, you can set the properties in Table 9-2. You must set these properties in addition to any properties that you would set to connect directly to EAServer. For JMS clients, see the *JMS User's Guide*.

Table 9-2: Properties that affect reverse proxy use

C++/ PowerBuilder property	CORBA property	EJB property	To indicate
`ORBProxyHost` or environment variable `JAG_PROXYHOST`	`com.sybase.CORBA.ProxyHost`	`com.sybase.ejb.ProxyHost`	Specifies the machine name or the IP address of the reverse-proxy server.
`ORBProxyPort` or environment variable `JAG_PROXYPORT`	`com.sybase.CORBA.ProxyPort`	`com.sybase.ejb.ProxyPort`	Specifies the port number of the reverse-proxy server, typically 80 for HTTP-tunnelled connections or 443 for SSL (HTTPS-tunnelled) connections.
`ORBHttp` or environment variable `JAG_HTTP`	`com.sybase.CORBA.http`	`com.sybase.ejb.http`	Set this property to true if the reverse-proxy server requires HTTP-tunneled connections. If you do not set this property, connections still go through, but only after the client ORB first tries to open an IIOP connection. Setting the property eliminates the overhead that is incurred by trying plain IIOP each time a connection is made.
`ORBforceSSL` or environment variable `JAG_FORCESSL`	`com.sybase.CORBA.forceSSL`	`com.sybase.ejb.forceSSL`	Set this property to true if the connection to the reverse proxy must use SSL (HTTPS) tunnelling, but the connection from the proxy to the EAServer does not use SSL tunnelling.

C++/ PowerBuilder property	CORBA property	EJB property	To indicate
`ORBqop` or environment variable `JAG_QOP`	`com.sybase.CORBA.qop`	`com.sybase.ejb.qop`	In applications that connect to a proxy using SSL (HTTPS) tunnelling, set the quality of protection (QOP) to a security characteristic that matches the one supported by the reverse-proxy server. See “Configuring security profiles” on page 90 for more information. If the connection to the proxy server requires SSL, but the connection from the proxy does not, do not set the QOP; instead, set the forceSSL property to true. Do not set QOP in Java applets that use SSL. Instead, code the applet to connect to a listener that supports the required security level.
N/A	`com.sybase.CORBA.autoProxy`	`com.sybase.ejb.autoProxy`	In Java applets, set this property to true to enable connections to a reverse-proxy server. You must also configure your applet to download through the reverse-proxy server itself. The default is false. This property is ignored if the client is not a Java applet, or has not initialized the Java ORB with the ORB.init method that takes an Applet parameter. When automatic proxy is enabled, the ORB uses the applet’s download address as the reverse-proxy server address. If the port number is 443, SSL (HTTPS tunnelling) is used; otherwise, HTTP tunnelling is used.

CHAPTER 10 Security Configuration Tasks

This chapter describes the most common declarative security mechanisms provided by EAServer and configured with the Sybase Management Console.

Configuring domains

EAServer 6.0 uses the “user@domain” style for user security. This improves the ability for administrators to separate different user security settings into different domains. For example, you can separate users into different domains, and use a different access control policy for each domain.

As a result, every user is a member of a domain. For example, admin@system represents the admin user in the system domain. If you provide only the user name, the server assigns it to the ‘default’ domain; user@default. Any user, in any domain, can have “admin-role” assigned to it. “admin@system” is the default. EAServer includes two preinstalled domains; system, and default.

In addition, there is no default password set for the user 'admin@system'. When you install EAServer, you are asked to provide an admin password. Otherwise use the set-admin-password command to set the admin password. You must set this password before you can start EAServer.

❖ **Adding a new domain**

1. From the Web Management Console, expand the Security folder.
2. Right-click the Domains folder and select Add.
3. The New Domain wizard guides you through adding a new security domain. You can then set domain properties.

❖ **Setting domain properties**

1. From the Web Management Console, expand the Security folder.
2. Select Domains | *domain_name*, where *domain_name* is the domain for which you are setting properties. The right pane displays the domain properties.
3. Select the General, Password Settings, or Login Properties tab to access the properties. See Table 10-1 (General) and Table 10-2 (Password Settings) for a description of the properties.

   The Login Properties tab contains one property, Certificate Digest Algorithm, which defines the Secure Hash Algorithm (SHA) used for logins to this domain. SHA-512 is the default.
4. Select Apply to apply your changes, or Reset to restore them to their previous values.

Table 10-1: General domain properties

Property name	**Description**
Login Method	Select the method used for login from the drop-down list. See “Login methods” on page 81 for a description of each.
Login Cache Timeout	Specifies how long to cache login information before timing out. The value is in seconds. Once a user is logged in, the login information (user name and password) is maintained in the system. The next time the same user logs in, the system checks the cached information and compares it with the password. This way, normal authentication is bypassed and performance enhanced.
Login Failure Lock Threshold	Specifies the number of times the client can retry with wrong credentials before being locked out.
Login Failure Lock Timeout	Specifies the amount of time a user is locked out when the login failure lock threshold is reached.

Property name	Description
Access Control Policy	The access control policy for the selected login method. The Access Control Policy should be configured to the name of the policy class for all security domains which might be associated with JACC-enabled modules. The policy class selected for this purpose is expected to perform JACC policy checks only. See “JACC (JSR-115) support” on page 84. JRE-related policy can be separately enabled (if required) using standard JRE security policy mechanisms.
Audit Access Denied	When set to true, any failure caused by a user trying to access a server resource is logged to the server’s log file (*$EAServer/logs/server_name.log*).
Audit Access Permitted	When set to true, any success caused by a user trying to access a server resource is logged to the server’s log file (*$EAServer/logs/server_name.log*).
Permission Cache Timeout	The number of seconds that the result of an authorization (access control) check is cached. This applies to both denied access and permitted access. Caching of authorization results improves performance.
Authentication Service	The name and path to your custom authentication service component (if any). This allows you to customize EAServer security and to integrate with third-party enterprise security software. See Chapter 7, “Creating and Using Custom Security Components.”
FTP Host Name(s)	The host name of the FTP server to which the security domain delegates authentication requests. A comma-separated list can be used to specify multiple servers (for high availability, not load balancing).
FTP Port Number	The port number of the FTP server to which the security domain delegates authentication requests.
Http Resource URL(s)	The URL for an HTTP resource which the security domain attempts to access when delegating authentication requests to an HTTP server. A comma-separated list can be used to specify multiple URLs (for high availability, not load balancing).
JAAS Login Context	The name of a JAAS (Java Authentication and Authorization Service) login context that has been configured in *jaas.conf*. Refer to the JDK documentation for details on *jaas.conf*.
JDBC Driver Class	The JDBC driver class to be used for database authentication.
JDBC Database URL(s)	The URL for a JDBC database which the security domain attempts to access when delegating authentication requests to a database server. A comma-separated list can be used to specify multiple URLs (for high availability, not load balancing).
JNDI Initial Context Factory	The name of an initial context factory class to be used for JNDI authentication.

Property name	Description
JNDI Provider URL	The provider URL which the security domain attempts to access when delegating authentication requests to a JNDI server.
JNDI Lookup Name	The name of a JNDI-bound object which the security domain attempts to lookup when delegating authentication requests to a JNDI server.
Role Service Component	The name and path to your custom role service component (if any). This allows you to customize EAServer security and to integrate with third-party enterprise security software. See Chapter 7, “Creating and Using Custom Security Components.”
Web Realm Names	EAServer contains a default security realm. The security realm is a container used to store the roles used to allow and limit access to your Web services. When you connect to EAServer from the Web Management Console, you see the security realm.

Table 10-2: Domain password properties

Property name	Description
Password Hash Algorithm	The algorithm used to encrypt the password. The strongest supported algorithm is used (currently SHA-512).
Minimum Password Length	The minimum total number of characters.
Maximum Password Length	Defines the maximum password length. Used only if the Login Method is Local-hash. The default value is 14.
Minimum Password Letters	The minimum number of letters contained in the password.
Require Mixed Case Passwords	When true, the password must contain both uppercase and lowercase letters.
Password Start Characters	When set, the password must start with one of the characters in the string you enter here. For example: a, B, r, t, 3.
Minimum Password Digits	The minimum number of digits (0 – 9) the password contains.
Minimum Password Special Characters	The minimum number of special characters (*, &, #, and so on) the password contains.
Password Special Characters	A comma-separated list of special characters the password can contain.
Password End Characters	A comma-separated list of characters required to end the password. If left blank, any character can end the password.
Retain Old Passwords	The number or historical passwords saved for this user.

Login methods

This section describes the various login methods supported by EAServer 6.0:

- CTS-Auth – enter the name of the authentication service component in the field provided. See Chapter 7, “Creating and Using Custom Security Components.”
- FTP – enter one or more host names for FTP login validation. If there are multiple host names (allowing for fault tolerance), use a comma as a separator.”). The default value is “localhost”. You must also enter the FTP port number.

- HTTP – enter one or more resource URLs to be used for HTTP login validation. If there are multiple URLs (allowing for fault tolerance), use double-semicolons as a separator.”). The default value is http://*host_name*:8000/wsh/login.
- JAAS – supply the name of a JAAS (Java Authentication and Authorization Service) login context in the Access Control Policy field which has been configured in the JAAS configuration file. See Chapter 8, “Using the JAAS API.”
- JDBC – supply the name of the jConnect driver in the Access Control Policy field. The default value is com.sybase.jdbc2.jdbc.SybDriver. You must also supply one or more database URLs to be used for JDBC login validation. If there are multiple URLs (allowing for fault tolerance), use double-semicolons as a separator.
- JNDI – supply the name of the initial context factory class, the provider URL to be used for initialization of the context factory, and the name to be looked up to determine that the login was successful. Upon login, the user name and password are verified against the JNDI entry.
- Local-hash – The default login method. Local-hash uses a hashing algorithm to encrypt the password and store it on a local table using either Secure Hash Algorithm 1 (SHA1) or Secure Hash Algorithm 512 (SHA-512).
- None – requires no authentication.
- OS Auth – native operating system authentication. User names for an EAServer connection map directly to a login name on the host operating system. To enable OS Auth, EAServer 6.0 must be running under a process that has local admin privileges. Ensure that the user that is running the process is a member of the Local Administrator Group. You may need to log off and log back on in order for this to take effect. You must also:

    a Select Start Menu | Control Panel | Administrator Tools | Local Security Policy. This starts the Local Security Settings Window.

    b Choose Local Policies | User Rights Assignment | Act As Part of the Operating System.

    c Use “Add User or Group” to add the user that EAServer will be running under.

    d Log off and log back in for this to take effect.

e Define a domain definition that proxies the authentication to the OSLoginModule. The following is a sample domain definition. You should modify files only through the Web Management Console or a configuration script.

```
#Instance Properties
#Mon Aug 08 15:43:51 EDT 2005
#Location:
${djc.home}/Repository/Instance/com/sybase/djc/security/?SecurityDo
main/sybase.com.properties
jdbcDriverClass=com.sybase.jdbc2.jdbc.SybDriver
passwordSpecialCharacters=
loginFailureLockTimeout=600
webRealmNames=
maximumPasswordLength=14
accessControlPolicy=${javax.security.jacc.policy.provider}
requireMixedCasePasswords=false
auditAccessPermitted=false
passwordHashAlgorithm=SHA-512
jndiProviderURL=
user-roles\:delafran=wmc.admin
minimumPasswordSpecialCharacters=0
minimumPasswordLetters=2
jdbcDatabaseURL=jdbc\:sybase\:Tds\:localhost\:2638
all-roles\:=all
all-users\:=delafran
ftpPortNumber=21
minimumPasswordLength=6
httpResourceURL=http\://???\:8000/wsh/login
loginMethod=os-auth
permissionCacheTimeout=3600
retainOldPasswords=8
loginCacheTimeout=3600
role-users\:all=delafran
ftpHostName=localhost
jndiInitialContextFactory=
passwordEndCharacters=
minimumPasswordDigits=1
jndiLookupName=
passwordStartCharacters=
auditAccessDenied=false
jaasLoginContext=${jaas.login.context}
loginFailureLockThreshold=5
```

JACC (JSR-115) support

This section describes how to use JACC (Java Authorization Contract for Containers) in EAServer. For details about JACC see the JACC specification at http://java.sun.com/j2ee/javaacc.

If you intend to use JACC as the access control mechanism for EJB or web modules, you must specify the name of the JACC policy implementation class in each security domain which defines users who access the EJB or web modules. The default value of this property is taken from a Java system property in accordance with the JACC specification. If this property was not previously set, then you should redeploy all EJB and web modules for which you wish to use JACC.

You may find it easier to use role-based access control. See Chapter 7, "Creating and Using Custom Security Components" for information.

JACC support requires you to pass an extra command-line option to the deploy command, as these examples illustrate:

- `deploy myweb.jar -jacc`
- `deploy myweb.war -jacc`
- `deploy myweb.ear -jacc`

Deploying with the -jacc option creates additional Ant configuration scripts, for example:

- *ejbjar-myejb-jacc.xml*
- *webapp-myweb-jacc.xml*
- *application-myapp-jacc.xml*, as well as one script for each EJB Jar file and Web application in the application.

Redeploying discards the previously generated JACC script(s), if any.

Each generated JACC configuration script contains "configure" and "delete" targets. Refer to the "Provider Configuration Subcontract" in the JACC specification for details.

Run the config-jacc and delete-jacc command line tools to configure and delete targets. For example:

- `delete-jacc webapp-myweb`
- `config-jacc ejbjar-myejb`

You must create and edit an Ant script named *jacc-provider-info.xml*, and set the property jacc.factory to the name of your JACC PolicyConfigurationFactory class. If this property is not set, the PolicyConfiguration object returned by the default factory simply echos configuration information to the console when you run the config-jacc command.

Running config-jacc or delete-jacc for an application runs it on all EJBs and Web modules in the application (similar to the existing configure and recompile commands).

When run, the config-jacc command extends (does not delete) the previous provider configuration. To replace the provider configuration, first run delete-jacc.

After running config-jacc or delete-jacc, refresh the security domain(s) configured with a JACC policy.

Run the configure jacc-on command to enable JACC globally for all subsequent deployment. You can turn it off again using configure jacc-off.

Managing users

This section describes how to add and manage users in EAServer 6.0.

❖ **Adding a new user**

1. From the Web Management Console, expand the Security folder.
2. Right-click the Users folder and select Add.
3. The New User wizard guides you through the process of adding a new user. When you click Finish, you can define roles assigned to the user.

❖ **Modifying user properties**

1. From the Web Management Console, expand the Security folder.
2. Expand the Users folder.
3. Highlight the user whose properties you want to modify. The General Properties pane displays the inherited and excluded roles for the user. See "Configuring roles" on page 86 for more information. The Assigned Roles Properties pane allows you to define or modify the roles assigned to the selected user.

To modify the roles assigned to a user, select:

- All – assigns all defined roles to the user.
- None – no roles are assigned to the user.
- Select – allows you to select individual roles, and assign them to the user.

Configuring roles

This section describes how to add roles to a domain and control role membership.

The EAServer authorization model is based on *roles*, which are defined in the Sybase Management Console. Each role can include and exclude specific user names or digital IDs. If you use native operating system authentication, you can also include and exclude operating system group names; all users in the specified group are affected.

All roles belong to the default security domain. For example, role1@system would not exist.

There are a couple of things to keep in mind when configuring roles:

1. You do not need to specify a domain name when defining a role, since all roles belongs in the default domain.
2. Role changes take effect after a Security Domain refresh. But the permissionCacheTimeout is relevant, successful and failed authorization results are cached and even after a refresh of the domain, it requires waiting for the timeout before cached results for any given user are automatically dropped.
3. A user can only be authorized by the domain to which it belongs. Authentication and authorization are limited to a domain for the particular user.
4. foo@default and foo@system are considered the same as far as authorization, although they are different entities in the server. For example:
    a. You can create and authorize a user called testuser@default to access the Sybase Management Console.

b There is a user called testuser@default. Since testuser belongs to the default security domain, you must create a role called admin-role and add the user to the newly created role.

c Since the Sybase Management Console is authorized only by admin-role, testuser@default is authorized.

You must either refresh or restart EAServer for any role changes to take effect.

❖ **Adding a role to a domain**

1. From the Web Management Console, expand the Security folder.
2. Right-click the Roles folder and select Add.
3. Follow the New Role wizard instructions to define a new role. When prompted for Security Role Name, enter the name of a role in the form *role_name*. The role is assigned to the "default" domain.

❖ **Controlling role membership**

1. From the Web Management Console, expand the Security folder.
2. Expand the Roles folder.
3. Select the role that you are modifying.
4. Control role membership by selecting these tabs:
   - General – provides general information about the role, including name, inherited roles, and role membership.
   - Assigned Roles – allows you to grant authorization to groups by selecting All (all users of a group are authorized members), None (no user belonging to the group is authorized), and Select (allows you to individually select groups to which you grant authorization).
   - Excluded Roles – allows you to exclude authorization to individual groups by selecting All (all users within the group are excluded), None (no group is excluded), and Select (allows you to individually select groups to exclude).
   - Excluded Users – allows you to exclude users by denying access to the users based on their user names and not the authorized groups to which they belong.

❖ **Refreshing the roles**

1. Right-click the Roles folder.
2. Select Refresh Node.

❖ **Defining a new role**

1 Right-click the Roles folder.

2 Select Add. Enter the required information in the subsequent dialogs:

- Security Role Name – the name of the role you are defining.
- Click Finish – define the Assigned Roles, Excluded Roles, and Excluded Users for this role by selecting the corresponding tab. See “Controlling role membership” on page 87.

❖ **Deleting an existing role**

1 Highlight the Roles folder. You see a list of existing roles.

2 Highlight the role you want to delete.

3 Right-click the role and select Delete. This option is available only to the owner of the role or the admin user.

4 Click Yes to confirm deletion of the selected role.

Note Only the owner or a member of the role named Admin-Role can delete a role, except for Admin-Role itself, which cannot be deleted.

❖ **Modifying an existing role**

1 Highlight the Roles folder. You see a list of existing roles.

2 Highlight the role you want to modify.

3 Select any of the tabs to make your modifications.

4 Make your modifications and click Apply.

Admin role granularity

This feature allows you to control access with greater precision, and is configured in the security domain.

For example the following command indicates the role “foo” is authorized to call the method refresh, which resides in the com.sybase.jaguar.server.ManagementImpl component, whose first parameter is “WebApplication” and second parameter is “bar”:

```
role-
access\:foo=com.sybase.jaguar.server.ManagementImpl.refresh(Webapplication,
bar)
```

Supported parameter types are:

- int
- float
- double
- java.lang.String
- java.lang.Integer
- java.lang.Float
- java.lang.Double

Supported components include com.sybase.jaguar.server.managementImpl.

Inherited roles

Inherited roles are those roles that are included indirectly. For example, if you include roleB in roleA, and roleB includes roleC, roleC displays as an inherited role in roleA's property page.

Configuring OS authentication

To enable OS authentication for an EAServer domain, follow the procedures described in "Configuring domains" on page 77, and use OS Auth as the Login Method. See "Login methods" on page 81.

Configuring security profiles

This section describes how to add security profiles to a domain. See Chapter 6, "Using TLS and FIPS" for information about FIPS/TLS security profiles.

Note EAServer contains two predefined security profiles; sample1, which features the domestic security characteristic, and sample2, which features the domestic_mutual security characteristic.

Security profiles define the security characteristics of a client-EAServer session. Assign a security profile to a listener, which is a port that accepts client connection requests of various protocols. EAServer can support multiple listeners. Clients that support the same characteristics can communicate to EAServer via the port defined in the listener.

Each security profile has an associated security characteristic. A security characteristic is a name that has a set of cipher suites associated with it. A security characteristic, along with the cipher suites, defines these characteristics of a client/server connection:

- **Protocol** All profiles use SSL version 3 as the underlying protocol. IIOPS and HTTPS traffic is tunneled through SSL.
- **Authentication** Whether or not authentication is used. Profiles can support:
  - No authentication – neither client nor server need to provide a certificate for authentication.
  - Server authentication – only the server must provide a certificate to be accepted or rejected by the client.
  - Client and server authentication – both the client and server supply certificates to be accepted or rejected by the other.
- **Encryption strength and method** Whether or not data is encrypted, and if so, the key strength and method of the encryption.
- **International use** All cipher suites are available domestically, but not all are suitable for export outside of the United States and Canada.
- **Hashing method** The method used to create the message digest.

For example, the cipher suite SSL_RSA_WITH_NULL_MD5 can be interpreted as:

- SSL – the protocol used. All profiles use SSL.

- RSA – the key exchange algorithm used.
- NULL – no encryption.
- MD5 – the hash method used to compute the message digest.

Table 10-3 and Table 10-4 clarify the relationship between cipher suite terminology and security characteristics.

Table 10-3: Cipher suite terms

Name	Defines	Description
SSL	Protocol	SSL protocol uses public-key encryption to establish secure Internet communications.
RSA DH_anon	Key exchange algorithm	RSA and DH (Diffie-Hellman) are public-key cryptography systems, which define both authentication and encryption: • RSA provides full encryption and authentication support. • DH_anon provides only encryption support.
EXPORT	Suitable for export	Because of export regulations, some cipher suites are not suitable for export. Only cipher suites that contain the word EXPORT are suitable for international use.
NULL	No encryption	Data is not encrypted.
DES 3DES DES40 RC4_40 RC4_128	Encryption algorithms	System: Key length: DES 56 3DES 168 DES40 40 RC4_40 40 RC4_128 128 The greater the key length, the greater the encryption strength.
EDE CBC	Encryption and decryption modes	CBC and EDE are modes by which DES algorithms are encrypted and decrypted.
SHA MD5	Hash function	SHA and MD5 are hash methods used to compute the message digest when generating a digital signature.

Note Browsers do not support anonymous cipher suites.

❖ **Adding a new security profile**

1 From the Web Management Console, expand the Security folder.

2 Right-click the Profiles folder and select Add.

3 The New Security Profile wizard guides you through adding a new security profile. After you click Finish, define the security profile properties.

❖ **Modifying security profile properties**

1 From the Web Management Console, expand the Security folder.

2 Expand the Profiles folder.

3 Highlight the security profile whose properties you want to modify. The General Properties pane appears, from which you can define or modify these general profile properties:

   1 Certificate Label – the name of the certificate label for this profile. The certificate label identifies the certificate used for authentication:

      - For server and Java clients, certificate label corresponds to the name of the key entry retrieved by the keytool -list command.
      - For C++ clients, certificate label corresponds to the certificate names generated by the sc-tool -list command.

   2 Security Characteristic – select a security characteristic to use for the security profile from the drop-down list. The characteristic defines the required level of security, including authentication. Table 10-4 lists the characteristics, cipher suite support, and authentication level.

Table 10-4: Security characteristics

Name of characteristic	Authenticates	Cipher suites
domestic	server	RSA_WITH_3DES_EDE_CBC_SHA RSA_WITH_RC4_128_MD5 RSA_WITH_RC4_128_SHA RSA_WITH_DES_CBC_SHA RSA_EXPORT_WITH_RC4_40_MD5 RSA_EXPORT_WITH_DES40_CBC_SHA TLS_RSA_WITH_NULL_MD5 TLS_RSA_WITH_NULL_SHA
domestic_anon	neither	DH_anon_WITH_3DES_EDE_CBC_SHA DH_anon_WITH_RC4_128_MD5 DH_anon_WITH_DES_CBC_SHA DH_anon_EXPORT_WITH_RC4_40_MD5 DH_anon_EXPORT_WITH_DES40_CBC_SHA The _anon profiles are used for anonymous Diffie-Hellman communications. Neither the client nor the server is authenticated.

Name of characteristic	Authenticates	Cipher suites
domestic_anon_tls	neither	DH_anon_WITH_3DES_EDE_CBC_SHA DH_anon_WITH_RC4_128_MD5 DH_anon_WITH_DES_CBC_SHA DH_anon_EXPORT_WITH_RC4_40_MD5 DH_anon_EXPORT_WITH_DES40_CBC_SHA The _anon profiles are used for anonymous Diffie-Hellman communications. Neither the client nor the server is authenticated.
domestic_mutual	client/server	RSA_WITH_3DES_EDE_CBC_SHA RSA_WITH_RC4_128_MD5 RSA_WITH_RC4_128_SHA RSA_WITH_DES_CBC_SHA RSA_EXPORT_WITH_RC4_40_MD5 RSA_EXPORT_WITH_DES40_CBC_SHA RSA_WITH_NULL_MD5 RSA_WITH_NULL_SHA
domestic_mutual_tls	client/server	RSA_WITH_3DES_EDE_CBC_SHA RSA_WITH_RC4_128_MD5 RSA_WITH_RC4_128_SHA RSA_WITH_DES_CBC_SHA RSA_EXPORT_WITH_RC4_40_MD5 RSA_EXPORT_WITH_DES40_CBC_SHA RSA_WITH_NULL_MD5 RSA_WITH_NULL_SHA
domestic_tls	server	RSA_WITH_3DES_EDE_CBC_SHA RSA_WITH_RC4_128_MD5 RSA_WITH_RC4_128_SHA RSA_WITH_DES_CBC_SHA RSA_EXPORT_WITH_RC4_40_MD5 RSA_EXPORT_WITH_DES40_CBC_SHA TLS_RSA_WITH_NULL_MD5 TLS_RSA_WITH_NULL_SHA
intl	server	RSA_EXPORT_WITH_RC4_40_MD5 RSA_EXPORT_WITH_DES40_CBC_SHA
intl_mutual	client/server	RSA_EXPORT_WITH_RC4_40_MD5 RSA_EXPORT_WITH_DES40_CBC_SHA
intl_mutual_tls	client/server	RSA_EXPORT_WITH_RC4_40_MD5 RSA_EXPORT_WITH_DES40_CBC_SHA
intl_tls	server	RSA_EXPORT_WITH_RC4_40_MD5 RSA_EXPORT_WITH_DES40_CBC_SHA
simple	server	RSA_WITH_NULL_MD5 RSA_WITH_NULL_SHA

Name of characteristic	Authenticates	Cipher suites
simple_mutual	client/server	RSA_WITH_NULL_MD5 RSA_WITH_NULL_SHA
simple_tls	server	RSA_WITH_NULL_MD5 RSA_WITH_NULL_SHA
simple_mutual_tls	client/server	RSA_WITH_NULL_MD5 RSA_WITH_NULL_SHA
ssl_rsa_with_3des_ede_cbc_sha	server	rsa_with_3des_ede_cbc_sha
ssl_rsa_with_3des_ede_cbc_sha_mutual	client/server	rsa_with_3des_ede_cbc_sha
ssl_with_rc4_128_sha	server	rsa_with_rc4_128_sha
ssl_with_rc4_128_sha_mutual	client/server	rsa_with_rc4_128_sha
strong	server	RSA_WITH_3DES_EDE_CBC_SHA RSA_WITH_RC4_128_MD5 RSA_WITH_RC4_128_SHA
strong_mutual	client/server	RSA_WITH_3DES_EDE_CBC_SHA RSA_WITH_RC4_128_MD5 RSA_WITH_RC4_128_SHA
strong_tls	server	RSA_WITH_3DES_EDE_CBC_SHA RSA_WITH_RC4_128_MD5 RSA_WITH_RC4_128_SHA
strong_mutual_tls	client/server	RSA_WITH_3DES_EDE_CBC_SHA RSA_WITH_RC4_128_MD5 RSA_WITH_RC4_128_SHA
tls_rsa_with_3des_ede_cbc_sha	server	rsa_with_3des_ede_cbc_sha
tls_rsa_with_3des_ede_cbc_sha_mutual	client/server	rsa_with_3des_ede_cbc_sha
tls_rsa_with_aes_256_cbc_sha	server	rsa_with_aes_256_cbc_sha
tls_rsa_with_aes_256_cbc_sha_mutual	client/server	rsa_with_aes_256_cbc_sha
tls_rsa_with_aes_128_cbc_sha	server	rsa_with_aes_128_cbc_sha
tls_rsa_with_aes_128_cbc_sha_mutual	client/server	rsa_with_aes_128_cbc_sha
tls_rsa_with_des_cbc_sha	server	rsa_with_des_cbc_sha
tls_rsa_with_des_cbc_sha_mutual	client/server	rsa_with_des_cbc_sha
tls_rsa_with_rc4_128_sha	server	rsa_with_rc4_128_sha
tls_rsa_with_rc4_128_sha_mutual	client/server	rsa_with_rc4_128_sha
tls_rsa_export_with_rc4_40_md5	server	rsa_export_with_rc4_40_md5
tls_rsa_export_with_rc4_40_md5_mutual	client/server	rsa_export_with_rc4_40_md5

❖ Deleting a security profile

1. Right-click the profile entry you want to delete and select Delete.
2. Follow the wizard instructions to delete the security profile.

Associating a listener with a security profile

This section describes how to add HTTP(S), and IIOP(S) listeners, and associate them with a security profile.

❖ **Adding an EAServer listener**

1. Right-click the Listeners folder and select Add.
2. Follow the wizard instructions to create/add a listener. Include:
   - Name – the name of the new listener
   - Protocol – select https or iiops from the drop-down list.
   - Host name – the name of the host name/server to which this listener belongs. Once defined the listener is automatically associated with the server.
   - Port number – the port number of the host used as the listener. For example, 8001.
   - Security Profile – the name of the security profile used by the listener. For example, default.
3. Click Finish when done.

❖ **Modifying an EAServer listener**

1. You can modify a listener from either the Servers or Listeners folder. This procedure uses the Servers folder. From the Servers folder, expand the server from which you are modifying the listener.
2. Select the Listeners tab and click the listener you are modifying.
3. Select the General tab and make your modifications. For example:
   - Protocol – https
   - Port – 8001
   - Security Profile – default
4. Click Apply to apply your changes.

Sample configuration

This section provides a sample HTTPS listener configuration.

1 Modify the listener property. See “Configuring listeners” on page 39 in Chapter 3, “Creating and Configuring Servers” of the *EAServer System Administration Guide* for instructions.

   Add a new listener with these settings:

```
port=8001
host=${host.name}
securityProfile=default
protocol=https
```

2 Add the new listener to the server.

3 Associate a *truststore* and *keystore* files with the server (they can be the same file). Supported file types are JKS and PKCS12. See “Configuring JSSE” on page 96 for instructions.

   See “Managing keys and certificates on EAServer” on page 102 for information about managing the keystore.

4 Restart the server.

Use the same procedure for associating an IIOPS listener with a server, except add a new IIOPS listener instead of HTTPS.

Configuring JSSE

The Java Secure Socket Extension (JSSE) is a set of Java packages that implements SSL and Transport Layer Security, which enable data encryption, server authentication, message integrity, and client authentication. JSSE is a client-side feature, which can be used with EAServer when it has been configured for SSL communication.

❖ Configuring JSSE on the server

1 From the Web Management Console, expand the Servers folder.

2 Expand the server for which you are configuring JSSE.

3 Select the Security tab.

4 Configure JSSE by completing these properties:

- SSL Trust Store – the path of an SSL truststore. The trust store is a trusted keystore. For example, if you receive data from an entity that you already trust, and if you can verify that the entity is the one it claims to be, then you can assume that the data really came from that entity.

  Add an entry to a truststore only if the user makes a decision to trust that entity. By generating a keypair or importing a certificate, you grant trust to that entry, and any entry in the keystore is considered a trusted entry.

- SSL Trust Store Password – the password used to access the trust store.
- SSL Trust Store Type – the format of the truststore. Possible values are JKS and PKCS12.
- SSL Key Store – the path to the key store. A key store is a database of key material (certificates or key pairs for example), which are used for authentication and data integrity. Supported key store types include PKCS12 and JKS.
- SSL KeyStore Password – the password used to access the keystore.
- FIPS Mode Enabled – enable FIPS, which has the following effect on EAServer:
  - Permits TLS protocol only by the SSL/TLS runtime engine.
  - Permits the use of TLS cipher suites and security characteristics.
  - Accepts X.509 certificates signed using a SHA1WithRsa algorithm. Certificates signed with any other algorithm are not accepted and generate an error.
  - Other cryptographic functionality that normally employs a non-FIPS approved algorithm now fails. For example, a PKCS #12 certificate containing a private key shrouded (signed) with a pbeWithSHA1And40bitRc4 algorithm fails to import, since RC4 is not a FIPS 140-2-approved algorithm. The private key and public keys must be shrouded using pbeWithSHA1And3KeyTripleDescbc.
- JSSE Key Store Type – valid options are PKCS12 or JKS.

Sample configuration

This example illustrates how you might configure EAServer to use JSSE:

❖ **Configuring JSSE and adding an HTTPS listener**

1. From the Web Management Console, expand the Servers folder.
2. Expand the server for which you are configuring JSSE.
3. Select the JSSE Configuration tab.
4. Modify these properties:
    - SSL Key Store and SSL Trust Store – point to the keystore file named *keystore.jks*.
    - SSL Key Store Type and SSL Trust Store Type – JKS.
    - SSL Key Store Password – changeit.
    - SSL Trust Store Password – changeit, if you are using default trust store that ships with the JDK.
    - FIPS Mode Enabled – select this option.

    There are four entries in the *keystore.jks* file, one of which is a key entry used to start an SSL listener.
5. Specify a key entry name for the associated profile.
6. Create a new security profile that uses a TLS security characteristic:
    1. From the Security folder, right-click Profiles and select Add.
    2. Follow the wizard to create a new security profile using these values:
        - Security Profile Name – test_tls.
        - Certificate Label – use keytool commands to identify a label. For example:

          ```
          keytool -list -keystore ${keystore}
          ```

          You can see keyentry and any associated alias. Use the desired entry name as the certificate label.
        - Security Characteristics – select domestic_tls from the drop-down list.
    3. Click Apply to apply your changes.
7. Define an HTTPS listener:

   1. From the Servers folder, expand the server to which you are adding the listener.
   2. Select the Listeners properties tab and click https1.
   3. Select the General tab and add an HTTPS listener at port 8001 using these settings:
      - Protocol – https
      - Port – 8001
      - Security Profile – test_tls
   4. Click Apply to apply your changes.

8 Add the listener to the listener properties.

Configuring database set-proxy for CMP

Current versions of Adaptive Server Enterprise and Adaptive Server Anywhere allow a user to assume the identity and privileges of another user. You can use this feature with any database that recognizes this command:

```
set session authorization "login-name"
```

When proxy support is enabled, connections retrieved from the cache are set to act as a proxy for the user name associated with the EAServer client. The user name specified in the cache properties must have set-proxy privileges in the database and server used by the cache.

DataSource contains:

- A new Boolean property: setSessionAuth, with the default set to false.
- A new string property: setSessionAuthSystemID, default is " " (an empty string). If this property is not specified, Datasource defaults to the user name property for the DataSource when the user is the system user.

DatabaseType contains a new Boolean property: supportsSetSessionAuth, with the default set to false. For Adaptive Sever Anywhere and Adaptive Server Enterprise, the default configuration script value is set to true.

If setSessionAuth is enabled for a DataSource and the DatabaseType supports this feature, when a connection is retrieved from the ConnectionPool set session authorization "login-name" on the connection is called, where login-name is the client identity.

If the current client identity is the system user, the setSessionAuthSystemID is used.

To support this feature, these com.sybase.djc.sql.ConnectionPool methods are changed to call set session authorization:

- public Object getConnection(ConnectionWrapper wrapper)
- public Object getConnection()

Note Adaptive Server Enterprise users may also have to set the database option ddl in tran.

CHAPTER 11 Managing Keys and Certificates

This chapter describes how to manage keys and certificates for SSL security in EAServer.

SSL overview

You can configure EAServer to accept client connections over secure protocols IIOPS and HTTPS using:

- keytool commands to manage key pairs and certificates for EAServer.
- Sybase Management Console to define security profiles that establish various levels of security on EAServer and assign them to a listener. Profiles allow you to determine:
    - Client and server authentication requirements
    - Encryption and decryption algorithms

  See "Configuring security profiles" on page 90 and "Associating a listener with a security profile" on page 95 for information on establishing security profiles and assigning them to EAServer listeners.
- EAServer to use certificates and listeners to authenticate clients, if necessary, and encrypt and decrypt data.

Managing keys and certificates on EAServer

EAServer includes SSL support, using the JDK's built-in keystore and keytool commands to manage the certificates and keys in the key store (both on EAServer and for Java clients). The keytool executable is located in the *jdk/jdk1.4/bin* subdirectory of your EAServer installation. You can list the installed keys using the keytool command with the -storetype option set to "pkcs12":

```
keytool -list -storetype pkcs12
```

For information about all of the keytool commands, enter:

```
keytool -help
```

For more information, see the Sun keytool command page at http://java.sun.com/j2se/1.4.2/docs/tooldocs/windows/keytool.html.

Set-certificate

The set-certificate script (located in the *bin* subdirectory of your EAServer installation) is required for client certificate to user name mappings. You cannot use mutual authentication in EAServer 6.0 without this mapping.

set-certificate sets mapping information for a given username by using the keytool alias "user." For example, `user:jim@mydomain.com` (or just `user:jim` if "jim" is in the "default" domain). You cannot just use keytool to directly set up the mapping, because some mapping information is also needed in the "default" security domain properties file. However, you can use keytool to examine or export the certificate after you use set-certificate for a user.

Keytool examples

This section describes keytool commands used to manage a keystore, and assumes that your keystore is located in *m:\target1.4\Repository\Security\keystore.jks*:

- To see what is contained in the keystore:

```
keytool -list -keystore m:\target1.4\Repository\Security\keystore.jks
-storepass <storepass>
```

  <storepass> is the keystore's password.

- To import a certificate:

```
keytool -import -file <certificate file> -keystore
m:\target1.4\Repository\Security\keystore.jks -storepass <storepass>
-noprompt -trustcacerts -alias <alias>
```

<alias> is the logical name for the certificate stored in the keystore.

- To delete a certificate:

```
keytool -delete -alias <alias> -keystore
m:\target1.4\Repository\Security\keystore.jks -storepass <storepass>
```

<alias> is the logical name for the certificate you want to delete.

- To create a new keyentry in a keystore:

```
keytool -genkey -keystore m:\target1.4\Repository\Security\keystore.jks
```

The certificate request must be signed by a CA. Alternatively, you can self-sign the certificate by using the -selfcert keytool option.

Note keytool can read/manipulate PKCS12 type keystores (specify a -storetype pkcs12 in the command line), but *cannot* import it into another jks type keystore. jks is the default keystore type for keytool commands.

Client-side security

This section describes the commands and procedures used for client-side security.

Using SSL in PowerBuilder clients

You can create PowerBuilder clients that connect to EAServer using SSL connections, using techniques similar to that used in other client types. Since PowerBuilder connects to EAServer using the C++ client ORB, SSL in PowerBuilder requires a full EAServer C++ client installation, and uses sc-tool commands to manage client-side certificates. See “Client-side security certificate tool (sc-tool)” on page 104 for more information about sc-tool and your PowerBuilder documentation.

Client-side SSL

When the client communicates with EAServer over SSL, the client must obtain and manage its certificates. Client-side SSL is separated into two parts:

- Java clients – uses keytool commands, which are included in the JDK distribution. keytool can modify the certificate database, generate certificate request, and so on. See "Managing keys and certificates on EAServer" on page 102.
- C/C++ clients – uses sc-tool commands, included as a batch file in the *bin* subdirectory of your EAServer installation, to manage certificates and the certificate database. See "Client-side security certificate tool (sc-tool)" on page 104.

Client-side security certificate tool (sc-tool)

When the client communicates with EAServer over SSL, the client must obtain and manage its certificates. sc-tool is a command line client tool used for managing the client-side C/C++ certificate database.

To use sc-tool (*sc-tool.bat* on Windows, and *sc-tool.sh* on UNIX), set JAGUAR_CLIENT_ROOT on the client to the location of DJC_HOME (in order for the client to pick up the correct certificate database), and add the location of the tool to the client's path.

❖ Using sc-tool

1. Run sc-tool from the *bin* directory of your EAServer installation.
2. Enter sc-tool -help for information about all sc-tool options, including:
   - Request a Certificate:

```
sc-tool -certreq [-alias <alias>] [-keyStrength <sigalg>] [-
keyStrength <keystrength>] [-CN <CommonName>]  [-UserID <UserID>] [-
EmailID <EmailID>] [-Org <Orgnization>] [-OrgUnit OrgnizationUnit>]
[-Locality <Locality>] [-State <state>] [-Country <Country>] [-
ReqName <requestor name>] <-SrvAdmin server admin> [-file <csr_file>]
<-Phone phone number>  <-UTF8>
```

csr_file – the file name to which the request is saved. The remaining certificate parameters are used so a CA can sign it.

alias – the new certificate name.

sigalg – the length for the key, can be 512, 1024, and so on.

keypass – the password used to protect the certificate.

- Delete a client-side certificate:

```
sc-tool -delete -alias <alias> [-storepass <storepass>]
```

alias – the name or the certificate label which you are deleting.

storepass – the name of the keystore that contains the certificate.

- Export a certificate:

```
sc-tool -export [-alias <alias>] [-file <cert_file>] [-storepass
<storepass>] [-exportType <exportType>]
```

alias – the name or the certificate label in the certificate database which you are exporting.

file – the path/name of the file to which the exported certificate is written.

storepass – the name of the keystore that contains the certificate.

exportType – only pkcs7 is allowed as the type.

- Import a certificate:

```
sc-tool -import [-file <cert_file>] [-keypass <keypass>] [-storepass
<storepass>]
```

file – the certificate file to be imported.

keypass – required if importing a private key. keypass is not needed when cert_file does not include a private key.

storepass – the password for the certificate database.

If cert_file trails with pfx or p12, it is a PKCS12 file, and keypass must be specified. Keypass is the password for recovering the private key.

- List all available certificates in the database:

```
sc-tool -list -storepass password [-userCerts true/false] [-
trustedCerts true/false]  [-CACerts true/false]  [-otherCerts
true/false] [-verbose]
```

You can choose what certificates to list. -verbose displays detailed information of the certificate(s).

- Change the database password:

```
sc-tool -changepin -storepass [-new <new_storepass>]
[-storepass <storepass>]
```

Changes the PIN for the certificate database.

Index