Oracle® Communications Instant Messaging Server

Security Guide

Release 10.0

E61908-01

September 2015

ORACLE®

Oracle Communications Instant Messaging Server Security Guide, Release 10.0

E61908-01

Contents

Preface

This guide provides guidelines and recommendations for setting up Oracle Communications Instant Messaging Server in a secure configuration.

Audience

This document is intended for system administrators or software technicians who work with Instant Messaging Server.

Related Documents

For more information, see the following documents in the Instant Messaging Server documentation set:

- *Instant Messaging Server Release Notes*: Describes new features, fixes and enhancements to existing features, known issues, troubleshooting tips, and required third party products and licensing.
- *Instant Messaging Server Installation and Configuration Guide*: Describes the requirements for installing Instant Messaging Server, installation procedures, and post-installation tasks.
- *Instant Messaging Server System Administrator's Guide*: Describes the tasks and concepts for administering Instant Messaging Server.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

1

Instant Messaging Server Security Overview

This chapter provides an overview of Oracle Communications Instant Messaging Server security.

Basic Security Considerations

The following principles are fundamental to using any application securely:

1. **Keep software up to date.** This includes the latest product release and any patches that apply to it.
2. **Limit privileges as much as possible.** Users should be given only the access necessary to perform their work. User privileges should be reviewed periodically to determine relevance to current work requirements.
3. **Monitor system activity.** Establish who should access which system components, how often they should be accessed, and who should monitor those components.
4. **Install software securely.** For example, use firewalls, secure protocols (such as SSL), and secure passwords. See "Performing a Secure Instant Messaging Server Installation" for more information.
5. **Learn about and use Instant Messaging Server security features.** See "Implementing Instant Messaging Server Security" for more information.
6. **Use secure development practices.** For example, take advantage of existing database security functionality instead of creating your own application security.
7. **Keep up to date on security information.** Oracle regularly issues security-related patch updates and security alerts. You must install all security patches as soon as possible. See "Critical Patch Updates and Security Alerts" on the Oracle Web site at:

   `http://www.oracle.com/technetwork/topics/security/alerts-086861.html`

Understanding the Instant Messaging Server Environment

When planning your Instant Messaging Server implementation, consider the following:

- Which resources must be protected?

  For example:

  - GlassFish Server (Instant Messaging 8.3 and prior releases)
  - Instant Messaging Server

  - Instant Messaging Server multiplexor
  - Protocols: HTTP, WMAP, SMTP, WCAP, LDAP, XMPP, HTTPBIND, and SIP/SIMPLE

- From whom am I protecting the resources?

  In general, resources must be protected from everyone on the Internet. But should the Instant Messaging Server deployment be protected from employees on the intranet in your enterprise? Should your employees have access to all resources within environment? Should the system administrators have access to all resources? Should the system administrators be able to access all data? You might consider giving access to highly confidential data or strategic resources to only a few well trusted system administrators. On the other hand, perhaps it would be best to allow no system administrators access to the data or resources.

- What happens if protections on strategic resources fail?

  In some cases, a fault in your security scheme is easily detected and considered nothing more than an inconvenience. In other cases, a fault might cause great damage to companies or individual clients that use Instant Messaging Server. Understanding the security ramifications of each resource helps you protect it properly.

Note: Oracle recommends that you configure Instant Messaging Server in secure mode.

Overview of Instant Messaging Server Security

Instant Messaging Server provides security in the following ways:

- LDAP: The basic level of security is through LDAP. The Instant Messaging server communicates with Directory Server for both authentication and user search. For chat to occur, the users must be in LDAP.
- Roles and policies: You can define roles and policies for those roles that enforce chat access, conference room access, contact list management, and user settings.
- Single sign-on: You can configure end users to authenticate once (that is, log on with user ID and password) and have access to multiple applications.
- Encryption: In conjunction with the TLS protocol, Instant Messaging Server provides client-to-server and server-to-server encrypted communications as well as certificate-based authentication between servers.

For an overview of operating system security, see *Oracle Solaris Security for System Administrators*.

Recommended Deployment Topologies

You can deploy Instant Messaging Server on a single host or on multiple hosts, splitting up the components into multiple front-end hosts and multiple back-end hosts. For more information, see the topic on planning your Instant Messaging Server deployment in *Instant Messaging Server Installation and Configuration Guide*.

The general architectural recommendation is to use the well-known and generally accepted Internet-Firewall-DMZ-Firewall-Intranet architecture.

Operating System Security

This section lists Instant Messaging Server-specific OS security configurations. This section applies to all supported OSs.

Firewall Port Configuration

Instant Messaging Server communicates with various components on specific ports. Depending on your deployment and use of a firewall, you might need to ensure that the firewalls are configured to manage traffic for the following components:

- XMPP port (default 5222)
- Multiplexed XMPP port (default 45222)
- XMPP Server port (default 5269)
- Notification Server port (default 47676)

Close all unused ports, especially non-SSL ports. Opt for SSL-enabled ports, instead of non-SSL ports, for all communications (for example: HTTPS, IIOPS, t3s).

For more information about securing your OS, see your OS documentation.

Transport Layer Security

In an Instant Messaging Server deployment, you can configure Transport Layer Security (TLS) for secure communication.

Instant Messaging uses a startTLS extension to the TLS 1.0 protocol for client-to-server and server-to-server encrypted communications and for certificate-based authentication between servers. In the latter case, a certificate is used to validate the identity of the server to which the client connects, but certificates are not used for authentication.

See "Implementing Instant Messaging Server Security" for more information.

LDAP Security

Instant Messaging Server requires a Directory Server to provide LDAP services. See the topic on using LDAP in *Instant Messaging Server System Administrator's Guide* for more information on configuring LDAP access, using schema, and other aspects of LDAP management.

To enhance client security in communicating with Directory Server, use a strong password policy for user authentication. For more information on securing Directory Server, see "Directory Server Security" in *Oracle Directory Server Enterprise Edition Administration Guide*.

2

Performing a Secure Instant Messaging Server Installation

This chapter presents planning information for your Oracle Communications Instant Messaging Server system and describes recommended deployment topologies that enhance security.

For more information about installing Instant Messaging Server, see *Instant Messaging Server Installation and Configuration Guide*.

Pre-Installation Tasks

When installing and configuring Instant Messaging Server:

- You must use a system user and group with specific privileges to run specific server processes. Normally, the **configure** utility creates the following users and groups:
  - User: **inetuser**
  - Group: **inetgroup**
- If the **configure** utility does not create a UNIX user and group for Instant Messaging, you need to create them manually. After you create the user and group for Instant Messaging Server, you must then set permissions appropriately for the directories and files owned by that user.
- Do not choose **root** as a server user ID.
- If you decide to enable TLS, the respective server configuration is mandatorily set to TLS for all communication.

Installing Instant Messaging Server

Follow the steps in *Instant Messaging Server Installation and Configuration Guide* to install Instant Messaging Server. Change the default port numbers as needed.

The installation prompts for authentication credentials for the following:

- Directory Server manager (bind DN and password)
- Web administrator for HTTP Gateway (user ID and password)

Post-Installation Tasks

After installation, configuring Instant Messaging Server for a secure deployment involves a number of potential steps:

1. About Transport Layer Security for Instant Messaging Server
2. Writing a Custom Single Sign-On Module
3. Controlling End User and Administrator Privileges

See "Implementing Instant Messaging Server Security" for more information.

3

Implementing Instant Messaging Server Security

This chapter explains the security features of Oracle Communications Instant Messaging Server.

About System Security in Instant Messaging Server

Security requirements arise from the need to protect data: first, from accidental loss and corruption, and second, from deliberate unauthorized attempts to access or alter that data. Secondary concerns include protecting against undue delays in accessing or using data, or even against interference to the point of denial of service. The global costs of such security breaches run up to billions of dollars annually, and the cost to individual companies can be severe, sometimes catastrophic.

The critical security features that provide these protections are:

- Authentication
- Access Control
- Secure Communications

Authentication is the way in which an entity (a user, an application, or a component) determines that another entity is who it claims to be. An entity uses security credentials to authenticate itself. The credentials might be a user name and password, a digital certificate, or something else. Usually, servers or applications require clients to authenticate themselves. Additionally, clients might require servers to authenticate themselves. When authentication is bidirectional, it is called *mutual authentication*.

Access Control, also known as authorization, is the means by which users are granted permission to access data or perform operations. After a user is authenticated, the user's level of authorization determines what operations the user can perform.

About Transport Layer Security for Instant Messaging Server

Instant Messaging Server supports Transport Layer Security (TLS) for secure communications. This section provides instructions for setting up security for Instant Messaging Server by using TLS.

Overview of Using TLS in Instant Messaging Server

Instant Messaging Server uses a startTLS extension to the TLS 1.0 protocol for client-to-server and server-to-server encrypted communications and for certificate-based authentication between servers.

Communication between multiplexor and server is over an unsecured transport. When you use TLS for client-to-server communication, the multiplexor simply passes the data from the client to the server and back and does not perform any encryption or decryption.

TLS is fully compatible with SSL and includes all necessary SSL functionality. TLS and SSL function as protocol layers beneath the application layers of XMPP and HTTP.

For Java requirements to use TLS with Instant Messaging Server, see the topic on required software in *Instant Messaging Server Installation and Configuration Guide*.

For information on TLS and StartTLS in XMPP, see Use of TLS in RFC 3920, *Extensible Messaging and Presence Protocol: Core*, at:

`http://www.ietf.org/rfc/rfc3920.txt`

For an overview of certificates, SSL, and TLS, see *Oracle GlassFish Server Administration Guide*. These procedures assume that you are using GlassFish Server to generate certificates.

Setting Up TLS for Instant Messaging Server

Enabling TLS for Instant Messaging Server server-to-server and client-to-server communication requires the following general steps:

1. Creating a Java *keystore* (JKS) and a private key by using the **keytool** utility.

   For an overview of the **keytool** utility, see "Tools for Managing Security" in *Sun GlassFish Enterprise Server v2.1.1 Administration Guide*. For instructions on generating the JKS by using GlassFish Server, see "Working with Certificates and SSL" in *Sun GlassFish Enterprise Server v2.1.1 Administration Guide*.

2. Using the private key to generate a server certificate for the Instant Messaging server.

   See "Generating a Certificate Using the keytool Utility" in *Sun GlassFish Enterprise Server v2.1.1 Administration Guide* for instructions.

3. Getting the Instant Messaging server certificate signed by a Certificate Authority (CA).

   See "Signing a Digital Certificate Using the keytool Utility" in *Sun GlassFish Enterprise Server v2.1.1 Administration Guide* for instructions. Replace GlassFish Server with Instant Messaging Server where applicable.

4. Restarting Instant Messaging Server.

   See *Instant Messaging Server System Administrator's Guide* for details.

5. Obtaining the CA's root certificate.

   Contact your CA for instructions on obtaining the CA's root certificate.

6. Importing the certificates into the keystore.

   You import the CA root certificate and the signed server certificate into the keystore by using the **keytool** utility as described in "Using the keytool Utility" in *Sun GlassFish Enterprise Server v2.1.1 Administration Guide*.

7. Activating TLS in the server by setting the appropriate configuration properties.

   For instructions see "Activating TLS on Instant Messaging Server."

8. For server-to-server communication over TLS, you need to repeat these steps for each server that communicates over TLS. You also do not need to configure the multiplexor for TLS.
9. Configuring the gateway to communicate directly with the Instant Messaging server and not the multiplexor, if you are using the XMPP/HTTP Gateway in your deployment.

If you are using GlassFish Server, steps 1 through 6 are documented in "Working with Certificates and SSL" in *Sun GlassFish Enterprise Server v2.1.1 Administration Guide*. Step 7 is described in "Activating TLS on Instant Messaging Server."

Activating TLS on Instant Messaging Server

Before you can activate TLS on the server, you must create a JKS, obtain and install a signed server certificate, and trust the CA's certificate as described in "Setting Up TLS for Instant Messaging Server." You activate TLS on the server when you want to use TLS for server-to-server and/or client-to-server communication.

Table 3–1 lists the configuration properties used to enable TLS in Instant Messaging Server. It also contains the description and the default values for these properties.

Table 3–1 Instant Messaging Server TLS Configuration Properties

Property	Default Value	Description
iim_server.sslkeystore	None	Contains the relative path and file name for the server's Java keystore (JKS). For example: *InstantMessaging_home*/**server-keystore.jks**
iim_server.keystorepasswordfile	**sslpassword.conf**	Contains the relative path and the name of the file that contains the password for the keystore. This file should contain the following line: `Internal (Software) Token:`*`password`* where *password* is the password protecting the keystore.
iim_server.requiressl	**false**	If this value is **true**, the server terminates any connection that does not request a TLS connection after the initial stream session is set up.
iim_server.trust_all_cert	**false**	If this value is **true**, the server trusts all certificates, including expired and self-signed certificates, and also adds the certificate information into the log files. If **false**, the server does not log certificate information and trusts only valid certificates signed by a CA.

To Activate TLS Communication in Instant Messaging Server

Use this procedure to configure Instant Messaging Server to use secure communication over TLS in the following ways:

- Require TLS for all client and server connections.
- Require TLS only for specific server-to-server connections.
- Allow TLS connections for clients and servers that request a secure transport after the initial communication session has been set up.
- A combination of requiring TLS for specific server-to-server connections and allowing TLS connections for other clients and servers.

Ensure that you have created a JKS, obtained and installed a server certificate, and configured the server to trust the CA's certificate as described in "Setting Up TLS for Instant Messaging Server."

For server-to-server TLS communication, you must complete this procedure on each server you want to configure to use TLS.

1. Set the **iim_server.sslkeystore** and **iim_server.keystorepasswordfile** configuration properties.

   For example:

   ```
   imconfutil -c /opt/sun/comms/im/config/iim.conf.xml set-prop iim_
   server.sslkeystore=/opt/sun/comms/im/config/server-keystore.jks

   iim_server.keystorepasswordfile=sslpassword.conf
   ```

   The server now responds to a connection request from any client or another Instant Messaging server with the information that it is able to communicate over TLS. The requesting client or server then chooses whether to establish a secure connection over TLS.

2. If you want the server to require TLS for all connections from clients, and remote and peer servers, add the **iim_server.requiressl=true** configuration property.

   For example:

   ```
   imconfutil -c /opt/sun/comms/im/config/iim.conf.xml set-prop iim_
   server.requiressl=true
   ```

   When you set this configuration property to **true**, the server terminates a connection with any client or remote or peer server that does not support TLS. Use this parameter to require secure client-server communication over TLS.

   For more information about server-to-server communication, see *Instant Messaging Server System Administrator's Guide*.

3. If you want to require TLS for communication with a specific remote or peer server, set the **requiressl** coserver property.

   For example:

   ```
   imconfutil -c /opt/sun/comms/im/config/iim.conf.xml set-coserver-prop coserver1
   requiressl=true
   ```

   Set this property for each coserver for which you want to require TLS.

   When you set **iim_server.requiressl** to **true**, the server requires a TLS connection for any server with which it communicates. In this case, you do not need to set this parameter for specific coservers.

4. (Optional) If you want the server to trust all certificates it receives, and to add certificate information to the log files, add the **iim_server.trust_all_cert=true** configuration property:

   For example:

   ```
   imconfutil -c /opt/sun/comms/im/config/iim.conf.xml set-prop iim_server.trust_
   all_cert=true
   ```

WARNING: You might need to use this feature to test your deployment before you go live. However, you typically should not do this on a deployed system as it presents severe security risks. When this value is true, the server trusts all certificates, including expired and self-signed certificates, and also adds the certificate information into the log files. When this value is false, the server does not log certificate information and trusts only valid certificates signed by a CA.

5. Refresh the server configuration by using the **imadmin** command.

   ```
   imadmin refresh server
   ```

6. Verify that TLS is working properly.

Enabling TLS Protocols

Instant Messaging Server disables SSLv3 support by default. However, to allow or restrict TLS protocols, you can set the **iim_server.tls.enabledprotocols** configuration property at default and listener levels.

To enable TLS protocols in the Instant Messaging Server:

```
imconfutil -c InstantMessaging_home/config/iim.conf.xml -u set-prop iim_
server.tls.enabledprotocols="TLSv1,TLSv1.1,TLSv1.2"
```

To enable TLS protocols in **c2s** and **s2s** listeners:

```
imconfutil -c InstantMessaging_home/config/iim.conf.xml -u set-listener-prop s2s
iim_server.tls.enabledprotocols="TLSv1.2"
imconfutil -c InstantMessaging_home/config/iim.conf.xml -u set-listener-prop c2s
iim_server.tlsenabledprotocols="SSLv3,SSLv2Hello,TLSv1,TLSv1.1,TLSv1.2"
```

To enable TLS protocols in the SFS Gateway:

1. Configure the **iim_server.tls.enabledprotoocls** property:

   ```
   imconfutil -c InstantMessaging_home/config/iim.conf.xml -u set-prop iim_
   server.tls.enabledprotocols="TLSv1,TLSv1.1,TLSv1.2"
   ```

2. Create the war file by running the **create_sip_war** command.

   For more information on the **create_sip_war** command, see the "Configuring the SIP Gateway" chapter in *Instant Messaging Server System Administrator's Guide*.

Encrypting Passwords

You can use the **passwordtool** command to encrypt a password for either the Instant Messaging HTTPBIND Web component or the Web Presence API and use the encrypted password in the component's configuration file. For more information, see the **passwordtool** reference information in *Instant Messaging Server System Administrator's Guide*.

Writing a Custom Single Sign-On Module

This information describes how to write a custom single sign-on (SSO) module for Instant Messaging Server so that SSO support can be added for third-party SSO solutions.

As with any SSO aware application, when a user is authenticated, Instant Messaging Server loads the authentication module to validate the user. On successful validation, the user is allowed to access the application. If the validation is not successful, the standard password validation occurs.

API Reference Documentation

The SSOProvider API documentation is provided as part of the Instant Messaging Server installation. You can find the documentation in *InstantMessaging_home*/**html/apidoc**, which, by default, is **/opt/sun/comms/im/html/apidoc.**

How the Custom SSO Module for Instant Messaging Server Works

Instant Messaging Server calls the verify method provided by the custom SSO provider with the uid and token arguments.

The token argument is the password provided by the Instant Messaging client during the SASL PLAIN authentication. For example the client has sent the following:

```
<body rid='607740' sid='7675823097240743042'
xmlns='http://jabber.org/protocol/httpbind'
key='c654f46426c6d12cf2a0e1a9beb218915e0afd6f'  ><auth
xmlns='urn:ietf:params:xml:ns:xmpp-sasl'
mechanism='PLAIN'>c2hqb3J0aEBhdS5vcmFjbGUuY29tAHNoam9ydGgAc2VjcmV0dG9rZW4=</auth><
/body>
```

`c2hqb3J0aEBhdS5vcmFjbGUuY29tAHNoam9ydGgAc2VjcmV0dG9rZW4=` is the base64 encoded SASL PLAIN string as per the XEP-0034 standard. This string decodes to:

```
shj@example.com<NUL>shjorth<NUL>secrettoken
```

where:

shj@example.com is the authorization identity

shj is the authentication identity

secrettoken is the password (or in this case the SSO token) provided by the IM client

<NUL> is the **NUL** character

Implementing the Custom SSO Module

Before designing a solution for the custom SSO module, the Instant Messaging Server SSO provider framework needs to be implemented:

- All custom SSO modules must implement SSOProvider interface.
- The SSO verification implementation must provide the domain of the authenticating user if **iim.userprops.store** is set to **ldap**.
- The SSO implementation can use any other classes that are required to make the custom SSO module work.

To implement a custom SSO module:

1. Create the custom SSO Java file.

```
mkdir -p com/client/sample/
```

2. Use the following sample code to create the file.

```
com/client/sample/CustomSSOProvider.java
import java.util.Map;
import com.iplanet.im.server.RealmManager;
import com.iplanet.im.server.LocalUser;
import com.iplanet.im.server.LDAPUserSettings;
import com.iplanet.im.common.util.Log;

public class CustomSSOProvider implements SSOProvider {

   // This is called for each authentication. It can return true or false.
   public boolean verify(String uid, String token, java.util.Map attributes,
java.util.Set attributeNames){

     String domain = "your.domain.com";
     Log.debug(String.format("CustomSSO: Trying to authenticate user: %s, token
= %s\n", uid, token));

     // Replace the following check with your SSO token verification routine
     if (token.equalsIgnoreCase("secrettoken") == false) {
       Log.debug("CustomSSO: 'secrettoken' not provided as password, SSO login
attempt failed");
       return false;
     }

     // If user properties are stored in LDAP then need to retrieve them
     if (RealmManager.getUserSettingsStorageProvider() instanceof
LDAPUserSettings) {
       Log.debug("CustomSSO: iim.userprops.store = \"ldap\"");

       try{
         LocalUser u = RealmManager.getUser(uid, domain, true);

         // the domain is mandatory if using a hosted domain configuration
         // the boolean third parameter forces an LDAP fetch
         // (disregarding previously cached entries)
         if(u != null){
           for(Object o: attributeNames){
             String s = (String) o;
             Log.debug("CustomSSO: attributeName: " + s);
             Set prop = u.getAttributeValues(s);
             attributes.put(s, prop);
           }
           return true;
         }
       }
       catch(RealmException ex){
         Log.error("CustomSSO: could not load user: " + uid + " in domain: " +
domain);
         return false;
       }
     }
     else {
       Log.debug("CustomSSO: iim.userprops.store = \"file\"");
       return true;
     }
     return false;
   }
```

```
    // This is called each time there's some XMPP activity by the user. The
    // SSO implementation can use this to keep the session from timing out.
    public boolean refresh(String uid) {
       Log.debug("CustomSSO: refresh called " + uid);
       return true;
    }

    // This is for any SSO initialization and is called once on server startup
    public void open() throws Exception {
       Log.debug("CustomSSO: open called");
    }

    // this is called before the server is shutdown, though its not guaranteed
    public void close() {
       Log.debug("CustomSSO: close called");
    }
}
```

3. Compile the source.

   Make sure to compile with JDK1.6. This example uses the JDK installed under **/usr/jdk/latest** on Solaris OS and Instant Messaging Server under **/opt/sun/comms/im**.

   ```
   /usr/jdk/latest/bin/javac -classpath
   /usr/share/lib/xmpp/improvider.jar:/opt/sun/comms/im/lib/xmppd.jar:/opt/sun/com
   ms/im/lib/imcommon.jar:/opt/sun/comms/im/lib/imnet.jar
   com/client/sample/CustomSSOProvider.java
   ```

4. Create the jar archive from the compiled files.

   ```
   /usr/jdk/latest/bin/jar -cvf CustomSSOProvider.jar
   com/client/sample/CustomSSOProvider.class
   ```

5. Move the compiled jar file to the Instant Messaging Server library directory.

   ```
   chown bin:bin CustomSSOProvider.jar
   cp -p CustomSSOProvider.jar /opt/sun/comms/im/lib
   ```

6. Add the jar file to the **imadmin** command.

   ```
   cp imadmin imadmin.orig
   ```

   Edit the **imadmin** command and add the following to the end of the **server_classpath** variable setting:

   ```
   $PCD/CustomSSOProvider.jar
   ```

7. Run the **imconfutil** command with the following properties to enable the provider.

   ```
   iim_server.usesso = "1"
   iim_server.ssoprovider = "com.client.sample.CustomSSOProvider"
   ```

8. Restart the XMPP server.

   ```
   cd /opt/sun/comms/im/
   ./imadmin stop
   ./imadmin start
   ```

9. You should now see the following in the **xmppd.log** file (when debug log-level is enabled in the **log4j.conf** file) when logging in by using the password **secrettoken**:

```
[13 Sep 2010 08:53:34,857] DEBUG xmppd [Thread-15] CustomSSO: Trying to
authenticate user: shjorth, token = secrettoken
[13 Sep 2010 08:53:34,857] DEBUG xmppd [Thread-15] CustomSSO:
iim.userprops.store = "ldap"
[13 Sep 2010 08:53:34,898] DEBUG xmppd [Thread-15] CustomSSO: attributeName:
uid
[13 Sep 2010 08:53:34,898] DEBUG xmppd [Thread-15] CustomSSO: attributeName:
uniquemember
[13 Sep 2010 08:53:34,898] DEBUG xmppd [Thread-15] CustomSSO: attributeName:
givenname
[13 Sep 2010 08:53:34,898] DEBUG xmppd [Thread-15] CustomSSO: attributeName:
sunPresenceAccessPermitted
[13 Sep 2010 08:53:34,898] DEBUG xmppd [Thread-15] CustomSSO: attributeName:
sunIMUserNewsRoster
[13 Sep 2010 08:53:34,898] DEBUG xmppd [Thread-15] CustomSSO: attributeName:
sunIMUserConferenceRoster
[13 Sep 2010 08:53:34,898] DEBUG xmppd [Thread-15] CustomSSO: attributeName:
userpassword
[13 Sep 2010 08:53:34,898] DEBUG xmppd [Thread-15] CustomSSO: attributeName:
sunIMRoster
[13 Sep 2010 08:53:34,898] DEBUG xmppd [Thread-15] CustomSSO: attributeName:
sunIMConferenceRoster
[13 Sep 2010 08:53:34,898] DEBUG xmppd [Thread-15] CustomSSO: attributeName:
sunIMNewsRoster
[13 Sep 2010 08:53:34,898] DEBUG xmppd [Thread-15] CustomSSO: attributeName:
sunIMUserProperties
[13 Sep 2010 08:53:34,898] DEBUG xmppd [Thread-15] CustomSSO: attributeName:
sunPresenceEntityDefaultAccess
...
```

Troubleshooting the Custom SSO Module

For Solaris OS installations of Instant Messaging Server, check the following file for errors when starting the Instant Messaging server:

```
/var/svc/log/application-sunim:default.log
```

Writing a Custom SASL Module

This information describes how to write a custom Simple Authentication and Security Layer (SASL) module for Instant Messaging Server to authenticate clients using custom authentication mechanism. For example, you can implement Kerberos or OAuth-based authentication by using a custom SASL module.

Instant Messaging Server supports authenticating clients by using both default (PLAIN) and custom SASL mechanism. Upon successful authentication, the server sends a SASL success response to the client and allows communication with other clients.

API Reference Documentation

The SASL Provider API documentation is provided as part of the Instant Messaging Server installation. The documentation is located in the *InstantMessaging_home*/**html/apidoc** directory.

How the Custom SASL Module for Instant Messaging Server Works

Instant Messaging Server provides a framework for authenticating clients by means of the SASL protocol. SASL provides a generalized method for adding authentication support to any Instant Messaging clients. The protocol event flow is as follows (For more detail refer to XMPP core RFC3920):

The following high-level steps explain how SASL works:

1. The server informs the client of the available authentication mechanisms:

```
<features xmlns='http://etherx.jabber.org/streams'>
<mechanisms xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
<mechanism>PLAIN</mechanism>
<mechanism>CustomSASL</mechanism>
</mechanisms>
</features>
```

2. The client selects an authentication mechanism:

```
<auth xmlns='urn:ietf:params:xml:ns:xmpp-sasl'
     mechanism='CustomSASL'/>
```

3. The server sends a nonce as a BASE64 encoded challenge to client:

```
<challenge
xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>T0E2TUc5dEVRR20yaGg=</challenge>
```

   The decoded challenge is:

```
OA6MG9tEQGm2hh
```

4. The client sends a BASE64 encoded response to the challenge:

```
<response xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
T0E2TUc5dEVRR20yaGh1c2VyaWQ=
</response>
```

   The decoded response is

```
OA6MG9tEQGm2hhuserid
```

5. The server decodes the userId which is appended with challenge, and validates the user. If the user exists, it informs the client of a successful authentication:

```
<success xmlns='urn:ietf:params:xml:ns:xmpp-sasl'/>
```

The sample can be used to implement any custom authentication including Single Sign-On. For example, you can use an SSO-based token instead of userID in Step 4.

Implementing the Custom SASL Module

To develop a custom SASL module, you must implement the Instant Messaging Server SASL provider framework:

1. Use the following sample code to develop **SampleSASLProviderFactory**. The mechanism in the sample is **CustomSASL**.

```
public class SampleSASLProviderFactory implements SASLServerProviderFactory {
    @Override
    public String[] getSupportedMechanisms() {
        return new String[] {"CustomSASL"};
    }
    @Override
```

```
    public SASLServerProvider createInstance(String mechanism) {
        return new SampleSASLProvider();
    }
}
```

2. The **SampleSASLProvider** implements **SASLServerProvider**. It checks if the received response contains the nonce sent suffixed with the user, and sends authentication success. The SASL challenge and responses should be modified to suit your use.

```
public class SampleSASLProvider implements SASLServerProvider {
    private CollaborationPrincipal user = null;
    private String domain = null;
    private Map authProperties = new HashMap(2);
    private int reqCount;
    private String[] digestSchemas;
    private Realm realm;
    private boolean failure = false;
    private String nonce;
    @Override
    public void init(Map properties) throws SASLProviderException {
        domain = (String)properties.get(SASLServerProvider.DOMAIN);
        // Supported digest schemas.
        digestSchemas = new String[1];
        digestSchemas[0] = "CustomSASL";
    }
    // Called for START - Send a challenge string
    private void handleAuthRequest(SASLData data) throws Exception {
        Long nonceL = new Random().nextLong();
        nonce = String.valueOf(nonceL);
        data.setResponseData(nonce.getBytes("UTF-8"));
        data.setResponseStatus(SASLData.CHALLENGE);
    }

    // Called for RESPONSE - Validate response, send success or another
challenge here
    private void handleResponse(SASLData data) throws Exception {
        byte [] pktData = data.getRequestData();
        String digestString;
        digestString = new String (pktData, "UTF-8");
        if (digestString.startsWith(nonce)) {
            realm = RealmManager.getRealm();
            String username = digestString.substring(nonce.length());
            user = realm.getPrincipal(realm.getSearchBase(domain), username);
            data.setResponseData(null);
            data.setResponseStatus(SASLData.SUCCESS);
            authProperties.put(USER , user);
            return;
        }
        data.setResponseStatus(SASLData.FAILURE);
        data.setResponseData(null);
        failure = true;
    }
    @Override
    public Map getProperties() throws AuthenticationException {
        return authProperties;
    }

    @Override
    public void close() {
```

```
            // Do nothing
        }
        // This method gets called for every SASL packet (start/response) received
        @Override
        public void process(SASLData data) throws SASLProviderException {
            if (failure) {
                throw new SASLProviderException("auth failed");
            }
            if (SASLData.FAILURE == data.getRequestStatus() || SASLData.ABORT ==
    data.getRequestStatus() ) {
                data.setResponseStatus(SASLData.ABORT);
                data.setResponseData(null);
                failure = true;
                return;
            }
            if (null == data.getRequestData() ) {
                throw new SASLProviderException("Has no data");
            }
            if ( (reqCount == 0 && SASLData.START != data.getRequestStatus() ) ||
                 (reqCount == 1 && SASLData.RESPONSE != data.getRequestStatus()) ||
                 (reqCount == 2) ) {
                data.setResponseStatus(SASLData.ABORT);
                data.setResponseData(null);
                failure = true;
                return ;
            }
            try {
                if (SASLData.START == data.getRequestStatus()) {
                    reqCount = 1;
                    handleAuthRequest(data);
                }

                if (SASLData.RESPONSE == data.getRequestStatus()) {
                    reqCount = 2;
                    handleResponse(data);
                }
            } catch (Exception e) {
                failure = true;
                throw new SASLProviderException(e);
            }
        }
    }
```

3. Run the following command to compile the Java files:

```
javac -cp
".:$SHAREDLIB/imservice.jar:$SHAREDLIB/improvider.jar:$IMBASEDIR/lib/xmppd.jar"
SampleSASLProvider.java SampleSASLProviderFactory.java
```

where:

- *$IMBASEDIR* is **/opt/sun/comms/im/** in Solaris, and **/opt/sun/im/** in Linux
- *$SHAREDLIB* is **/usr/share/lib** in Solaris, and **/opt/sun/share/lib** in Linux.

4. Create a JAR file out of the generated classes and add it to the server classpath:

```
jar -cf /tmp/CustomSASL.jar *.class
imconfutil -c $IMBASEDIR/config/iim.conf.xml set-prop
iim_server.classpath=/tmp/CustomSASL.jar
imconfutil -c $IMBASEDIR/config/iim.conf.xml set-prop iim_
ldap.sasl.mechanism.factories=com.sun.im.provider.sample.SampleSASLProviderFact
ory
```

5. Restart Instant Messaging Server.

Controlling End User and Administrator Privileges

Different sites using Instant Messaging Server have different needs in terms of enabling and restricting the type of access end users have to the Instant Messaging service. The process of controlling end user and administrator Instant Messaging Server features and privileges is referred to as policy management. You administer policy management through access file controls. The access control file method for managing policies enables you to adjust end-user privileges for conference room management and the ability to change user preferences. It also enables specific end users to be assigned as system administrators.

Policy Configuration Properties

Table 3–2 lists Instant Messaging Server policy configuration properties.

Table 3–2 Policy Configuration Properties

Property	Use	Values
iim.policy.modules	Indicates the schema to be used by Instant Messaging Server.	**iim_ldap** (default), **iim_ldap_ schema1**, or **iim_ldap_schema2**
iim.userprops.store	Indicates whether the user properties are in a user properties file or stored in LDAP. Only significant when the service definitions for the Presence and Instant Messaging services have been installed.	One of the following: ■ **ldap** (the default) ■ **file** (not recommended)

Setting the Policy Management Method

The **iim.policy.modules** configuration property identifies **iim_ldap** for the access control file method, which is also the default.

To set the policy management method:

1. Use the **imconfutil** command to set the **iim.policy.modules** configuration property to **iim_ldap** (default, the access control file method).
2. Use the **imconfutil** command to set the **iim.userprops.store** configuration property to one of the following:
   - **ldap** (default, for storing user properties in LDAP)

     If you choose **ldap**, you can run **imadmin assign_services** to add the required object classes that store user properties to user entries in the directory.
   - **file** (for storing user properties in files, no longer recommended.)
3. Refresh the configuration.

Managing Policies by Using Access Control Files

By editing access control files you control the following end-user privileges:

- Access to the presence status of the other end users
- Save properties on the server

- Create new conference rooms

By default, end users are provided the privileges to access the presence status of other end users and save properties to the server. For most deployments, default values do not need to be changed.

Although certain privileges can be set globally, the administrator can also define exceptions for these privileges. For example, the administrator can deny certain default privileges to select end users or groups.

In addition, if you are enforcing policy through access control files in your deployment, those files must be the same for all servers in a server pool.

Table 3–3 lists the global access control files for Instant Messaging Server and the privileges these files provide end users.

Table 3–3 Access Control Files

ACL File	Privileges
sysSaveUserSettings.acl	Defines who can and cannot change their own preferences. Users who do not have this privilege cannot add contacts, create conferences, and so on.
sysRoomsAdd.acl	Defines who can and cannot create Conference rooms.
sysWatch.acl	Defines who can and cannot watch changes of other end users.
sysAdmin.acl	Reserved for administrators only. This file sets administrative privileges to all Instant Messaging Server features for all end users. This privilege overrides all the other privileges and gives the administrator the ability to create and manage conference rooms as well as access to end user presence information, settings, and properties.

Changing End-user Privileges in Access Control Files

To change end-user privileges in access control files:

1. Change to the *InstantMessaging_cfg_home*/**acls** directory.

   See the configuration and file directory structure content in the *Instant Messaging Server System Administrator's Guide* for information on locating *InstantMessaging_cfg_home*.

2. Edit the appropriate access control file.

   For example:

   ```
   vi sysRoomsAdd.acl
   ```

   See "Access Control File Location" for a list of access control files.

3. Save the changes.

4. End users need to refresh Instant Messaging client to see the changes.

Using Access Control Files in a Server Pool

If you are enforcing policy through access control files in your deployment, the content of the files must be the same for all servers in a server pool. To ensure this, copy the files from one server to each of the other nodes in the pool. See "Access Control File Location" for information on finding these files.

Access Control File Location

The location of the access control files is *InstantMessaging_cfg_home***/acls**. See the configuration and file directory structure content in the *Instant Messaging Server System Administrator's Guide* for information about the default location of the configuration directory.

Access Control File Format

The access control file contains a series of entries that define the privileges. Each entry starts with a tag as follows:

- **d:** - default
- **u:** - user
- **g:** - group

The tag is followed by a colon (**:**). In case of the default tag it is followed by **true** or **false**.

End-user and group tags are followed by the end-user or group name.

Multiple end users and groups are specified by having multiple end users (**u**) and groups (**g**) in lines.

The **d:** tag must be the last entry in an access control file. The server ignores all entries after a **d:** tag. If the d: tag is true, all other entries in the file are redundant and are ignored. You cannot set the **d:** tag as true in an access control file and selectively disallow end users that privilege. If default is set to false, only the end users and groups specified in the file will have that particular privilege.

The following are the default **d:** tag entries in the ACL files for a new installation:

- **sysAdmin.acl** - Contains **d:false**
- **sysRoomsAdd.acl** - Contains **d:true**
- **sysSaveUserSettings.acl** - Contains **d:true**
- **sysWatch.acl** - Contains **d:true**

A

Secure Deployment Checklist

The following security checklist provides guidelines to help you secure Oracle Communications Instant Messaging Server and its components.

Secure Deployment Checklist

- Install only the components you require.
- Lock and expire default user accounts.
- Use a strong LDAP password policy for user authentication.
- Enable data dictionary protection on the Oracle Database for Instant Messaging Server.
- Restrict, control, and revisit user privileges:
  - Grant only the necessary privileges to each user.
  - Revoke unnecessary privileges from the PUBLIC user group.
  - Restrict permissions on run-time facilities.
- Enforce the use of access controls by using the Authorization Policies.
- Require clients to authenticate.
- Restrict network access by doing the following:
  - Use firewalls.
  - Never leave an unnecessary hole in a firewall.
  - Password-protect the Oracle listener against remote access.
  - Monitor listener activity.
  - Monitor who accesses your systems.
  - Restrict system access by IP addresses.
  - Encrypt network traffic.
- Apply all security patches and workarounds.
- Encrypt sensitive information.
- Contact Oracle Security Products if you discover a vulnerability in any Oracle product.