Oracle Service Bus Behind the Firewall in a Service-Oriented Architecture

An Oracle White Paper
Updated October 2008

ORACLE

Oracle Service Bus Behind the Firewall in a Service-Oriented Architecture

Oracle Service Bus Behind the Firewall in a Service-Oriented Architecture

INTRODUCTION

As organizations move to service-oriented architecture (SOA), security becomes one of the key concerns impacting deployment. A company's sensitive information is accessed by services deployed on the distributed components in a SOA. Therefore, security concerns have become part of the enterprise decision-making process related to the adoption of an SOA.

Typically, a company sends request messages from Oracle Service Bus (formerly BEA AquaLogic Service Bus) behind its internal firewall to business services hosted outside its firewall. In such a scenario, Oracle Service Bus acts as a forward proxy. The company receives response messages from business services through a demilitarized zone into Oracle Service Bus; in this case, Oracle Service Bus acts as a reverse proxy.

This white paper discusses the security setup and configuration for clients, Oracle Service Bus (version 2.0 or later) proxy services, and business services. The setup assumes that a client Web application sends a HyperText Transfer Protocol request message from outside a company's firewall to the Oracle Service Bus server located behind its firewall (the inbound request).

This white paper discusses the security setup and configuration for clients, Oracle Service Bus (version 2.0 or later) proxy services, and business services. The setup assumes that a client Web application sends a HyperText Transfer Protocol (HTTPS) request message from outside a company's firewall to the Oracle Service Bus server located behind its firewall (the inbound request). Oracle Service Bus then routes the HTTPS request message to a business service hosted outside its firewall (the outbound request). The business service sends the response message through Oracle Service Bus to the client. This setup involves an inbound one- or two-way Secure Sockets Layer (SSL) authentication and an outbound one- or two-way SSL authentication. It capitalizes on Oracle WebLogic Server and Oracle Service Bus security.

Included in this white paper is an example of how to configure the inbound two-way and outbound two-way SSL authentication from the command line, the Oracle WebLogic Server administration console, and the Oracle Service Bus console. The example includes a description of the deployment architecture and how to test the system with request and response messaging.

CONTEXT AND BACKGROUND

Defining Terms

Firewall

A firewall limits traffic between two networks. Firewalls can be a combination of software and hardware, including routers and dedicated gateway machines. They employ filters that allow or disallow traffic to pass based on the transport protocol, service requested, routing information, and origin and destination hosts or networks. They might also allow access for authenticated users.

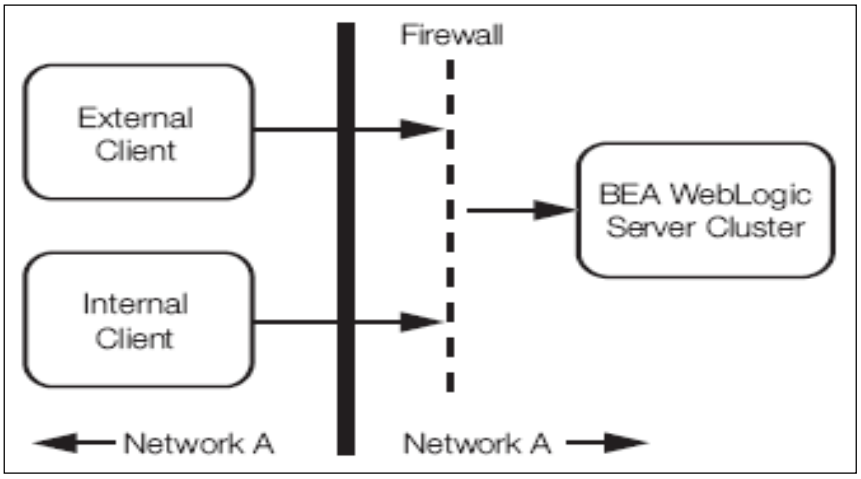

Figure 1: A typical setup for an organization with a firewall that filters the traffic destined for an Oracle WebLogic Server cluster.

Demilitarized Zone

A demilitarized zone (DMZ) is a network area that sits between an organization's internal network and an external network, usually the internet. The typical approach is to create a DMZ between two firewalls. The network DMZ must be monitored carefully, because sensitive objects are exposed to higher risk than services behind the firewall.

It is important to carefully control administrative access to services on the DMZ. Most likely, access should be allowed only from the internal network and preferably over a cryptographically protected connection such as Secure Shell.

A DMZ is an example of defense in depth: multiple layers of security provide a better shield than a simple firewall. If attackers penetrate the first firewall, they gain access to the DMZ, but not necessarily to the internal network.

A DMZ is an example of defense in depth: multiple layers of security provide a better shield than a simple firewall. If attackers penetrate the first firewall, they gain access to the DMZ, but not necessarily to the internal network.[1]

[1] For a full description of DMZs, refer to *Firewalls and Internet Security: Repelling the Wily Hacker* (Addison-Wesley Professional, 2003).

Secure Sockets Layer

The Secure Sockets Layer enables secure communication between applications connected through the Web. An SSL connection begins with a handshake during which the applications exchange digital certificates, agree on the encryption algorithms to be used, and generate the encryption keys to be used for the remainder of the session.

The Secure Sockets Layer (SSL) enables secure communication between applications connected through the Web.[2]

Oracle WebLogic Server uses the Certicom SSL Plus Java version 4.0 SSL implementation. The RSA BSAFE Cert-J and Crypto-J have been upgraded to Cert-J version 2.1.1 and Crypto-J version 3.5.

Network Interface Card

A network interface card (also called a network adapter or network card) is a piece of computer hardware designed to enable computers to communicate over a computer network.

Oracle WebLogic Server Support for One-Way and Two-Way SSL

Oracle WebLogic Server provides a pure-Java implementation of SSL. Generally, SSL provides

- A mechanism that the communicating applications can use to authenticate each other's identity
- Encryption of the data exchanged by the applications

When SSL is used, the target (the server) always authenticates itself to the initiator (the client). Optionally, if the target requests it, the initiator can authenticate itself to the target. Encryption makes data transmitted over the network intelligible only to the intended recipient. An SSL connection begins with a "handshake," in which the applications exchange digital certificates, agree on the encryption algorithms to be used, and generate the encryption keys to be used for the remainder of the session.

SSL Security Features

SSL provides the following security features:

- **Server authentication.** Oracle WebLogic Server uses its digital certificate, issued by a trusted certificate authority (CA), to authenticate to clients. SSL minimally requires the server to authenticate to the client using its digital certificate. If the client is not required to present a digital certificate, the connection type is called one-way SSL authentication.
- **Client identity verification.** Optionally, clients can be required to present their digital certificates to Oracle WebLogic Server, which then verifies that the digital certificate was issued by a trusted CA and establishes the SSL connection. An SSL connection is not established if the digital certificate is not presented and verified. This type of connection is called two-way SSL authentication, a form of mutual authentication.

---

[2] For a discussion of the components of SSL communication and why each component is necessary, refer to "OpenSSL Documents" at http://www.openssl.org/docs/ and *SSL and TLS: Designing and Building Secure Systems* (Addison-Wesley, 2001).

- **Confidentiality.** All client requests and server responses are encrypted to maintain the confidentiality of data exchanged over the network.
- **Data integrity.** Each SSL message contains a message digest computed from the original data. On the receiving end, a new digest is computed from the decrypted data, and then compared with the digest that came with the message. If the data has been altered, the digests do not match and tampering is detected.
- **Protection.** Data that flows between a client and Oracle WebLogic Server is protected from tampering by a third-party validation of user identities.
- **Web format.** If you are using a Web browser to communicate with Oracle WebLogic Server, you can use HTTP over SSL (HTTPS) to secure network communications.

SSL Tunneling

Oracle WebLogic Server supports tunneling with HTTP, T3, and Internet Inter-Orb Protocol (IIOP) over SSL. SSL can be used by Web browsers and Java clients as follows:

- A Web browser makes an SSL connection to a server over HTTPS. The browser then sends HTTP requests and receives HTTP responses over this SSL connection, for example, https://myserver.com/mypage.html.
- Oracle WebLogic Server supports SSL versioning. This means it can communicate with any client, including Web browsers that use SSL v2, SSL v3, and Transport Layer Security v1.
- Java clients using HTTP and T3 protocols can tunnel over SSL, for example, t3s://myserver.com:7002/mypage.html.
- Java clients running on Oracle WebLogic Server can establish either T3 connections to other instances of Oracle WebLogic Server or HTTPS connections to other servers that support SSL, such as Web servers or secure proxy servers.

One-Way and Two-Way SSL Authentication

Oracle WebLogic Server supports both one-way and two-way SSL authentication.

Oracle WebLogic Server supports both one-way and two-way SSL authentication. With one-way SSL authentication, the target (the server) is required to present a digital certificate to the initiator (the client) to prove its identity. The client performs two checks to validate the digital certificate.

- The client verifies that the certificate is trusted (it was issued by the client's trusted CA), is valid (not expired), and satisfies other certificate constraints.
- The client checks that the certificate subject's common name field value matches the host name of the server to which the client is trying to connect. This is called host name verification.

- Advanced options include inbound and outbound certificate lookup and validation (CLV). This feature provides additional protection by validating the certificate against the list of CAs compiled by the server administrator. The Oracle WebLogic Server security service provides the CLV API that finds and validates X.509 certificate chains.

A CertPath is a Java Development Kit (JDK) class that stores a certificate chain in memory. The term *CertPath* is also used to refer to the JDK architecture and framework that is used to locate and validate certificate chains. The CLV framework extends and completes the JDK CertPath functionality. CertPath providers rely on a tightly coupled integration of Oracle WebLogic Server and JDK interfaces. Your application code can use the default CertPath providers furnished by Oracle WebLogic Server to build and validate certificate chains, or it can use any custom CertPath provider.

If all the above checks return true, the SSL connection is established.

With two-way SSL authentication, both the client and the server must present digital certificates before the SSL connection is enabled between the two.

With two-way SSL authentication, both the client and the server must present digital certificates before the SSL connection is enabled between the two. Thus, in this case, Oracle WebLogic Server not only authenticates itself to the client (which is the minimum requirement for certificate authentication), but it also requires authentication from the requesting client.

Two-way SSL authentication is useful when you must restrict access to trusted clients only. Oracle WebLogic Server SSL connections support one-way SSL, two-way SSL, or both. The Web browser client, Web server, fat client, Web services client, and SSL server connections can be configured for either one-way or two-way SSL. Oracle WebLogic Server determines whether an SSL connection is configured for one-way or two-way authentication, and SSL is configured using the Oracle WebLogic Server administration console.

Host Name Verification

Host name verification is the process of verifying that the name of the host to which an SSL connection is made is the intended or authorized party. Host name verification prevents man-in-the-middle attacks, in which a client requests an SSL connection to a remote application server.

By default, the SSL client, as a function of the SSL handshake, compares the common name in the SubjectDN of the SSL server's digital certificate with the host name of the SSL server to which it is trying to connect. If these names do not match, the SSL connection is dropped.

Trust Manager

The trust manager provides a way to override the default SSL trust validation rules. It allows the server to decide whether it trusts the client that is contacting it. Using a trust manager, you can perform custom checks before continuing an SSL connection. For example, you can use the trust manager to specify that only users

from specific localities (such as towns, states, and countries) or users with other special attributes can gain access via the SSL connection.

Oracle WebLogic Server provides the weblogic.security.SSL.TrustManager interface. This interface enables custom trust manager implementations to be called during the SSL handshake. The custom implementation can override the handshake error detected by the SSL implementation validation check or indicate an error based on its own certification rules.

Oracle WebLogic Server also provides the weblogic.security.SSL.CertPath.TrustManager interface,[3] which applications and custom code can use to control outbound SSL certificate lookup and validation.

Asymmetric Key Algorithm

Asymmetric key (also referred to as public key) algorithms are implemented through a pair of different but mathematically related keys: a public key and a private key.

Asymmetric key (also referred to as public key) algorithms are implemented through a pair of different but mathematically related keys: a public key and a private key.

The public key (which is widely distributed) is used for verifying a digital signature or transforming data into a seemingly unintelligible form. The private key (which is always kept secret) is used for creating a digital signature or returning the data to its original form.

The public key infrastructure (PKI) in Oracle WebLogic Server also supports digital signature algorithms. Digital signature algorithms are simply public key algorithms used to generate digital signatures. Oracle WebLogic Server supports the Rivest, Shamir, and Adelman (RSA) algorithm.

PROBLEM DEFINITION

The focus of the rest of this white paper is on outlining how to establish two-way inbound and two-way outbound SSL authentication among a client, Oracle Service Bus, and business services separated by firewalls. The goal is to establish inbound two-way and outbound two-way SSL authentication for a scenario in which

- A client sends an HTTPS request to an Oracle Service Bus proxy service
- An Oracle Service Bus proxy service routes the request to an Oracle WebLogic Server–hosted business service over HTTPS
- A business service receives a request and sends an HTTPS response to the client via the Oracle Service Bus proxy service

DEPLOYMENT ARCHITECTURE, SETUP, AND CONFIGURATION

In general, the details of the installation and configuration depend on the operating system, the number of machines, and the desired cluster sizes of Oracle WebLogic

[3] The weblogic.security.SSL.CertPath.TrustManager interface replaces the weblogic.security.SSL.TrustManagerJSSE interface, which is superseded in Oracle WebLogic Server 9.1.

Server domains. Typically in a production environment, the client, the Oracle Service Bus proxy service, and the business service are hosted on separate machines. In addition, each Oracle WebLogic Server cluster node is hosted on a separate hardware system.

The following section describes a simplified example of how to set up and configure request and response messaging among a client, an Oracle Service Bus proxy service, and a business service for testing.

The following section describes a simplified example of how to set up and configure request and response messaging among a client, an Oracle Service Bus proxy service, and a business service for testing. The example uses minimum hardware consisting of two systems, each running a Microsoft Windows operating system. The first machine hosts the client and business service. The second machine hosts the Oracle Service Bus proxy service.

The first and second machines are separated by two firewalls, with a lab network between firewalls. The lab network simulates a public network, like the internet. This is a simplification of a typical setup in which each company—hosting the client, the Oracle Service Bus proxy service, or the business service—has two firewalls with a DMZ in between. The example outlines common configuration steps, which can be extrapolated to suit production requirements.

Figure 2 is an example of a deployment architecture, which provides context for the ensuing conversation.

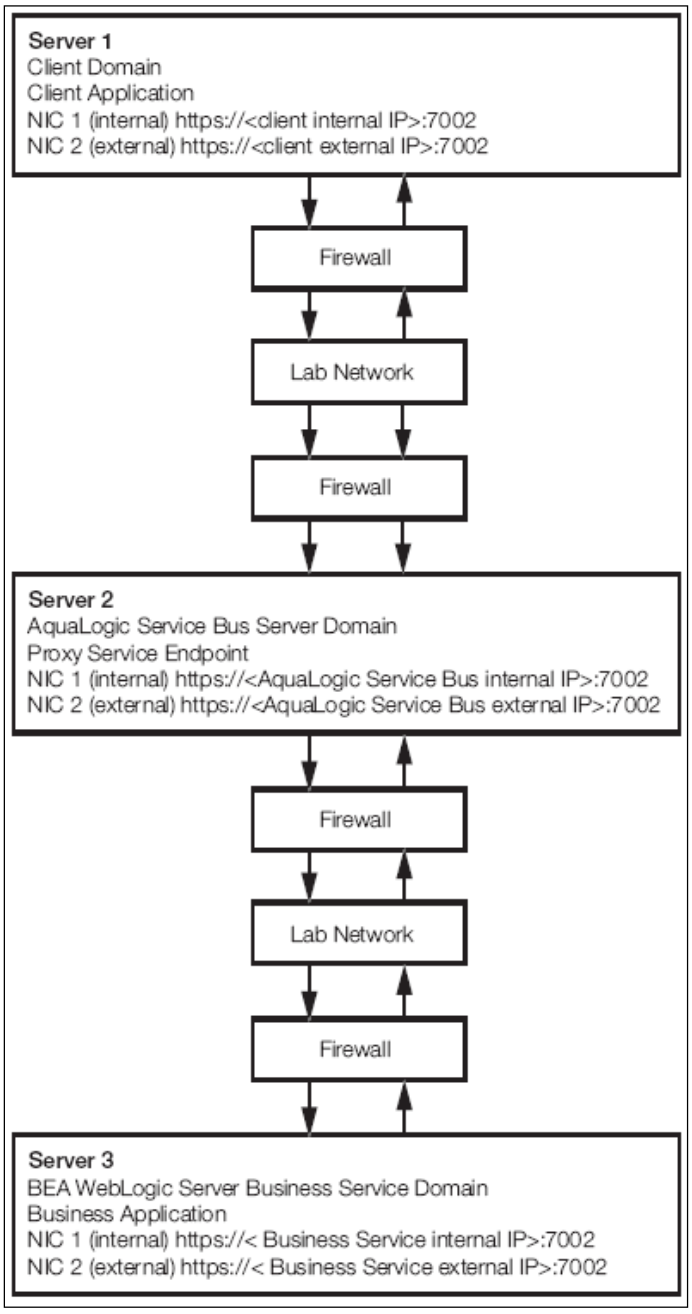

Figure 2: A sample deployment configuration for a client with an Oracle Service Bus proxy service and a business service; DMZ not shown.

Although Figure 2 does not show the DMZ, Figure 3 shows the network setup for a firewall/DMZ Oracle Service Bus test. The following characteristics describe the setup:

- The firewall between FW-B and WLI Lab Network is a hardware firewall—Cisco PIX 506.

- The firewall between FW-E and WLI Lab Network is a software firewall—Checkpoint Firewall/1.
- Solid lines represent the paths followed by the messages that are sent from FW-B.
- Dotted lines represent the paths followed by the messages sent from FW-E.
- Ports 7001 and 7002 are opened on the external IPs for the internal boxes.
- The client and Oracle WebLogic Server business service domain are hosted on FW-B.
- The Oracle Service Bus domain is hosted on FW-B.

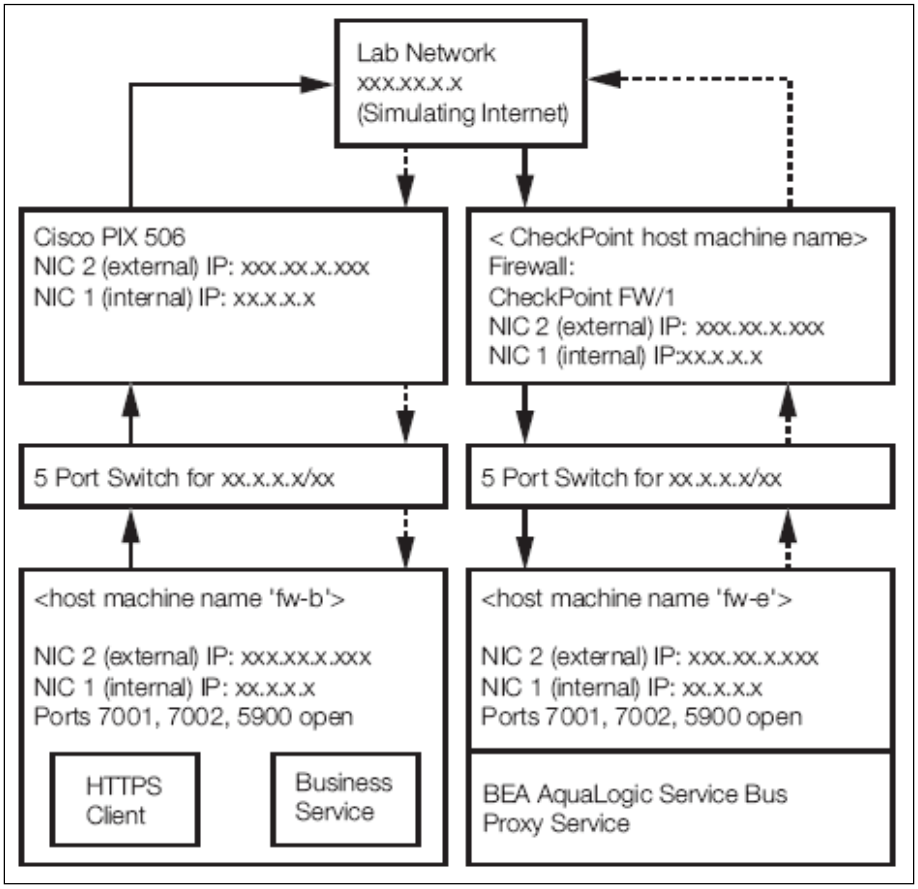

Figure 3: An Oracle Service Bus DMZ/firewall test hardware network setup

A typical setup includes two firewalls (Figure 4): one in front of the public servers and one between the public servers and the internal LAN. This setup is typically referred to as the DMZ, because the zone between the two firewalls is not trusted and is heavily restricted. Additionally, you can set up application proxies (such as WWW and File Transfer Protocol proxies), and then block outbound access from the internal LAN on the exterior firewall and force all clients to go through your application proxies (which can have antivirus capabilities, for example). The DMZ should be a "quiet" zone; that is, any hostile packets trying to enter it (from the

internet or the internal LAN) should be blocked at either firewall, increasing the effectiveness of any intrusion-detection systems. (This results in fewer false positives.)

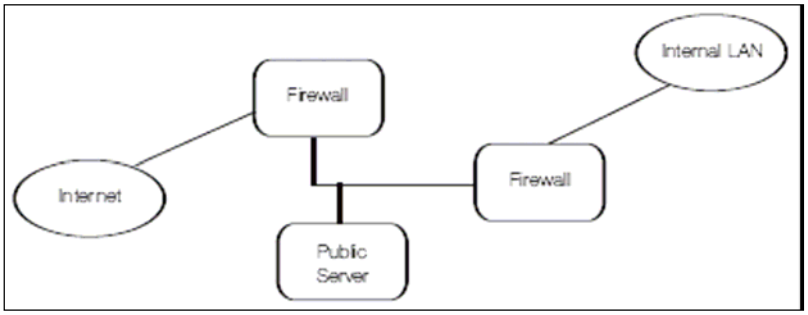

Figure 4: A LAN setup

Inbound and Outbound Two-Way SSL Authentication Configuration

In this scenario, two-way SSL certificate authentication is used among the client, the proxy service, and the business service. Each of them can be deployed on their own Oracle WebLogic Server. If this is the case, each Oracle WebLogic Server sends a digital certificate to the requesting client. The client examines the digital certificate to ensure that it is authentic, has not expired, and matches the Oracle WebLogic Server instance that presented it.

Oracle WebLogic Server accepts only digital certificates that have root certificates from the specified trusted CAs.

The requesting client also presents a digital certificate to Oracle WebLogic Server. Requesting clients must present digital certificates from a specified set of CAs. Oracle WebLogic Server accepts only digital certificates that have root certificates from the specified trusted CAs.

Oracle WebLogic Server Configuration of One-Way and Two-Way SSL

By default, Oracle WebLogic Server is configured for one-way SSL authentication; however, the SSL port is disabled. You can enable an SSL port using the Oracle WebLogic Server administration console.

To use two-way SSL between a client and a server:

- Enable the SSL port on the server.
- Configure identity for the server and trust for the client.
- Enable two-way SSL on the server.
- Configure trust for the server and identity for the server.

The peer certificate chain needs to contain a certificate from the CA that issued the peer's identity certificate. This CA certificate does not need to be the root CA certificate.

To acquire a digital certificate for your server, first generate a public key, a private key, and a certificate signature request (CSR) that contains your public key. You

then send the CSR to a CA and follow the CA's procedures for obtaining a signed digital certificate.

When you have your private keys, digital certificates, and the additional trusted CA certificates that you need, you must store them so that Oracle WebLogic Server can use them to verify identity.[4] Store private keys and certificates in keystores. (For purposes of backward compatibility, you can also store your private keys and certificates in files.)

To use SSL when connecting to an Oracle WebLogic Server application with your browser, you simply specify HTTPS and the secure port in the URL. For example:

```
https://<hostname>:7002/examplesWebApp/SnoopServlet.jsp
where <hostname> is the name of the system hosting the Web
application.
```

PKI Credential Mapper

The Oracle WebLogic Server PKICredentialMapper stores the key alias and key password (for key pairs) in the embedded Lightweight Directory Access Protocol.

A PKI credential mapper maps key pairs (a public key and a private key) to an alias. The Oracle WebLogic Server PKICredentialMapper stores the key alias and key password (for key pairs) in the embedded Lightweight Directory Access Protocol (LDAP). Key passwords are encrypted before they are stored on LDAP. The PKICredentialMapper does not store the keys; they are stored in a keystore. The PKICredentialMapper is configured with the location of the keystore and the keystore password.

The target Oracle WebLogic Server not only requires a trusted and valid certificate, but it allows the SSL connection only if it can authenticate a user ID stored somewhere in the X.500 distinguished name in the client certificate.

Configuration Steps

Complete the following steps to configure your system:

1. Install Oracle WebLogic Server on each of the participating machines. Optionally, configure server domains in production mode connected to an Oracle database.
2. Configure keystores for all participating domains using the Oracle WebLogic Server administration console.
3. Populate the keystores with server keys and certificates using command line tools.
4. Import each server's certificate as a trusted certificate into the other server's keystore using command line tools.

---

[4] For more information on private key, public key, and certificate requirements and procedures, refer to "Security for Oracle WebLogic Server" at http://e-docs.bea.com/wls/docs91/security.html.

5. Configure the Oracle WebLogic Server that hosts the Oracle Service Bus proxy service to support two-way SSL using the Oracle WebLogic Server administration console.
6. Configure the Oracle WebLogic Server that hosts business services to support two-way SSL using the Oracle WebLogic Server administration console.
7. Add the user ID from the Oracle Service Bus server certificate to the business service server LDAP for business service using the Oracle WebLogic Server administration console.
8. Add the user ID from the client (or client and server**)** certificate to the Oracle Service Bus server LDAP system for Oracle Service Bus using the Oracle WebLogic Server administration console.
9. Configure a PKI credential mapper for Oracle Service Bus using the Oracle WebLogic Server administration console.
10. Configure a proxy service provider using the Oracle Service Bus console.
11. Associate PKI credentials (SSL client key pair) with the proxy service provider using the credentials section of the security configuration module using the Oracle Service Bus console.
12. Register a business service definition, which uses the HTTPS transport protocol with client certificate authentication in Oracle Service Bus using the Oracle Service Bus console.
13. Configure a proxy service to route a message to the business service using the Oracle Service Bus console.

TESTING

Business Service

A business service can be implemented as a servlet like EchoServlet.java shown in the following listing. To compile the code, weblogic.jar and xbean.jar files from the server library must be on your classpath.

EchoServlet.java Source Code

```
import com.bea.xbean.common.IOUtil;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.ServletException;
import javax.servlet.ServletOutputStream;
import javax.servlet.ServletInputStream;
import java.io.IOException;
import java.io.ByteArrayOutputStream;
```

```
import java.util.Enumeration;
public class EchoServlet extends HttpServlet
{
     public static final long serialVersionUID = 1L;
     public void service(HttpServletRequest
     httpServletRequest, httpServletResponse
     httpServletResponse)
               throws IOException, ServletException
     {
          System.out.println("Entered the echo servlet");
          System.out.println("EchoServlet: Content-Type =
          " + httpServletRequest.getContentType());
          /** optionally wait for the specified time
          interval */
          String waitTime =
          httpServletRequest.getParameter("wait");
          if (waitTime != null && waitTime.length() > 0)
          {
              try
              {
                   Thread.sleep(Integer.parseInt(waitTime) *
                   1000);
              }
              catch (Exception e)
              {
                   e.printStackTrace();
              }
          }

          String toThrow =
          httpServletRequest.getParameter("throw");
          if (toThrow != null && toThrow.length() > 0)
          {
              throw new ServletException("Throwing exception
              for testing purposes");
          }
          // Copy all request headers to response (although
          some may not be relevant!)
          Enumeration headers =
          httpServletRequest.getHeaderNames();
          while (headers.hasMoreElements())
          {
              String o = (String) headers.nextElement();
              httpServletResponse.addHeader(o,httpServletReq
              uest.getHeader(o));
          }
          ServletInputStream inputStream =
          httpServletRequest.getInputStream();
          ServletOutputStream outputStream =
          httpServletResponse.getOutputStream();
          boolean useStreaming = true;
          if (useStreaming) {
              System.out.print("EchoServlet: message = ");
              byte[] buf = new byte[1024];
              int totalCount = 0;
              while (true)
              {
```

```
                    int byteCount = inputStream.read(buf, 0,
                    buf.length);
                    if (byteCount == -1) break;
                    System.out.print(new String(buf, 0,
                    byteCount));
                    outputStream.write(buf, 0, byteCount);
                    totalCount += byteCount;
                }
                httpServletResponse.setContentLength
                (totalCount);
                System.out.println();
            } else {
                // Read entire InputStream into byte array
                ByteArrayOutputStream out = new
                ByteArrayOutputStream(1024);
                IOUtil.copyCompletely(inputStream, out);
                // Dump message to be echoed to console
                System.out.println("EchoServlet: message = "
                + out);
                // Set content-length and dump response
                httpServletResponse.setContentLength(out.size(
                ));
                out.writeTo(outputStream);
            }
            String toThrow2 =
            httpServletRequest.getParameter("throw2");
            if (toThrow2 != null && toThrow2.length() > 0)
            {
                throw new ServletException("Throwing exception
                for testing purposes");
            }
            httpServletResponse.setStatus(HttpServletResponse.
            SC_OK);
        }
}
```

To deploy EchoServlet, you must have a Web deployment descriptor—web.xml—and an Oracle WebLogic Server deployment descriptor—weblogic.xml.

Web.xml Source Code

```
<?xml version="1.0" ?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web
Application 2.3//EN""http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
     <servlet>
          <servlet-name> EchoServlet </servlet-name>
          <display-name> EchoServlet </display-name>
          <servlet-class> EchoServlet </servlet-class>
     </servlet>
     <servlet-mapping>
          <servlet-name>EchoServlet</servlet-name>
          <url-pattern>/echo</url-pattern>
     </servlet-mapping>
     <security-constraint>
```

```
            <web-resource-collection>
                  <web-resource-name>AllEndPoints</web
                  resource-name>

                  <url-pattern>/*</url-pattern>
            </web-resource-collection>
            <auth-constraint>
                  <description>
                        These are the roles who have access to
                        inbound HTTPs endpoints

                  </description>
                  <role-name>
                        Admin
                  </role-name>
            </auth-constraint>
            <user-data-constraint>
                  <description>SSL required</description>
                  <transport
                  guarantee>CONFIDENTIAL</transport-guarantee>

            </user-data-constraint>
      </security-constraint>
      <login-config>
            <auth-method>CLIENT-CERT</auth-method>
      </login-config>
      <security-role>
            <description>
                  An administrator
            </description>
            <role-name>
                  Admin
            </role-name>
      </security-role>
</web-app>
```

Weblogic.xml Source Code

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE weblogic-web-app PUBLIC "-//BEA Systems, Inc.//DTD
Web Application 8.1//EN"
"http://www.bea.com/servers/wls810/dtd/weblogic810-web-
jar.dtd">

<weblogic-web-app>
      <security-role-assignment>
            <role-name>Admin</role-name>
      <externally-defined/>
</security-role-assignment>
</weblogic-web-app>
```

Create a WAR file—ExternalServiceClientCert.war—using the JAR tool, with the following command:

```
jar c[v0M]f jarfile [-C dir] inputfiles [-Joption]
```

For example:

```
<your directory>jar cvf ExternalServiceClientCert.war *
```

After starting Oracle WebLogic Server, choose the "Deployments" option from the main menu and deploy ExternalServiceClientCert.war.

Proxy Service

Configure the proxy and business services using the Oracle Service Bus console. The proxy service definition relies on the configuration of a proxy service provider. You must associate PKI credentials (SSL client key pair) with the proxy service provider using the credentials section of the security configuration module.

After you have registered a business service definition, which uses the HTTPS transport protocol with client certificate authentication, configure a proxy service to route a message to the business service.

After you have registered a business service definition, which uses the HTTPS transport protocol with client certificate authentication, configure a proxy service to route a message to the business service.

Client

For testing this scenario, the Grinder open source development test framework is used. It is available for download at the following URL: http://sourceforge.net/projects/grinder/.

1. Download the grinder-3.0-beta27.zip or later release.
2. To run the Grinder client you need three files: grinder.properties, https-ClientCert.py, and an input XML file.

Example File 1: grinder.properties

```
# Example grinder.properties
# grinder.jvm.arguments=-Dpython.home=d:/jython/jython-
2.1
grinder.processes=1
grinder.threads=1
grinder.runs=1
grinder.useConsole=false
grinder.logDirectory=log
grinder.numberOfOldLogs=1
#grinder.initialSleepTime=500

#grinder.sleepTimeFactor=0.01
#grinder.sleepTimeVariation=0.005
grinder.script=https-ClientCert.py
```

Example File 2: ClientCert.py

```
# A simple example using the HTTP plug-in that shows
# the retrieval of a single page via HTTP. The
# resulting page is written to a file. More complex
# HTTP scripts are best created with the TCPProxy.
from net.grinder.script.Grinder import grinder
from net.grinder.script import Test
from net.grinder.plugin.http import HTTPRequest
from HTTPClient import HTTPResponse
from HTTPClient import HTTPConnection
from HTTPClient import NVPair
from net.grinder.plugin.http import HTTPPluginControl
import jarray
```

```
test1 = Test(1, "Request resource")
request1 = test1.wrap(HTTPRequest())
class TestRunner:
     def __call__(self):
            #Set the client cert to be send
        grinder.SSLControl.setKeyStoreFile("TestClien
        tKeystore.jks","password")

        file =
        open("./INPUTS/sslInputString.xml","r")

          contents = file.read()
          print "---------request---------------"
          print contents
          print "------------------------"
          file.close()
          request1.addHeader("Content
          type","text/xml")

        result =
        request1.POST("https://172.16.1.225:7002/C
        ientCertPS",contents)
        print "---------response---------------"
          print result.getText()
          print "------------------------"
```

Example File 3: An Input XML File

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelopexmlns:soap=http://schemas.xmls
oap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

    <soap:Body>

          <Result xmlns="urn:acme-org:results">
              <Job>
                  <contacts>
                      <contact>J Doe</contact>
                      <contact>J Q Public</contact>
                  </contacts>
              </Job>
          </Result>
     </soap:Body>
</soap:Envelope>
```

Set up the classpath in a process environment by executing a script like setEnvBEA.cmd in the cmd shell.

setEnvBEA.cmd

```
@echo off
set BEA_HOME=D:\wlw\src_15004jr\bea
set
DOMAIN_HOME=D:\wlw\src_15004jr\domains\alsb_client_domain
set WEBLOGIC_HOME=%BEA_HOME%\weblogic90
set BEA_JDK=D:\wlw\dev\src\build\jrockit-jdk1.5.0_04
```

```
set OLDPATH=%PATH%
set PATH=%WEBLOGIC_HOME%\server\bin
set PATH=%PATH%;%BEA_JDK%\bin
set PATH=%PATH%;%OLDPATH%
set CLASSPATH=.;%WEBLOGIC_HOME%\server\lib\weblogic.jar
set WL_HOME=
set ANT_HOME=
set JAVA_HOME=
set ANT_ARGS=
```

1. Start the Grinder client from the directory hosting your scripts.

   C:\bea\user_projects\domains\alsb_services_domain\ClientCert>java net.grinder.Grinder

   The output is displayed in the shell window. It is similar to the following listing:

   12/16/05 2:51:22 PM (agent): The Grinder 3.0-beta27

   12/16/05 2:51:22 PM (agent): Worker process command line: java –classpath

   'E:\Grinder3\grinder-3.0-beta27\lib\grinder.jar;.;E:\Grinder3\grinder-3.0 beta27\lib\jython.jar;C:\bea\jrockit90_150_04\lib\tools.jar;C:\bea\jrockit90_150_ 4\jre\lib\rt.jar;'
   net.grinder.engine.process.GrinderProcess

   12/16/05 2:51:23 PM (agent): process fw-b-0 started

   12/16/05 2:51:26 PM (process fw-b-0): starting threads

   ---------request----------

   ```
   <?xml version="1.0" encoding="utf-8"?>
   <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
       <soap:Body>
           <Result xmlns="urn:acme-org:results">
               <Job>
                   <contacts>
                       <contact>J Doe</contact>
                       <contact>J Q Public</contact>
                   </contacts>
               </Job>
           </Result>
       </soap:Body>
   </soap:Envelope>
   ```

   ---------response----------

   ```
   <?xml version="1.0" encoding="utf-8"?>
   <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   ```

```
<soapenv:Header/>
    <soap:Body xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
        <Result xmlns="urn:acme-org:results">
            <Job>
                <contacts>
                    <contact>J Doe</contact>
                    <contact>J Q Public</contact>
                </contacts>
            </Job>
        </Result>
    </soap:Body>
</soapenv:Envelope>
------------------------

12/16/05 2:51:27 PM (process fw-b-0): finished
12/16/05 2:51:28 PM (agent): finished
```

2. Include the following setting in the setDomainEnv.cmd or setDomainEnv.sh file to receive the full SSL debug stack:

```
set EXTRA_JAVA_PROPERTIES=%EXTRA_JAVA_PROPERTIES%
-Dssl.debug=true
-Dweblogic.StdoutDebugEnabled=true
-Dweblogic.security.SSL.verbose=true
```

The Oracle Service Bus proxy service routes the request to the business service located outside the firewall.

The inbound request is sent through the firewall using the reverse proxy service from the DMZ to the LAN/WAN behind the firewall. Specifically, the Grinder HTTPS client sends the request from outside the firewall to the Oracle Service Bus proxy service behind the firewall. The Oracle Service Bus proxy service routes the request to the business service located outside the firewall.

The conversation between Oracle Service Bus and a server hosting the business services should complete successfully. Recall that we use inbound and outbound two-way SSL transport security.

To validate the SSL handshake, examine the console windows of both the Oracle Service Bus server and the Oracle WebLogic Server that hosts the business services. Look for <HANDSHAKEMESSAGE: > entries to see how the SSL handshake was executed. (The Oracle Service Bus server console window shows the business services server certificate when it is received, and the business services console shows the client's certificate.)

In both console windows you will see alerts and exceptions that indicate the SSL connection was closed. These can be ignored if they are of Type: 0.

The certificate record in the business services console log is similar to the following listing:

<Dec 16, 2005 2:49:29 PM PST> <Debug> <SecuritySSL> <000000> <8025904 received HANDSHAKE>

<Dec 16, 2005 2:49:29 PM PST> <Debug> <SecuritySSL> <000000> <HANDSHAKEMESSAGE:Certificate>

<Dec 16, 2005 2:49:29 PM PST> <Debug> <SecuritySSL> <000000> <validationCallback: validateErr = 0>

<Dec 16, 2005 2:49:29 PM PST> <Debug> <SecuritySSL> <000000> < cert[0] = Serial number: 1169085348630412047812102211314679274 54

Issuer:C=US, L=San Jose, O=BEA Systems, Inc., OU=DEV, CN=QuickSilver-TEST-CA

Subject:C=US, L=San Jose, O=BEA Systems, Inc., OU=DEV, CN=QS-SERVER-TEST

Not Valid Before:Sun Jul 18 00:00:00 PDT 2004

Not Valid After:Thu Oct 18 00:00:00 PDT 2007

Signature Algorithm:SHA1withRSA>

<Dec 16, 2005 2:49:29 PM PST> <Debug> <SecuritySSL> <000000> < cert[1] = Serial number: 0

Issuer:C=US, L=San Jose, O=BEA Systems, Inc., OU=DEV, CN=QuickSilver-TEST-CA

Subject:C=US, L=San Jose, O=BEA Systems, Inc., OU=DEV, CN=QuickSilver-TEST-CA

Not Valid Before:Sat Jul 17 00:00:00 PDT 2004

Not Valid After:Tue Oct 17 00:00:00 PDT 2006

Signature Algorithm:SHA1withRSA>

<Dec 16, 2005 2:49:29 PM PST> <Debug> <SecuritySSL> <000000> <weblogic user specified trustmanager validation status 0>

The certificate record in the Oracle Service Bus console log looks similar to this:

```
<Dec 16, 2005 2:44:11 PM PST> <Debug> <SecuritySSL>
<000000> <17480322 received HANDSHAKE>

<Dec 16, 2005 2:44:11 PM PST> <Debug> <SecuritySSL>
<000000> <HANDSHAKEMESSAGE:Certificate>

<Dec 16, 2005 2:44:11 PM PST> <Debug> <SecuritySSL>
<000000> <validationCallback:
validateErr = 0>

<Dec 16, 2005 2:44:11 PM PST> <Debug> <SecuritySSL>
<000000> < cert[0] = Serial number:
1169085348630412047812102211314679274 54

Issuer:C=US, L=San Jose, O=BEA Systems, Inc., OU=DEV,
CN=QuickSilver-TEST-CA
```

```
Subject:C=US, L=San Jose, O=BEA Systems, Inc., OU=DEV,
CN=QS-SERVER-TEST

Not Valid Before:Sun Jul 18 00:00:00 PDT 2004

Not Valid After:Thu Oct 18 00:00:00 PDT 2007

Signature Algorithm:SHA1withRSA>

<Dec 16, 2005 2:44:11 PM PST> <Debug> <SecuritySSL>
<000000> < cert[1] = Serial number: 0

Issuer:C=US, L=San Jose, O=BEA Systems, Inc., OU=DEV,
CN=QuickSilver-TEST-CA

Subject:C=US, L=San Jose, O=BEA Systems, Inc., OU=DEV,
CN=QuickSilver-TEST-CA

Not Valid Before:Sat Jul 17 00:00:00 PDT 2004

Not Valid After:Tue Oct 17 00:00:00 PDT 2006

Signature Algorithm:SHA1withRSA>

<Dec 16, 2005 2:44:11 PM PST> <Debug> <SecuritySSL>
<000000> <weblogic user specified trustmanager validation
status 0>
```

CONCLUSION

This white paper described an example of the configuration of a client, a proxy service, and a business service communicating over firewalls and a DMZ, using inbound and outbound two-way SSL authentication. Specifically, a client generates inbound traffic to a Oracle Service Bus proxy service over a secure transport-level connection using two-way SSL. The Oracle Service Bus proxy service generates outbound traffic to business services over a secure transport-level connection using two-way SSL.

APPENDIX: REFERENCES AND RELATED DOCUMENTS

Security for Oracle WebLogic Server

http://e-docs.bea.com/wls/docs91/security.html
http://e-docs.bea.com/wls/docs92/security.html
http://e-docs.bea.com/wls/docs100/security.html
http://e-docs.bea.com/wls/docs103/security.html

Understanding Oracle WebLogic Security: Security Fundamentals

http://e-docs.bea.com/wls/docs91/secintro/concepts.html
http://e-docs.bea.com/wls/docs92/secintro/concepts.html
http://e-docs.bea.com/wls/docs100/secintro/concepts.html
http://e-docs.bea.com/wls/docs103/secintro/concepts.html

Oracle Service Bus Documentation: Security Configuration

http://e-docs.bea.com/alsb/docs21/consolehelp/securityconfiguration.html
http://e-docs.bea.com/alsb/docs25/consolehelp/securityconfiguration.html
http://e-docs.bea.com/alsb/docs30/consolehelp/securityconfiguration.html

Oracle Service Bus Security Guide

http://e-docs.bea.com/alsb/docs25/security/
http://e-docs.bea.com/alsb/docs26/security/
http://e-docs.bea.com/alsb/docs30/security/

Securing Oracle WebLogic Server by Configuring SSL

http://e-docs.bea.com/wls/docs91/secmanage/ssl.html#configureSSL
http://e-docs.bea.com/wls/docs92/secmanage/ssl.html#configureSSL
http://e-docs.bea.com/wls/docs100/secmanage/ssl.html#configureSSL
http://e-docs.bea.com/wls/docs103/secmanage/ssl.html#configureSSL

Administration Console Online Help: Configure Two-Way SSL

http://e-docs.bea.com/wls/docs91/ConsoleHelp/taskhelp/security/ConfigureTwowaySSL.html

http://e-docs.bea.com/wls/docs92/ConsoleHelp/taskhelp/security/ConfigureTwowaySSL.html

http://e-docs.bea.com/wls/docs100/ConsoleHelp/taskhelp/security/ConfigureTwowaySSL.html

http://e-docs.bea.com/wls/docs103/ConsoleHelp/taskhelp/security/ConfigureTwowaySSL.html

Configure Keystores

http://e-docs.bea.com/wls/docs91/ConsoleHelp/taskhelp/security/ConfigureKeystoresAndSSL.html

http://e-docs.bea.com/wls/docs92/ConsoleHelp/taskhelp/security/ConfigureKeystoresAndSSL.html

http://e-docs.bea.com/wls/docs100/ConsoleHelp/taskhelp/security/ConfigureKeystoresAndSSL.html

http://e-docs.bea.com/wls/docs103/ConsoleHelp/taskhelp/security/ConfigureKeystoresAndSSL.html

Configuring Identity and Trust

http://e-docs.bea.com/wls/docs91/secmanage/identity_trust.html
http://e-docs.bea.com/wls/docs92/secmanage/identity_trust.html
http://e-docs.bea.com/wls/docs100/secmanage/identity_trust.html
http://e-docs.bea.com/wls/docs103/secmanage/identity_trust.html

Programming Oracle WebLogic Security: Using SSL Authentication in Java Clients

http://e-docs.bea.com/wls/docs91/security/SSL_client.html
http://e-docs.bea.com/wls/docs92/security/SSL_client.html
http://e-docs.bea.com/wls/docs100/security/SSL_client.html
http://e-docs.bea.com/wls/docs103/security/SSL_client.html

Grinder Framework

http://sourceforge.net/projects/grinder/

Firewall Overview

http://www.seifried.org/security/network/firewall/20011025-firewall-overview.html

Setting Up a Test Network

http://www.networkworld.com/columnists/2004/0712nutter.html

Oracle Dev2Dev Code-Sample ID S183: Outbound Transport Security Sample

https://codesamples.projects.dev2dev.bea.com/servlets/Scarab/remcurreport/true/template/ViewIssue.vm/id/S183/nbrresults/184

Oracle Service Bus Behind the Firewall in a Service-Oriented Architecture
Updated October 2008

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
oracle.com