Client authentication using digital certificates in WebSphere Application Server Community Edition

Vamsavardhana Reddy Chillakuru

June 07, 2006

Learn what is required and how to set up IBM® WebSphere® Application Server Community Edition for authenticating and authorizing Web clients in your applications based on digital certificates. Also, find out how to develop and deploy a Web application to use certificates for declarative and programmatic security.

Introduction

Web servers typically use digital certificates for server authentication to Web clients using HTTPS (instead of HTTP protocol) to access the applications running on the server. Client authentication, however, is not often enforced using digital certificates; more common is the situation where the client is authenticated using elements such as a username and password. From the server's perspective, either authentication method is just as good, but authentication using username and password over HTTPS is still potentially vulnerable due to lost, stolen, or otherwise compromised passwords. Client authentication using digital certificates can help overcome the problem of compromised credentials and give you a better sense of security about an application.

This article describes the requirements for performing client authentication with digital certificates, and for making authentication and authorization decisions in Web applications deployed in IBM WebSphere Application Server Community Edition. This article highlights the private-key and certificate management functionality provided by the administrative console, and presents examples of Web applications that illustrate declarative and programmatic Web application security with client authentication using digital certificates.

This article requires WebSphere Application Server Community Edition Version 1.0.1 or later.

Before you continue: This article was written for WebSphere Application Server Community Edition V1.0.1, which was the current version at the time the article was published. Some of the information in this article may not be applicable to later versions. To follow along with this article, be sure to use the product version on which this article is based. If you wish, you can download the current version of WebSphere Application Server Community Edition, or you can download an earlier version by visiting the product archive page.
About digital certificates

The most common use of a digital certificate is to verify that a user sending a message is who they claim to be, and to provide the receiver with the means to encode a reply. This feature is used for Web server and client authentication over HTTPS protocol, which is based on SSL. An entity wishing to send an encrypted message creates a public-private keypair and applies for a digital certificate from a Certificate Authority (CA). The CA issues an encrypted digital certificate containing the applicant's public key and a variety of other identification information. The CA makes its own public key readily available through print publicity. The most widely used standard for digital certificates is X.509.

Using the administrative console

The administrative console provides a convenient, user-friendly way to administer many aspects of WebSphere Application Server Community Edition (hereafter referred to as Community Edition). A work in progress that will continue to evolve over time, the admin console provides several portlets to perform a variety of administration tasks, such as the KeyStore, Web Server, Applications, and Security Realms portlets, which we will refer to throughout this article. The admin console also provides portlets that let an administrator view and monitor the status of the server, such as the Information, JVM, and DB Info portlets. Once Community Edition is started, you can access the admin console at http://localhost:8080/console. The default login is system with a password of manager.

KeyStore portlet

The KeyStore portlet in the administrative console operates on a JKS type Java™ keystore file preconfigured in Community Edition to point to the var/security/keystore file under server base directory. This portlet enables you to:

- Enable administrative security automatically during installation.
- View the keystore contents.
- Generate RSA keypair (of key sizes 512, 1024 and 2048 bits).
- Generate Certificate Signing Requests (CSR).
- Import trusted certificates.
- Delete trusted certificates.
- Import CA reply.
- Delete private-keys.

These operations are an integral part of key management. (Community Edition users do not require key management tools like keytool.) The private keys and trusted certificates in the keystore can be used to configure HTTPS listeners for a Web container. We will use the KeyStore portlet in this article to:

- Delete an existing keypair.
- Generate a new keypair.
- Generate a CSR.
- Import a trusted certificate.
• Import CA reply.

To launch KeyStore portlet in Community Edition, select the Keystore link under the Security heading in the Console Navigation section (Figure 1). To view details of any private-key or trusted certificate entry, click on the view link provided against that alias and the KeyStore portlet will display the entry details.

Figure 1. Administrative console: KeyStore portlet

Web Server portlet

The Web Server portlet enables an administrator to start/stop/delete various network listeners that are available for the Web container. It also lets you add new HTTP, HTTPS, and AJP network listeners. We will use the Web Server portlet in this article to:

• Add a new HTTPS listener.
• Stop and start an existing network listener.

To launch the Web Server portlet, select the Web Server link under the Server heading in the Console Navigation section (Figure 1).

Security Realms portlet

The Security Realms portlet enables an administrator to add a new security realm and edit existing security realms. The portlet also has a usage link that provides information on how applications can be configured to authenticate against the security realm. We will use the Security Realms portlet in this article to add a new Certificate Properties File Realm. Launch the Security Realms portlet by clicking the Security Realms link under the Security heading in the Console Navigation section (Figure 1).
Applications portlets

There are multiple Applications portlets that let an administrator deploy new Web applications, EJB components, J2EE applications, application clients, J2EE connectors, and so on, to Community Edition. The portlets also enable starting, stopping, and uninstalling existing applications, connectors, and more. We will use the Deploy New Applications portlet to deploy our sample applications. To launch this portlet, select the Deploy New link under the Applications heading in the Console Navigation section (Figure 1).

Setting up the keystore

The default keystore (<base-dir>/var/security/keystore) that is distributed with Community Edition contains a keypair with alias "geronimo," and a trusted certificate with the alias "geronimoca". Unless you generate a new keypair after server installation and delete this default keypair, you will be using a private-key that is used by several other Community Edition server installations. In this section, we describe the process of setting up your keystore using the KeyStore portlet, so that it can be used to configure HTTPS listeners.

Generate keypair

To generate a new keypair, launch the KeyStore portlet and click on the generate keypair link (Figure 1). This will display a form where details can be entered for the keypair to be generated (Figure 2). Definitions of key configuration fields follow.

Figure 2. Generate key pair

- **Alias:** Name used to identify this keypair in the keystore. This should be different from the alias of existing entries in the keystore. In our example, we use "mybizzzkeys" as the alias, which is different from the two existing aliases "geronimo" and "geronimoca".
• **Key Algorithm:** The private key algorithm to be used to generate the keypair. The administrative console supports only "RSA".

• **Key Size:** Length in bits of the modulus of RSA keypair. The higher the key size, more secure is the keypair. The administrative console supports 512, 1024, and 2048 key sizes. In our example, we use a key size of 1024. (See Related topics for key size recommendations for RSA keys.

• **Signature Algorithm:** The signature algorithm to be used to sign the self-signed certificate generated as part of “generate keypair” exercise. The administrative console supports MD2withRSA, MD5withRSA, and SHA1withRSA signature algorithms. In our example, we use MD5withRSA.

• **Validity:** Indicates the validity of the keypair in number of days. In our example, we use a validity value of 365.

• **Common Name (CN):** The DNS name of the Web server hosting your Web application(s) that wants to authenticate the server to Web clients. For example, if the application is to be hosted on www.mybizzz.com, use the same as the common name so that clients will use http://www.mybizzz.com or https://www.mybizzz.com to access the application. In our example, we use "localhost" as the common name.

• **Organizational Unit (OU)/Organizational Name (O)/Locality (L)/State (ST):** Complete these fields to reflect the identity of the requestor.

• **Country (C):** The two-letter ISO 3166 country code of the requestor's country. In our example, we use IN, which is the country code for India.

Upon submitting the form, the KeyStore portlet generates a new RSA keypair, a self-signed certificate, and adds the keypair to the keystore. To view the details of the newly added entry, click on the view link (Figure 1) provided against the alias (here, mybizzzkeys). Figure 3 shows the details of this newly generated keypair.

**Figure 3. Private key details**
Generate CSR

To obtain a digital certificate from a Certification Authority (CA), a Certificate Signing Request (CSR) needs to be generated from the keypair and sent to the CA. To generate a CSR for any private-key entry, first view the entry details in the KeyStore portlet (Figure 3). After you select the generate CSR link, the KeyStore portlet will generate a PKCS10 Certification Request and display it in a text area so that the text can be copied and pasted into a text file. Listing 1 shows the CSR generated in our example.

Listing 1. Certificate Signing Request

```
-----BEGIN CERTIFICATE REQUEST-----
MIIBtjCCARCAwewDESMBAGA1UEAxMJbG9jYWxob3N0MRkwEwYDVQQDEw5jb21lMQswCQYDVQQKEw1Tb21l
MIIBtjCCARCAwewDESMBAGA1UEAxMJbG9jYWxob3N0MRkwEwYDVQQDEw5jb21lMQswCQYDVQQKEw1Tb21l
MIIBtjCCARCAwewDESMBAGA1UEAxMJbG9jYWxob3N0MRkwEwYDVQQDEw5jb21lMQswCQYDVQQKEw1Tb21l
-----END CERTIFICATE REQUEST-----
```

Import CA reply

Upon receiving the CSR, the CA will verify the validity of the information in the CSR and issue a certificate. The reply from CA is usually in a PKCS7 encoded text format containing the client's certificate signed by the CA, and may contain additional certificates chaining up to a Root CA. For our purposes, the CA reply (Listing 2) from "My Own Root CA", which is a homegrown CA, contains a single certificate.

Listing 2. CA Reply from "My Own Root CA"

```
-----BEGIN CERTIFICATE-----
MIICdzCCAgAwIBAgIQDAk2cMA0GCSqGSIb3DQEBCwUAMQIGMA4GA1UEBhMCVVRUQxgZDcwNzElMjExNzA1NjM2NjA7
-----END CERTIFICATE-----
```

The CA reply given here is for the certificate request from our example only. The CA reply for any certificate request that you might generate while following along with this article will be different. To import the CA reply shown in Listing 2, overwrite "var/security/keystore" with the keystore.new file provided in the download file, then restart the Community Edition server at this point in the article.

To import CA reply:

1. View the entry details of "mybizzzkeys" in the KeyStore portlet (Figure 3).
2. Click on the **import CA reply** link and paste the contents of the CA reply text file into the text area under the PKCS7 Certificate Reply heading.
3. Click the **Save** button.

The keystore is now ready to be used by Community Edition for Web server authentication using the newly obtained certificate.

**Figure 4. Imported CA reply**

---

**Delete default private-key**

The default keystore contains a private-key with an alias of "geronimo". This is required for starting the default HTTPS listener. Once a new keypair is generated using the KeyStore portlet, the default private-key is no longer required. Since there is no provision to specify the alias of the private-key in the keystore to be used by HTTPS listeners, the default private-key entry should be removed from the keystore.

To remove the default private-key entry:

1. Click on the **view** link (Figure 1) provided against alias "geronimo".
2. Click on the **Delete Entry** link (Figure 3) and confirm the entry deletion.
3. Upon confirmation, the default private-key entry is deleted from the keystore. The keystore now has a single private-key entry with alias "mybizzzkeys". Any HTTPS listeners using the var/security/keystore file will now use this private-key for server authentication.

**Import CA certificate as trusted certificate**

Importing a CA certificate into the keystore as a trusted certificate is required to enable client authentication over HTTPS. Upon importing a CA certificate as a trusted certificate, all certificates issued by that CA will be accepted for client authentication. In our sample applications, we use client certificates issued by "My Own Root CA". To make sure that the HTTPS listener accepts certificates issued by this CA, the CA’s certificate (Listing 3) needs to be imported into the keystore as a trusted certificate.
Listing 3. Certificate of "My Own Root CA"

```
-----BEGIN CERTIFICATE-----
MIICChCCAe+qAwIBAgIBADANBgkqhkiG9w0BAQUFADCBiDELMAkGA1UEBhMCSU4xEDAOBgNVBAgTCE15IFN0YXRlMRQwEgYDVQQHEwtNeSBMb2NhbGl0eTEPMA0GA1UEChMGTXkgT3JnMRQwEgYDVQQQLExteSBPcmcgYWMiMTQzMiNTI2MzAwMS5jxATBgNVBAsMC2ggQ29kZSBTRWxvdWJsZXMgQ29kZSBQcm9tIENlbnRlckBBbXkgQ0Ew
38wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAOGfnak/SzH7D0zLX0zFZ1iI8m6HiPaiWdkYVoA3wxLrrzqRFD7khZUHIRE+nbz/C/DyGkJm3qRT6EJt/h8IPi2N2tgcbEWtmo68aYfMcQQgjPeDpq5MB36ALHL8zdIswZ/t6wr/TogCiJVEUnq+1ERmNHAqMBAAEwDQYJJoZKoIhvCNAQEFQDAQYEAOb/K1JKzq5Vtg/cW236+3Ft4gRNv81azjRsN3CX/zAM6xV1ECKvleLunBrCn/SNMQg5WVv231D6cxU1cE75Ncnb8931fboShpVxHeH6gT5saCie0D+VQQBktuS6QuSNIUZ+iNSr15Nz6yoZ2Iqj/UGeXOMxFg0wA3PDi-----END CERTIFICATE-----
```

To import a trusted certificate:

1. Click on **import trusted certificate** in the KeyStore portlet (Figure 1).
2. Select the file or enter the filename containing the CA's certificate and click on the **View Certificate** button. (In our example, the CA certificate is in a text file, myownrootca-cert.txt, provided in the **download file**.)
3. Enter an alias for this trusted certificate and click on the **Import** button (Figure 5) to complete the import. (The alias used should be different from the aliases of existing entries in the keystore.)

Figure 5. "My Own Root CA" certificate details

HTTPS listener with client authentication

The default HTTPS listener on port 8443 is not enabled for client authentication. This HTTPS listener is configured to use var/security/keystore. After the updates to keystore have been applied...
through the KeyStore portlet in the previous section, this HTTPS listener will use the newly added private-key entry for server authentication. To verify this, stop and start the default HTTPS listener through the Web server portlet and access the URL https://localhost:8443/. (Before accessing this URL, you must install the CA certificate into your Web browser to avoid any warning messages.) When the page has loaded, double-click on the lock icon provided on the browser taskbar and view the certificate details by clicking on the view button (Figure 6). Verify that the certificate is indeed issued by “My Own Root CA”. In our example, we used the Mozilla Firefox Web browser.

Figure 6. Server certificate details viewed in the browser

Add HTTPS listener with client authentication

An HTTPS listener enabled for client authentication is required to configure Web applications for client authentication using digital certificates. In this section, we will look at how a new HTTPS listener can be added using the Web Server portlet, and the significance of various parameters.

To add a new HTTPS Listener:

1. Launch the Web Server portlet and select the Add new HTTPS Listener for Tomcat link.
2. Enter values for all applicable parameters and click the Save button (Figure 7). This will add and start a new HTTPS listener.

Descriptions of key parameters follow.
Figure 7. HTTPS Listener parameters

- **Host**: Host name or IP to which the port will bind. We use 0.0.0.0 in our example.
- **Port**: Network port to bind. In our example, we use 443, which is the default port for HTTPS.
- **Keystore File**: File that holds the keystore. The KeyStore portlet is preconfigured to point to var/security/keystore under the Community Edition install directory. The keypair generated earlier in this article is added to this file by the KeyStore portlet, and so, we will be use var/security/keystore in this field.
- **Keystore Password**: Password used to access the keystore file and the private-key entry. The KeyStore portlet is configured to use the password secret.
- **Keystore Type**: The keystore type of the Keystore File field entry. The keystore type of var/security/keystore is "JKS".
- **Truststore File**: The file that holds the truststore. In our example, the trusted certificates are added to var/security/keystore, which itself will act as truststore. Therefore, we will be use var/security/keystore for this field.
- **Truststore Password**: Password used to verify the Truststore File field, which in our example is the password of var/security/keystore, which is secret.
• **Truststore Type:** The keystore type of the Truststore File field entry. In our example, it is "JKS."

• **HTTPS Algorithm:** The HTTPS Algorithm provider. This should typically be set to match the JVM vendor. Using "JVM Default" for this field will not require reconfiguring the HTTPS listener when the server is run on a different JVM than the value selected here. In our example, we use "JVM Default" for this parameter.

• **HTTPS Protocol:** This should typically be set to TLS, though some JVMs do not work properly with popular browsers unless it is changed to SSL. In our example, we use "SSL" for this parameter.

• **Client Auth Required:** In our example, we set this field to require client authentication. If this is field is set, then clients connecting through this connector must supply a valid client certificate. The validity is checked using the CA certificates stored in the first of the following to be found:
  1. The trust store configured above.
  2. A keystore file specified by the javax.net.ssl.trustStore system property.
  3. java-home/lib/security/jssecacerts.
  4. java-home/lib/security/cacerts.

Since we have configured a truststore, only the certificates issued by those CAs in the truststore will be accepted for client authentication.

### Verify HTTPS listener with client authentication

To verify that the newly added HTTPS listener is functional, access the URL https://localhost/. Since no client certificates issued by any of the trusted CAs are available in the browser, this results in an error and the Web page cannot be accessed.

Now, import a private-key and client certificate issued by "My Own Root CA" provided in the file webclient01.p12 into the Web browser, and then access the URL https://localhost/ again. This time, since the browser has a client certificate issued by one of the CAs in truststore, the browser displays a dialog (Figure 8) that lets you choose a certificate for authenticating to the Web server. After you select the client certificate, the server authenticates the client and the Web page is now accessible.
Adding a security realm for declarative security

Once the Web server has been configured with an HTTPS listener with client authentication, a certificate properties file realm can be added; Web applications can then be configured to authenticate against this realm. This section describes how such a security realm can be added to Community Edition.

Properties files

A certificate properties file realm is configured using two properties file parameters: users file and groups file. Samples are shown below of a users file (Listing 4) and a groups file (Listing 5). These files are based on the client certificates provided in webclient01.p12, webclient02.p12, webclient11.p12 and webclient12.p12 (available in the download file), which should be imported into a Web browser so you can use the sample applications.

Listing 4. var/security/cert-users.properties

```properties
webclient01=CN=Web Client01,OU=Org Unit0,O=Org0,L=Loc0,ST=St0,C=IN
webclient02=CN=Web Client02,OU=Org Unit0,O=Org0,L=Loc0,ST=St0,C=IN
webclient11=CN=Web Client11,OU=Org Unit1,O=Org1,L=Loc1,ST=St1,C=US
webclient12=CN=Web Client12,OU=Org Unit1,O=Org1,L=Loc1,ST=St1,C=US
```
In cert-users.properties file, the X509Certificate.getSubjectX500Principal().getName() of the client certificate is mapped to a username.

Listing 5. var/security/cert-groups.properties

admin=webclient01,webclient02
guest=webclient11,webclient12

In the cert-groups.properties file, the usernames from the cert-users.properties file are mapped to groups. Save these two properties files to the <base-dir>/var/security directory.

Add certificate properties file realm

To add a certificate properties file realm:

1. Launch the Security Realms portlet and select the add new security realm link.
2. Enter the Name of Security Realm (we use cert-prop-file-realm).
3. For Realm Type, select Certificate Properties File Realm, then Next.
4. Enter the URI for the Users File, which is var/security/cert-users.properties, and for the Groups File, var/security/cert-groups.properties. Click Next.
5. Click the Skip Test and Deploy button to add the realm.

The added realm can now be used to configure Web application security. When a client authenticates against this realm with a valid client certificate (that is, it matches the X500Principal in cert-users.properties), the realm returns a subject with a GeronimoUserPrincipal and a GeronimoGroupPrincipal.

Sample applications

Install CA certificate and client certificates into a Web browser

To use the sample applications, the CA certificate and client certificates provided in the download file should be installed into your Web browser, which will manage trusted CA and client certificates. Your browser will also provide a certificate installation function. For example, Mozilla Firefox provides a Certificate Manager that can be launched by clicking the Manage Certificates button under the Certificates => Advanced section.

Install CA certificate

Most Web browsers are shipped with pre-installed certificates for well known Root CAs, such as Verisign, Equifax, and so on. This is so that any Web sites certified by these CAs are readily verified and accessible by the browser. For our demonstration purposes, we use server and client certificates issued by a homegrown Root CA namely "My Own Root CA". If this CA's certificate (provided in download file) is not installed in the browser when a Web site (in our example, localhost) that was certified by this CA is accessed over HTTPS, then the browser will display a warning message, similar to "Web Site Certified by an Unknown Authority," and offer the option of...
accepting or rejecting the certificate presented by the Web site. If the CA's certificate is installed in
the browser, then the warning message will not display.

In Mozilla Firefox, a CA's certificate can be installed using the Import option under Certificate
Manager => Authorities section.

Install client certificates

The sample applications need the Web browser to authenticate using a digital certificate. Four
PKCS12 keystore files are provided in the download file, each containing a keypair and a
certificate issued by "My Own Root CA". Once installed, these client certificates can be used to
authenticate against Web servers (in our example, localhost) that designate "My Own Root CA"
as trusted. In Mozilla Firefox, a client certificate can be installed using the Import option provided
under the Your Certificates tab in the Certificate Manager section.

Sample application using declarative security

So far in this article, we have set up the infrastructure needed to use client authentication using
certificates. Now we will look at how to configure applications to use client authentication.

The sample application illustrating declarative security uses the security realm we added earlier.
This application contains secured Web resources under three different URL patterns: protect,
guest, and forbidden. Relevant portions of the web.xml application deployment descriptor is shown
in Listing 6. (The complete cert-realm-sample-web.xml file is provided in the download file.)

Listing 6. Declarative security sample: web.xml

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee" ...>
  ...
  <security-constraint>
    <web-resource-collection>
      <web-resource-name>Admin Role</web-resource-name>
      <url-pattern>/protect/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
      <role-name>content-administrator</role-name>
    </auth-constraint>
    <user-data-constraint>
      <transport-guarantee>CONFIDENTIAL</transport-guarantee>
    </user-data-constraint>
  </security-constraint>
  <security-constraint>
    <web-resource-collection>
      <web-resource-name>No Access</web-resource-name>
      <url-pattern>/forbidden/*</url-pattern>
    </web-resource-collection>
    <auth-constraint/>
  </security-constraint>
  ...
  <security-constraint>
    <web-resource-collection>
      <web-resource-name>Guest</web-resource-name>
      <url-pattern>/guest/*</url-pattern>
    </web-resource-collection>
    <auth-constraint/>
  </security-constraint>
  ...
</web-app>
```
This application uses two user roles: content-administrator and guest-user. Notice that the auth-
method is set to CLIENT-CERT so that the application can be configured to authenticate against a "Certificate Properties File Realm". Notice that the transportation-guarantee constraint is set to CONFIDENTIAL. This enables the Web server to forward the request to an HTTPS port if a secured resource is accessed using HTTP.

Relevant portions of the application deployment plan for this sample application is shown in Listing 7. (The complete cert-realm-sample-plan.xml file is provided in the download file.)

**Listing 7. Declarative security sample: deployment plan**

```xml
<web-app configId="cert-realm-sample" ... >
  <context-root>/cert-realm-sample</context-root>
  ... <security-realm-name>cert-prop-file-realm</security-realm-name>
  <security>
    <default-principal realm-name="cert-prop-file-realm">
      <principal class="org.apache.geronimo.security.realm.providers.GeronimoUserPrincipal" name="webclient01"/>
      <!-- name should match the entries from cert-users.properties of the realm -->
    </default-principal>
    <role-mappings>
      <role role-name="content-administrator">
        <realm realm-name="cert-prop-file-realm">
          <principal class="org.apache.geronimo.security.realm.providers.GeronimoGroupPrincipal" name="admin" designated-run-as="true"/>
          <!-- name should match the entries from cert-groups.properties of the realm -->
        </realm>
        <principal class="org.apache.geronimo.security.realm.providers.GeronimoUserPrincipal" name="webclient01"/>
        ... ...
      </role>
    </role-mappings>
  </security>
</web-app>
```

Notice that the security-realm-name in this plan is "cert-prop-file-realm," which we added earlier. The roles in the deployment plan should match those in the web.xml Web deployment descriptor.
The application is packaged in a WAR file, cert-realm-sample.war, and is available in the download file. The application can be deployed using either the Applications portlet or the command line deployer. Once the application is deployed and started, access the application at http://localhost:8080/cert-realm-sample/.

Sample application using programmatic security

The sample application illustrating programmatic security contains a servlet called AuthorizationServlet. If a Web client uses a certificate for client authentication, the servlet container makes the certificate chain available to the servlet in a request attribute with the name "javax.servlet.request.X509Certificate." The AuthorizationServlet retrieves the certificate chain by calling the request.getAttribute() method, which returns an array of certificate objects. The first element in the array corresponds to the client certificate. The certificate can then be processed by the servlet to make authorization decisions based on the certificate fields. In this sample, AuthorizationServlet retrieves and displays various certificate fields, and uses a MyX509Name class to process the certificate name fields. Relevant portions of AuthorizationServlet.java are shown in Listing 8. (The complete file AuthorizationServlet.java is provided in the download file.)

Listing 8. AuthorizationServlet.java

```java
public class AuthorizationServlet extends HttpServlet implements Servlet {

    public static final String SSL_CERT = "javax.servlet.request.X509Certificate";

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
            response.setContentType("text/plain");
            PrintWriter out = response.getWriter();

            java.security.cert.X509Certificate[] certs = (java.security.cert.X509Certificate[])
                    request.getAttribute(SSL_CERT);
            if(certs == null) {
                out.println("Client did not use a certificate for authentication");
                return;
            }

            MyX509Name sname = new MyX509Name(certs[0].getSubjectX500Principal().getName());
            out.println("Welcome "+sname.getCommonName()"!");
            out.println();
            out.println("You have been authenticated using a digital certificate");
            // Display certificate fields. The field values can be checked to make authorization decisions.
            out.println("Cert S.No. : "+certs[0].getSerialNumber());
            out.println();
            out.println("Subject Info:");
            out.println("Common Name : "+ sname.getCommonName());
            out.println("Org Unit: " + sname.getOrganizationalUnit());
            out.println("Org: " + sname.getOrganization());
            out.println("Locality: " + sname.getLocality());
            out.println("State: " + sname.getState());
            out.println("Country: "+ sname.getCountry());
            out.println();

            MyX509Name iname = new MyX509Name(certs[0].getIssuerX500Principal().getName());
            out.println("Issuer Info:");
            out.println("Common Name: "+ iname.getCommonName());
            ...
        }
    }
}```
The web.xml deployment descriptor for this sample application is shown in Listing 9.

**Listing 9. Programmatic security sample: web.xml**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="cert-auth-sample"
  xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
  http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
  version="2.4">
  <display-name>cert-auth-sample</display-name>
  <distributable/>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
  </welcome-file-list>
  <servlet>
    <display-name>AuthorizationServlet</display-name>
    <servlet-name>AuthorizationServlet</servlet-name>
    <servlet-class>clientauth.AuthorizationServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>AuthorizationServlet</servlet-name>
    <url-pattern>/secure/AuthorizationServlet</url-pattern>
  </servlet-mapping>
  <security-constraint>
    <web-resource-collection>
      <web-resource-name>Secure</web-resource-name>
      <url-pattern>/secure/AuthorizationServlet</url-pattern>
    </web-resource-collection>
    <user-data-constraint>
      <transport-guarantee>CONFIDENTIAL</transport-guarantee>
    </user-data-constraint>
  </security-constraint>
</web-app>
```

Notice that the transportation-guarantee constraint is set to CONFIDENTIAL. This enables the Web server to forward the request to an HTTPS port if a secured resource is accessed using HTTP.

The application is packaged in a WAR file, cert-auth-sample.war, and is available in the download file. The application can be deployed using either the Deploy New Applications portlet or the command line deployer. Once the application is deployed and started, access the application at http://localhost:8080/cert-auth-sample/.

**Conclusion**

IBM WebSphere Application Server Community Edition v1.0.1 and later can be configured to use digital certificates for client authentication in Web applications. This article illustrated how to accomplish this, and showed how the administrative console can be used for key management functions, like generating key pairs, importing CA reply and trusted certificates, and more. Through sample applications, we also saw how Web applications can be configured for declarative and programmatic security using certificates for client authentication.
### Downloadable resources

<table>
<thead>
<tr>
<th>Description</th>
<th>Name</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client authentication code samples</td>
<td>samples.zip</td>
<td>173 KB</td>
</tr>
</tbody>
</table>
Related topics

- Get started with WebSphere Application Server Community Edition
- X.500 User Schema
- X.509 Public Key Infrastructure Certificate and CRL Profile

© Copyright IBM Corporation 2006
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)