This page last changed on Feb 25, 2009 by jmazzitelli.

Securing Server-Agent Communications

Typically, the JON Server and JON Agents talk to each other *in the clear*, meaning all communications traffic is unencrypted and no authentication is performed on either end. Many times, the environment in which you install your JON Servers and JON Agents does not warrant the extra setup time and runtime performance degradation you incur when enabling security on JBoss ON communications traffic (for example, if you already have a VPN and/or firewall protections in place that guard your JON Servers and JON Agents against intrusion). However, there are those that need or simply want the peace of mind of knowing their JBoss ON traffic is fully encrypted and authenticated. This section will describe the steps that you need to perform in order to fully secure the communications traffic between JON Servers and JON Agents.

There is a basic authentication mechanism employed by the server in which it assigns security tokens to its agents which are used to identify and "authenticate" registered agents. This token mechanism should not, however, be considered a strong authentication scheme for the purposes of protecting your JBoss ON network from infiltration.

If you need information on configuring the non-security related communications settings, please refer to the Communications Configuration section.

Implications Of Not Using Secure Communications

JBoss ON does not secure the communications between the JON Server and JON Agent by default (out-of-box). Some issues that are of concern when running JBoss ON without secure communications are outlined below. You need to be aware of these issues before deciding to run JBoss ON with an unsecured communications channel between server and agent:

- It is possible for an unauthorized person to install a rogue JON Agent and have that agent register with the JON Server. A rogue agent is one in which the JBoss ON administrator did not install or give permission to register into the JBoss ON system.

- It is possible for an intruder to silently sniff the communications between the JON Agent and JON Server, possibly obtaining very sensitive data about the machines they are running on.
- It is possible for an intruder to capture and manipulate the communications traffic between the JON Agent and JON Server as part of a man-in-the-middle attack, possibly being able to do very damaging things to the machines they are running on.

Running JBoss ON without securing the communications should only be done under the following circumstances, and only when you understand the full implications of doing so (as explained above):

- If you are installing the JON Server and all JON Agents on a fully secured network, with firewalls and/or a VPN limiting access to your entire network to only authorized and trusted personnel.
- If you are running a demo of JBoss ON. When running JBoss ON as a demo, you may typically want to get the system installed and running as quickly and easily as possible. You normally would not want to concern yourself with securing the communications which involves manual, time-consuming steps.

JBoss ON and SSL

JBoss ON utilizes SSL technology to perform both encryption and authentication. You can enable encryption (scrambling the data between server and agent to avoid someone eavesdropping on the traffic) and optionally enable authentication (which prohibits an intruder from attempting to spoof either a JON Server or JON Agent). It is recommended that you understand the basics of SSL and certificate-based security before attempting to secure your JBoss ON communications.

Encryption

By simply using a transport that uses SSL, you automatically get encryption. This means that when you configure your JON Server and JON Agent's communications layer, use an SSL-enabled transport to encrypt the traffic. SSL-enabled transports includes *sslservlet* and *sslsocket*. You do not have to worry about setting up certificates if you want to use SSL-enabled transports just for encryption. The JON Server ships with a certificate and the JON Agent will create a self-signed certificate if it needs one.

Note that it is possible to just use SSL encryption without authentication. Some people may just wish to encrypt their JBoss ON traffic, without requiring the JON Servers and JON Agents authenticating each other. This setup, while less secure, is much easier to setup because it does not require creating and distributing trusted certificates.

Authentication

Authentication via SSL requires that you distribute trusted certificates to your JON Servers and JON Agents. You must also configure those JON Servers and JON Agents to reject any messages coming from remote clients that do not match any of those trusted certificates. In order to support authentication, you must use SSL-enabled transports (which include *sslservlet* and *sslsocket*). You must also obtain trusted certificates for all your servers and agents and package those certificates in a set of keystore and truststore files. You can configure the JON Server to authenticate JON Agents, JON Agents to authenticate the JON Server or both. JBoss ON provides this flexibility in case you only want to authenticate in one direction but not another. However, when people feel the need for authentication, they will usually enable it in both directions.

Authentication requires a bit more work to setup. Because true authentication requires a high degree of trust, you have to manually create and sign your certificates, create keystores and truststores that contain those certificates, then distribute your keystores and truststores in a highly secure manner to all your JON Servers and JON Agents. This may mean going as far as physically hand-delivering your trusted certificates via CD and copying the certificates from the CD to all the computers hosting the JON Servers and JON Agents. Many times it is not as highly paranoid as that - however, at some point along the way, you have to place trust in whatever certificates you are using and distributing.

Setting Up Secure JBoss ON Communications

The following are the steps necessary to set up secure communications in JBoss ON. Remember that you have the option for encryption-only or encryption-with-authentication. If you choose the former, you do not need to perform any of the steps dealing with the creation, packaging and distribution of certificates and keystores/truststores. If you wish to have full security with encryption-with-authentication, follow all the steps listed below. If you are setting up with encryption-with-authentication, these steps assume you want authentication in both directions (that is, the agents will need to authenticate with the server and the server will need to authenticate with the agents).

The JBoss ON low-level communications layer is based on JBoss/Remoting and uses what is known as a *transport* to ship messages back and forth between agents and servers. A transport can be either unencrypted (like the raw *socket* transport or *servlet*) or it can be encrypted (like *sslsocket* or *sslservlet*).

The servlet-based transports, *servlet* and *sslservlet*, are HTTP and HTTPS transports respectively. They are "servlet"-based because the HTTP/HTTPS traffic is routed through a servlet running in the Tomcat server hosted within the JON Server. We use these servlet based transports because they leverage the highly performant Tomcat connector infrastructure with no need for additional thread pooling to accept incoming agent requests since Tomcat will handle the requests. However, using these servlet-based transports means that the configuration you set up here for agent communications are the same settings that take effect for user requests coming into the GUI via a browser. You may not want this if, for example, you want your agents to always SSL-authenticate themselves with certificates but you do not want to require your users to have their own certificates installed in their browsers just to access the JON GUI over https. In this case, you will want to use non-servlet-based transports for agent-to-server communications. It is possible to use the *sslsocket* transport which does not utilize the Tomcat server when receiving incoming agent requests. The *sslsocket* transport will use its own SSL configuration, external to the Tomcat SSL configuration, thus allowing agents to be secured one way while allowing browser requests to the GUI over https to be secured another way.

The JON Agent does not host a servlet container, so you cannot use a servlet based transport (*servlet*, *sslservlet*) for server-to-agent communications. You can only use servlet based transports for agent-to-server communications.

We are going to have the server-to-agent channel use the *sslsocket* transport and the agent-to-server channel use the *sslservlet* transport for the following steps. Later on, we will discuss using the *sslsocket* transport for agent-to-server communications.

Step 1 - Use SSL Transport To Enable Encryption

The first thing you need to do is tell the JON Servers and JON Agents to use SSL when talking to each other.

JON Server Instructions

- Shutdown the JON Server
- Go to the `/bin` directory under the location where your JON Server is installed
- In that directory, find the file `rhq-server.properties` and load it into your favourite text editor
- Find the following configuration preferences used to set the server's communications connector and set them appropriately:

```
rhq.communications.connector.transport=sslservlet
rhq.communications.connector.bind-address=
rhq.communications.connector.bind-port=
rhq.communications.connector.transport-params=/jboss-remoting-servlet-invoker/ServerInvokerServlet
```

  where *transport* is the SSL transport you wish to use - **sslservlet** in this case. Because we are using sslservlet transport, the other settings can typically be left as-is - empty value defaults for *bind-address* and *bind-port* and the servlet path for *transport-params*.

- If you only want SSL encryption, ensure that certificate based authentication is disabled by having the following properties set as below:

```
rhq.server.tomcat.security.client-auth-mode=false
rhq.server.client.security.server-auth-mode-enabled=false
```

- *(Optional)* You may wish to explicitly define the secure socket protocol used by this connector, although the default (`TLS`) is usually good enough. If the default protocol of TLS is not what you want, find the following configuration preferences and set them appropriately:

```
rhq.server.tomcat.security.secure-socket-protocol=TLS
rhq.server.client.security.secure-socket-protocol=TLS
```

- Save the configuration file
- If you do not plan on proceeding to setup SSL authentication at this stage, you can restart the JON Server.

JON Agent Instructions

There are several ways to set the agent SSL configuration preferences. You can either:

- enter the JON Agent's *advanced setup* mode using the command line options `--cleanconfig --setup --advanced`
- from the agent prompt, you can enter the command `setup advanced`
- you can directly edit the `conf/agent-configuration.xml` file and restart the agent with the `--cleanconfig` command line option

If you want to set these preferences using the advanced setup mode, you can enter `!?` at any prompt to get help.

The following instructions will assume you are going through the prompts in setup mode, but they will also tell you the specific configuration preference names that are being set in case you want to directly edit the agent-configuration.xml file.

Unless specified in the following instructions, you can accept the default values for the setup prompts.

- Answer all prompts until you get to the prompt asking for the **Agent Transport Protocol** and enter `sslsocket`. The name for this configuration preference is *rhq.communications.connector.transport*.
- At the prompt asking for the **RHQ Server Hostname or IP Address**, enter the public endpoint address of the JON Server. This is typically just the hostname where your server is running. If you are not sure of this value, you can easily determine it by going to the server GUI's *Administration* > *Manage Servers* page - the public endpoint is listed there. Note that this hostname or IP address must be routable by the agent, otherwise, the agent will get connection failures when it tries to talk to the server. The name for this configuration preference is *rhq.agent.server.bind-address*.
- At the prompt asking for the **RHQ Server Port**, enter the port that the JON Server will be listening to for agent requests. Because we are using the *sslservlet* transport, this will be the Tomcat secure port, whose default is 7443. The name for this configuration preference is *rhq.agent.server.bind-port*.
- At the prompt asking for the **RHQ Server Transport Protocol**, enter `sslservlet` which is the JON Server's new transport that we set in the previous section. The name for this configuration preference is *rhq.agent.server.transport*.
- At the prompt asking for the **RHQ Server Transport Parameters**, enter the JON Server's new transport parameters as defined by its *rhq.communications.connector.transport-params* setting which, for sslservlet transport, must be "/jboss-remoting-servlet-invoker/ServerInvokerServlet". The name for this configuration preference is *rhq.agent.server.transport-params*.
- If you only want SSL encryption, ensure that certificate based authentication is disabled by having the following properties set:
  - Client Authentication Mode : none
  - Server Authentication Mode Enabled? : false
- *(Optional)* You may wish to explicitly define the agent connector's transport parameters. You may also wish to explicitly set the secure socket protocols used by the agent (however, the default protocol of TLS is usually sufficient).
  - **Agent Transport Parameters** are the optional JBoss/Remoting transport parameters that are used by the agent and server to interact with the agent's connector. See the JBoss/Remoting documentation for specifics. The name for this configuration preference is *rhq.communications.connector.transport-params*.
  - **Incoming Secure Socket Protocol** is the protocol used when accepting incoming messages from the JON Server (make sure this matches the JON Server's protocol setting *rhq.server.client.security.secure-socket-protocol*). The name for this configuration preference is *rhq.communications.connector.security.secure-socket-protocol*.
  - **Outgoing Secure Socket Protocol** is the protocol used when sending outgoing messages to the JON Server (make sure this matches the JON Server's protocol setting *rhq.communications.connector.security.secure-socket-protocol*). The name for this configuration preference is *rhq.agent.client.security.secure-socket-protocol*.
- Exit the agent (effectively shutting it down) and then restart it.

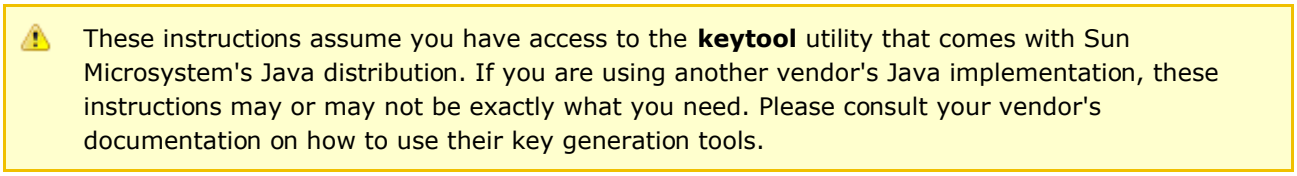
At this point, you have now configured your JON Servers and JON Agents to encrypt their messages to each other via SSL. You can be assured that no one can effectively eavesdrop on the communications between them. However, if you wish to strengthen the security of the network traffic even further, continue on to the next series of steps which will enable certificate-based authentication between your JON Servers and JON Agents.

Step 2 - Prepare Your SSL Certificates

If you wish to have your JON Servers and JON Agents authenticate one another, you need something that "identifies" each one of them. SSL requires digital certificates for this purpose. If your company or organization can request and receive officially signed certificates from a trusted CA, you will need to obtain one certificate for each of your JON Servers and JON Agents that you plan on deploying in your JBoss ON environment. If you already have the certificates given to you from your CA, you must place

each of them in their own keystore file and combine all of them into a single truststore file. Otherwise, please follow the instructions to create your own certificates.

The purpose of these instructions is to generate a *keystore* file and a *truststore* file for each JON Server and JON Agent. Each keystore file will contain a single self-signed certificate that belongs to one of the JON Server or JON Agent entities. Each truststore file contains all the certificates belonging to every JON Server and JON Agent.



- For each JON Server or JON Agent, execute this command to generate its keystore file:

```
keytool -genkey -dname "CN=myhost.mycorp.com" -keystore myhost-keystore.dat -validity 3650
-alias myhost \
-keyalg DSA -storetype JKS -keypass jonpassword -storepass jonpassword
```

  In this example, assume one of my JON Agents will be installed on a machine whose hostname is "myhost.mycorp.com". This command creates and self-signs a certificate and stores it under the alias "myhost" in the file "myhost-keystore.dat". The certificate is good for 10 years (3650 days). The certificate keys were generated using the DSA algorithm and were stored in the keystore using the JKS format. The key itself and the keystore file have been password protected with the password "jonpassword". It is recommended that you at least choose your own passwords when generating your keystores.

The important part is to make sure you set the Common Name (CN) of the Distinguished Name (the *-dname* option) to the correct address where this keystore is to be installed. That is because as part of the SSL handshake, a remote client will attempt to verify that the issuer of the certificate (as listed in the CN) is the same name as where the certificate actually came from. Now that we have generated a self-signed certificate for this JON Agent and stored it in a file named "myhost-keystore.dat", we can store that away for now and continue generating certificates/keystores for the rest of the machines until we have one keystore file for each and every machine that will host a JON Server or JON Agent. It is best to name the keystores so you remember which keystore file belongs to which machine (hence why, in the example above, the hostname was part of the filename). Same holds true with the alias names.

- Put each self-signed certificate you generated in the previous step in a single truststore file. You do this by exporting each certificate from each keystore and importing them all into a single truststore file.
  - For each keystore file, export the self-signed certificate:

```
keytool -export -keystore myhost-keystore.dat -alias myhost -storetype JKS -storepass
jonpassword \
-file myhost-cert
```

    This extracts the self-signed certificate from the previously created myhost-keystore.dat file and stores the certificate in the file "myhost-cert".

  - For each exported self-signed certificate, import them into a single truststore file:

```
keytool -import -keystore truststore.dat -alias myhost -storetype JKS -file myhost-cert
-noprompt -keypass jonpassword -storepass jonpassword
```

This command is similiar to the -genkey command used to create the original keystore certificate. However, rather than asking the keytool to generate a new certificate, we are giving it an existing certificate and asking keytool to place it in the truststore file. The -keystore option defines the name of our truststore file. -alias is the name that we assign this certificate within the truststore file (note that for convenience, we give it the same alias under which it was found in its keystore file).

- Repeat these steps for each keystore file you created. You want to import every certificate into the same truststore - eventually having a single truststore file that contains all of your certificates. For example, if I had a total of 5 JON Servers and JON Agents in my JBoss ON environment, I would have 5 separate keystore files but a single truststore.dat file that contains all 5 certificates. When you are all done, you can use keytool to list the certificates in your truststore file, to make sure you did them all:

```
> keytool -list -keystore truststore.dat -storepass jonpassword -storetype JKS

Keystore type: JKS
Keystore provider: SUN

Your keystore contains 2 entries

anotherhost, Feb 25, 2007, trustedCertEntry,
Certificate fingerprint (MD5): 24:D9:8A:50:BA:1B:26:08:DC:44:A8:2A:9E:8A:43:D9
myhost, Feb 25, 2007, trustedCertEntry,
Certificate fingerprint (MD5): 91:F8:78:15:21:E8:0C:73:EC:B6:3B:1D:5A:EC:2B:01
```

Step 3 - Distribute Your Keystores and Truststores

At this point, you have created a set of keystore files (one for each JON Server and JON Agent in your JBoss ON environment) and a single truststore file (a duplicate copy is to be given to each JON Server and JON Agent). You must now distribute those files to all the machines where your JON Servers and JON Agents live. You must do so in a secure fashion and ensure that no one can steal, intercept or otherwise manipulate your keystore/truststore files. You must also make sure that you distribute the keystore files to the host machines that match the certificates' CN host addresses. If you mix them up and, for example, put the "myhost" keystore file on the "anotherhost.mycorp.com" machine, the SSL communications will fail for the JON Server or JON Agent running on "anotherhost".

JON Server Instructions

Each JON Server distribution has a JBossAS instance within it. That JBossAS has a `conf` directory that you can use for the location to store the server's keystore/truststore files (but technically, you can put them anywhere that the server can access them. Please ensure you remember the location because you will need it for the next step). For each JON Server, take its keystore file (make sure the keystore file has the appropriate CN value that matches the JON Server's hostname) and store it in `$RHQ_SERVER_HOME/jbossas/server/default/conf` under the name `keystore.dat`. Make a copy of your truststore file and place it in that same directory under the name `truststore.dat`.

JON Agent Instructions

Each JON Agent distribution has a `/conf` directory. It is the logical choice to store the agent's keystore/truststore files. (note: putting them here makes them safe when performing agent auto-updates

- agents will retain all keystore/truststore files that are found in the `/conf` and `/data` directory). For each JON Agent, take its keystore file (make sure the keystore file has the appropriate CN value that matches the JON Agent's hostname) and store it in the agent's `/conf` directory. Make a copy of your truststore file and place it in the agent's `/conf` directory as well.

Step 4 - Tell The JON Servers and Agents About Their Keystores/Truststores

Now you have to tell your JON Servers and JON Agents where your keystore and truststore files are in addition to providing other information about those files so they can be read properly. After completing this step, your JON Servers and JON Agents will be able to successfully authenticate themselves to each other.

JON Server Instructions

- Shutdown the JON Server.
- Open the `/bin` directory in the JON Server installation directory.
- In that directory, find the file `rhq-server.properties` and load it into your favorite text editor.
- Find the following configuration preferences used to set the server's security using information about your new keystore and truststore files:

```
# Server-side SSL Security Configuration for HTTPS thru Tomcat
# These are used for browser https: access and for incoming messages from agents over
sslservlet transport
# [you cannot use ${x} variables - see https://jira.jboss.org/jira/browse/JBWEB-74]
rhq.server.tomcat.security.client-auth-mode=true
rhq.server.tomcat.security.secure-socket-protocol=TLS
rhq.server.tomcat.security.algorithm=SunX509
rhq.server.tomcat.security.keystore.alias=myhost
rhq.server.tomcat.security.keystore.file=conf/keystore.dat
rhq.server.tomcat.security.keystore.password=jonpassword
rhq.server.tomcat.security.keystore.type=JKS
rhq.server.tomcat.security.truststore.file=conf/truststore.dat
rhq.server.tomcat.security.truststore.password=jonpassword
rhq.server.tomcat.security.truststore.type=JKS

...

# Client-side SSL Security Configuration (for outgoing messages to agents)
rhq.server.client.security.secure-socket-protocol=TLS
rhq.server.client.security.keystore.file=${jboss.server.home.dir}/conf/keystore.dat
rhq.server.client.security.keystore.algorithm=SunX509
rhq.server.client.security.keystore.type=JKS
rhq.server.client.security.keystore.password=jonpassword
rhq.server.client.security.keystore.key-password=jonpassword
rhq.server.client.security.keystore.alias=myhost
rhq.server.client.security.truststore.file=${jboss.server.home.dir}/conf/truststore.dat
rhq.server.client.security.truststore.algorithm=SunX509
rhq.server.client.security.truststore.type=JKS
rhq.server.client.security.truststore.password=jonpassword
rhq.server.client.security.server-auth-mode-enabled=true
```

What is being configured here is the server-side Tomcat SSL security settings (*rhq.server.tomcat.security...*) to handle incoming messages from JON Agents via the sslservlet transport and the client-side SSL security settings (*rhq.server.client.security...*) to handle outgoing messages to JON Agents via the sslsocket transport. Because we are sharing the keystore and truststore for both directions, a lot of these values are the same.
Since we want to enable both server-side and client-side with SSL authentication, we set *rhq.server.tomcat.security.client-auth-mode* to "true" (which tells Tomcat to only process an incoming request if it has a valid SSL certificate) and we set *rhq.server.client.security.server-auth-mode-enabled* to "true" (meaning any outgoing messages sent to a JON Agent will only be sent if that JON Agent has a valid SSL certificate).

The rest of the settings and their values should be fairly self-evident. You are simply telling the JON Server where it can find its keystore and truststore files, the passwords to access data in those files, the alias of the JON Server's own certificate as found in the keystore, etc.

- Save the configuration file
- Restart the JON Server

JON Agent Instructions

- Answer all prompts until you get to the prompts asking about security (the configuration preferences can be edited directly in agent-configuration.xml - see all the preferences whose names start with *rhq.communications.connector.security* and *rhq.agent.client.security*, they match up fairly easily with the prompt names you see below). Because we are sharing the keystore and truststore for both directions, a lot of these values are the same. The prompts to look for and their new values are:
  - **Client Authentication Mode**: need
  - **Server Authentication Mode Enabled?**: true
  - **Incoming Secure Socket Protocol**: TLS
  - **Server-side Keystore File**: conf/myhost-keystore.dat
  - **Server-side Keystore Algorithm**: SunX509
  - **Server-side Keystore Type**: JKS
  - **Server-side Keystore Password**: jonpassword
  - **Server-side Keystore Key Password**: jonpassword
  - **Server-side Keystore Key Alias**: myhost
  - **Server-side Truststore File**: conf/truststore.dat
  - **Server-side Truststore Algorithm**: SunX509
  - **Server-side Truststore Type**: JKS
  - **Server-side Truststore Password**: jonpassword
  - **Outgoing Secure Socket Protocol**: TLS
  - **Client-side Keystore File**: conf/myhost-keystore.dat
  - **Client-side Keystore Algorithm**: SunX509
  - **Client-side Keystore Type**: JKS
  - **Client-side Keystore Password**: jonpassword
  - **Client-side Keystore Key Password**: jonpassword
  - **Client-side Keystore Key Alias**: myhost
  - **Client-side Truststore File**: conf/truststore.dat
  - **Client-side Truststore Algorithm**: SunX509
  - **Client-side Truststore Type**: JKS
  - **Client-side Truststore Password**: jonpassword

- Exit the agent (effectively shutting it down) and then restart it

At this point, you have now configured your JON Servers and JON Agents to both encrypt their messages to each other and to authenticate each other via SSL. You now can be assured that no one can effectively eavesdrop on your JBoss ON communications nor can an infiltrator attempt to spoof itself as a bogus JON Server or JON Agent.

Step 5 - Test Your Setup

Once you are done with the preceding steps, you can finally restart your JON Servers and JON Agents. They should begin to talk to each other normally - if you have done everything correctly, you will not

notice anything different! If you want to confirm that they really are using SSL authentication, simply remove a keystore or truststore file from either a JON Server or JON Agent and you should begin to notice errors appear in their log files. Removing a keystore file or truststore file from a JON Agent will prohibit that agent from being able to send and receive messages to/from the JON Server - which you can confirm by looking at the agent log file and looking for error messages. After you've finished testing, make sure you remember to restore the keystore/truststore files. You can test the SSL authentication by creating another keystore for one of your JON Agents, replace that keystore with the original keystore and try to see if that JON Agent can talk to the JON Server. Because this new keystore has a certificate that does not exist in the JON Server's truststore, the JON Server will no longer trust that agent and will reject its messages. In effect, you simulated an infiltrator trying to spoof a JON Agent and the JON Server detected this security breach.

Setting Up Server-Side sslsocket Transport

As mentioned previously, if you use the *sslservlet* transport, messages from the agent to the server are routed through Tomcat, the same as when your users make browser requests to the server GUI.

If:

1. you wish to allow your users to access the server GUI over the secure https: protocol...
2. and you use *sslservlet* transport...
3. and you require agents to authenticate themselves to the server via SSL certificates

then you will also require your users' browsers to have certificates installed and those user certificates must be placed in your server-side truststore file; otherwise, your users that try to access the server GUI over https: will find that Tomcat will reject their requests due to missing certificates, even if the user can authenticate themselves using their JON username and password. In highly secure environments, this may not be a problem because users might already have certificates assigned to them and installed in their browser and you might be able to get or build a truststore that contains all of your users' certificates. However, under most situations, this is not desireable. Users normally do not have their own certificates installed in their browsers and even if they did, you probably cannot obtain their certificates ahead of time to place them in the server's truststore.

The users will authenticate themselves to the server GUI via their own JON username and password. So the question becomes, how can you require agents to authenticate themselves with certificates without requiring users to do so? The answer lies in the ability of JON to use a different transport. Rather than use *sslservlet*, we can tell the JON Server to use *sslsocket* instead. This will enable the JON Server to open a special server-side socket that you designate that will be used to accept agent requests. This special socket will circumvent Tomcat and the Tomcat configuration - it will have its own SSL security configuration, allowing you to configure Tomcat to be less strict in the requests it accepts.

Using *sslsocket* requires your server to open another additional port, so if you have a firewall between your agents and server, you must punch a hole through your firewall so the agents can access this new port; otherwise, the agents will not be able to communicate with the server.

All of the steps covered in previous sections are still valid. The only difference in being able to use *sslsocket* versus *sslservlet* is a few configuration setting changes. These differences will be explained below.

sslsocket JON Server Settings

In general, the security settings with the names that start with *rhq.server.tomcat.security* do not need to be changed and should be left as they were when the server was initially installed; instead, you use the *rhq.communications.connector.security* settings:

```
rhq.communications.connector.transport=sslsocket
rhq.communications.connector.bind-address=
rhq.communications.connector.bind-port=55555
rhq.communications.connector.transport-params=

rhq.communications.connector.security.secure-socket-protocol=TLS
rhq.communications.connector.security.keystore.file=${jboss.server.home.dir}/conf/keystore.dat
rhq.communications.connector.security.keystore.algorithm=SunX509
rhq.communications.connector.security.keystore.type=JKS
rhq.communications.connector.security.keystore.password=jonpassword
rhq.communications.connector.security.keystore.key-password=jonpassword
rhq.communications.connector.security.keystore.alias=myhost
rhq.communications.connector.security.truststore.file=${jboss.server.home.dir}/conf/truststore.dat
rhq.communications.connector.security.truststore.algorithm=SunX509
rhq.communications.connector.security.truststore.type=JKS
rhq.communications.connector.security.truststore.password=jonpassword
rhq.communications.connector.security.client-auth-mode=true
```

Here the transport is now **sslsocket**. Because we are using sslsocket transport, you need to indicate which port to bind to (this is the port the server will listen to when receiving agent requests - in this example we used port 55555 but you can use any free port). The bind-address can still be left as-is, it will pick up a default for the server. You can specify a *bind-address* if you want to explicitly tell the server what to bind to. The *transport-params* can be any valid JBoss/Remoting transport parameters - this can be left blank. See the JBoss/Remoting documentation for more information on what you can do with these transport parameters.

For the security settings, all have the same values as before except their names start with *rhq.communications.connector.security*, not *rhq.server.tomcat.security*.

After you made your changes, start the server and go to the *Administration* > *Manage Servers* page - verify that the public endpoint address and port are the ones you want. Edit the server definition in the UI if they do not match.

sslsocket JON Agent Settings

Very little has to change from the previous instructions to get the agent to talk to the server over the *sslsocket* protocol. The only difference is that you have to tell the agent to use the *sslsocket* transport, what the new port is and what the server's transport parameters are:

- **RHQ Server Port**: enter the new port that the JON Server will be listening to for agent requests. The name for this configuration preference is *rhq.agent.server.bind-port*.
- **RHQ Server Transport Protocol**, enter `sslsocket`. The name for this configuration preference is *rhq.agent.server.transport*.
- **RHQ Server Transport Parameters**, enter the JON Server's new transport parameters as defined by its *rhq.communications.connector.transport-params* setting. The name for this configuration preference is *rhq.agent.server.transport-params*.