**SecurityManager**

JRE allows application developers to configure fine-grained security policies for constraining Java code via SecurityManagers.

This allows you to accept or reject a program’s attempt to

* Shut down the JVM,
* Access local disk files, or
* Connect to arbitrary network locations.

Turning on the security manager with a carefully configured security policy can ensure that malicious network clients cannot command the JVM to access anything that the administrator did not preapprove.

The configuration file for security decisions in Tomcat is (by default) $CATALINA\_HOME/conf/catalina.policy,

Written in the standard Java security policy file format.

The JVM reads this file when you invoke Tomcat with the -security option.

The file contains a series of permissions, each granted to a particular codebase or set of Java classes.

Securing Tomcat with SSL

certification authority CA : The digital server certificate is issued by one of a small handful of companies worldwide

These companies

verify that the person to whom they are issuing the digital server certificate to really is who he claims to be

These companies then sign your server certificate using their own certificate. Theirs has been, in turn, signedby another, and so on. This series of certificates is known as a certificate chain. At the end of the chain, there is one master certificate, kept in a very secure location.

So, how do you generate your server certificate? You use either the Java keytool program

(part of the standard JDK) or the popular OpenSSL

**Tomcat-Configuration[Chapter - 7]**

Configuring Tomcat is mainly done by editing files[$CATALINA\_HOME/conf] and restarting Tomcat.

Tomcat's configuration files:

* server.xml: The main Tomcat configuration file.
* web.xml : configuration file for servlets and other settings that are global to all web applications.
* tomcat-users.xml: The default list of roles, users, and passwords used by Tomcat’s UserDatabaseRealm for authentication.
* catalina.policy: The Java security policy file for Tomcat.
* context.xml: The default context settings that are applied to all deployed contexts of all hosts.

Simple server.xml [Tomcat 6.0]

<Server port="8005" shutdown="SHUTDOWN">

<Listener className="org.apache.catalina.core.JasperListener" />

<Listener className="org.apache.catalina.mbeans.ServerLifecycleListener" />

<Listener className="org.apache.catalina.mbeans.GlobalResourcesLifecycleListener" />

<GlobalNamingResources>

<Resource name="UserDatabase" auth="Container"

type="org.apache.catalina.UserDatabase"

description="User database that can be updated and saved"

factory="org.apache.catalina.users.MemoryUserDatabaseFactory"

pathname="conf/tomcat-users.xml"/>

</GlobalNamingResources>

<Service name="Catalina">

<Connector port="8080" protocol="HTTP/1.1"

maxThreads="150" connectionTimeout="20000" redirectPort="8443"/>

<Engine name="Catalina" defaultHost="localhost">

<Host name="localhost" appBase="webapps"

unpackWARs="true" autoDeploy="true"

xmlValidation="false" xmlNamespaceAware="false">

</Host>

</Engine>

</Service>

</Server>

Server

The Server element refers to the entire Tomcat server. There can be only one Server element in this file.

If you need two servers, run two Tomcat instances!

It accepts the three attributes:

* className The fully qualified class name of the server container implementation to use.
  + org.apache.catalina.core.StandardServer
* port Port number to listen for shutdown requests on. 8005
  + This port is only accessible from the computer on which you are running Tomcat, to prevent people out on the Internet from shutting down your server.
* shutdown The string to be sent to stop the server. SHUTDOWN
  + The shutdown string is sent to the running Tomcat instance when you invoke the catalina script with the stop argument. As you server.xml file shouldnot be visible outside your local machine, if you change this string from its default, it will be harder for outsiders (system crackers) to shut down your server.

Tomcat listens only for these connections on the localhost address, meaning that it should be impossible to shut down your machine from elsewhere on the network.

Service

the Service element must contain

one or more Connector elements and

one and only one Engine.

Each Connector receives all incoming requests on a given port and protocol and passes them to the Engine which processes the requests.

<Service name="Catalina">

<Connector port="8080" protocol="HTTP/1.1"

maxThreads="150" connectionTimeout="20000" redirectPort="8443"/>

<Engine name="Catalina" defaultHost="localhost">

<Host name="localhost" appBase="webapps"

unpackWARs="true" autoDeploy="true"

xmlValidation="false" xmlNamespaceAware="false">

</Host>

</Engine>

</Service>

Engine : An Engine element represents the software that receives requests from one of the Connectors in its Service, hands them off for processing, and returns the results to the Connector.

Host: A Host element represents one host (or virtual host) computer, whose requests are being processed within a given Engine.

web.xml

The web.xml file format is defined in the Java Servlet Specification,\* so this file format will be used in every servlet-conforming Java servlet container.

This file format is used in two places in Tomcat:

* CATALINA\_BASE/conf directory and
* Each web application.

Settings in the conf/web.xml file apply to all web applications, whereas settings in a application’s WEB-INF/web.xml apply only to that application.

With each time Tomcat deploys an application (during startup or when the application is reloaded),

Child elements of web-app

* **Icon:**  A display file, for use in GUI administration tools.
* **display-name** Short name, for use in GUI admin tools.
* **description** Longer description.
* distributable Whether the web application can be load-balanced, i.e., distributed to multiple servers.
* context-param Parameters to be made available to all servlets.
* filter Provides a general-purpose servlet-based filtering mechanism.
* filter-mapping Maps the invocation of a filter to either a servlet name or to a URL pattern.
* listener Context or session Listener classes.
* **Servlet**: Short name, class name, and options for a servlet.
* **servlet-mapping:** Specifies any non default URL for a servlet.
* Session-config Specifies session configuration (only session timeout in present version of specification).
* mime-mapping MIME types for files on server.
* welcome-filelist
* Alternate default page in directories.
* error-page Alternate error page by HTTP error code.
* jsp-config Used for global configuration for all JSP files in the webapp.
* resource-ref Reference to JNDI factory for objects such as SQL DataSources.
* resource-env-ref Reference to “administered objects,” such as JMS queues.
* Messagedestination: Declares that the webapp will use a message destination object.
* messagedestination-ref
* Configures a JNDI mapping for a message destination object.
* securityconstraint
* Requires authentication (e.g., for a protected area of a web site).
* login-config Specifies how the login mechanism is to work for a security-constraint.
* security-role Lists name of security role, for use with security-constraint.
* service-ref Declares a reference to a web service.
* env-entry JNDI lookup of static objects.
* ejb-ref Reference to EJBs used by servlets.
* ejb-local-ref Reference to EJB local interfaces used by servlets.
* locale-encodingmapping-
* list

do not run Tomcat with a stripped-down (Simple) server.xml file. We have seen many Tomcat users cause themselves obscure, difficult to diagnose problems by running with a minimalist server.xml file.

Your webapp may indirectly depend on some server.xml configuration elements or specific attribute settings.

When you remove them and start Tomcat, it fails that appear either that Tomcat is malfunctioning

or the webapp is malfunctioning.

When you upgrade from one Tomcat version to another, the attributes and values of the elements can (and do) change.

If you make your own minimal server.xml file, you will not be able to cleanly diff yours versus the new version’s to find out what changed.

once in a while a bug will be fixed that requires both Tomcat’s code to change as well as configuration change to server.xml.

If you upgrade and get new JAR files but not the corresponding server.xml change, the bug persists, but only in your installation.

When you ask about the bug on the mailing list or in IRC, people will tell you that they cannot reproduce your problem,

and it’s because they’re using the stock server.xml.

Long after you have initially set up Tomcat andyour minimal server.xml, you or

someone else comes back to it to make some changes. You have since forgotten

how the configuration works, or never knew in the first place, andyou needto

know what was configuredspecially for your webapp(s). You cannot simply diff

against the stock server.xml because it is too different from yours.

Instead, you could take a stock server.xml file and make just the configuration changes that

would make it functionally equivalent to your minimal server.xml file.