Apache Tomcat

Logging in Tomcat

Introduction

The internal logging for Apache Tomcat uses JULI, a packaged renamed fork of Apache Commons Logging that is hard-coded to use the java.util.logging framework. This ensures that Tomcat's internal logging and any web application logging will remain independent, even if a web application uses Apache Commons Logging.

To configure Tomcat to use an alternative logging framework for its internal logging, follow the instructions provided by the alternative logging framework for redirecting logging for applications that use java.util.logging. Keep in mind that the alternative logging framework will need to be capable of working in an environment where different loggers with the same name may exist in different class loaders.

A web application running on Apache Tomcat can:

- Use any logging framework of its choice.
- Use system logging API, java.util.logging.
- Use the logging API provided by the Java Servlets specification, jakarta.servlet.ServletContext.log(...)

The logging frameworks used by different web applications are independent. See class loading for more details. The exception to this rule is java.util.logging. If it is used directly or indirectly by your logging library then elements of it will be shared across web applications because it is loaded by the system class loader.

Java logging API — java.util.logging

Apache Tomcat has its own implementation of several key elements of java.util.logging API. This implementation is called JULI. The key component there is a custom LogManager implementation, that is aware of different web applications running on Tomcat (and their different class loaders). It supports

private per-application logging configurations. It is also notified by Tomcat when a web application is unloaded from memory, so that the references to its classes can be cleared, preventing memory leaks.

This java.util.logging implementation is enabled by providing certain system properties when starting Java. The Apache Tomcat startup scripts do this for you, but if you are using different tools to run Tomcat (such as jsvc, or running Tomcat from within an IDE), you should take care of them by yourself.

More details about java.util.logging may be found in the documentation for your JDK and on its Javadoc pages for the java.util.logging package.

More details about Tomcat JULI may be found below.

Servlets logging API

The calls to jakarta.servlet.ServletContext.log(...) to write log messages are handled by internal Tomcat logging. Such messages are logged to the category named

org.apache.catalina.core.ContainerBase.[${engine}].[${host}].[${context}]

This logging is performed according to the Tomcat logging configuration. You cannot overwrite it in a web application.

The Servlets logging API predates the java.util.logging API that is now provided by Java. As such, it does not offer you much options. E.g., you cannot control the log levels. It can be noted, though, that in Apache Tomcat implementation the calls to ServletContext.log(String) or GenericServlet.log(String) are logged at the INFO level. The calls to ServletContext.log(String, Throwable) or GenericServlet.log(String, Throwable) are logged at the SEVERE level.

Console

When running Tomcat on unixes, the console output is usually redirected to the file named catalina.out. The name is configurable using an environment variable. (See the startup scripts). Whatever is written to System.err/out will be caught into that file. That may include:

- Uncaught exceptions printed by java.lang.ThreadGroup.uncaughtException(..)
- Thread dumps, if you requested them via a system signal

When running as a service on Windows, the console output is also caught and redirected, but the file names are different.

The default logging configuration in Apache Tomcat writes the same messages to the console and to a log file. This is great when using Tomcat for development, but usually is not needed in production.

Old applications that still use System.out or System.err can be tricked by setting swallowOutput attribute on a Context. If the attribute is set to true, the calls to System.out/err during request processing will be intercepted, and their output will be fed to the logging subsystem using the jakarta.servlet.ServletContext.log(...) calls.
Note, that the swallowOutput feature is actually a trick, and it has its limitations. It works only with direct calls to System.out/err, and only during request processing cycle. It may not work in other threads that might be created by the application. It cannot be used to intercept logging frameworks that themselves write to the system streams, as those start early and may obtain a direct reference to the streams before the redirection takes place.

Access logging

Access logging is a related but different feature, which is implemented as a Valve. It uses self-contained logic to write its log files. The essential requirement for access logging is to handle a large continuous stream of data with low overhead, so it only uses Apache Commons Logging for its own debug messages. This implementation approach avoids additional overhead and potentially complex configuration. Please refer to the Valves documentation for more details on its configuration, including the various report formats.

Using java.util.logging (default)

The default implementation of java.util.logging provided in the JDK is too limited to be useful. The key limitation is the inability to have per-web application

logging, as the configuration is per-VM. As a result, Tomcat will, in the default configuration, replace the default LogManager implementation with a container friendly implementation called JULI, which addresses these shortcomings.

JULI supports the same configuration mechanisms as the standard JDK java.util.logging, using either a programmatic approach, or properties files. The main difference is that per-classloader properties files can be set (which enables easy redeployment friendly webapp configuration), and the properties files support extended constructs which allows more freedom for defining handlers and assigning them to loggers.

JULI is enabled by default, and supports per classloader configuration, in addition to the regular global java.util.logging configuration. This means that logging can be configured at the following layers:

- Globally. That is usually done in the ${catalina.base}/conf/logging.properties file. The file is specified by the java.util.logging.config.file System property which is set by the startup scripts. If it is not readable or is not configured, the default is to use the ${java.home}/lib/logging.properties file in the JRE.
- In the web application. The file will be WEB-INF/classes/logging.properties

The default logging.properties in the JRE specifies a ConsoleHandler that routes logging to System.err. The default conf/logging.properties in Apache Tomcat also adds several AsyncFileHandlers that write to files.

A handler's log level threshold is INFO by default and can be set using SEVERE, WARNING, INFO, CONFIG, FINE, FINER, FINEST or ALL. You can also target specific packages to collect logging from and specify a level.

To enable debug logging for part of Tomcat's internals, you should configure both the appropriate logger(s) and the appropriate handler(s) to use the FINEST or ALL level. e.g.:

org.apache.catalina.session.level=ALL

java.util.logging.ConsoleHandler.level=ALL

When enabling debug logging it is recommended that it is enabled for the narrowest possible scope as debug logging can generate large amounts of information.

The configuration used by JULI is the same as the one supported by plain java.util.logging, but uses a few extensions to allow better flexibility in configuring loggers and handlers. The main differences are:

- A prefix may be added to handler names, so that multiple handlers of a single class may be instantiated. A prefix is a String which starts with a digit, and ends with '.'. For example, 22foobar. is a valid prefix.
- System property replacement is performed for property values which contain ${systemPropertyName}.
- If using a class loader that implements the org.apache.juli.WebappProperties interface (Tomcat's web application class loader does) then property replacement is also performed for ${classloader.webappName}, ${classloader.hostName} and ${classloader.serviceName} which are replaced with the web application name, the host name and the service name respectively.
- By default, loggers will not delegate to their parent if they have associated handlers. This may be changed per logger using the loggerName.useParentHandlers property, which accepts a boolean value.
- The root logger can define its set of handlers using the .handlers property.
- By default the log files will be kept on the file system for 90 days. This may be changed per handler using the handlerName.maxDays property. If the specified value for the property is ≤0 then the log files will be kept on the file system forever, otherwise they will be kept the specified maximum days.

There are several additional implementation classes, that can be used together with the ones provided by Java. The notable ones

are org.apache.juli.FileHandler and org.apache.juli.AsyncFileHandler.

org.apache.juli.FileHandler supports buffering of the logs. The buffering is not enabled by default. To configure it, use the bufferSize property of a handler. The value of 0 uses system default buffering (typically an 8K buffer will be used). A value of <0 forces a writer flush upon each log write. A value >0 uses a BufferedOutputStream with the defined value but note that the system default buffering will also be applied.

org.apache.juli.AsyncFileHandler is a subclass of FileHandler that queues the log messages and writes them asynchronously to the log files. Its additional behaviour can be configured by setting some system properties.

Example logging.properties file to be placed in $CATALINA_BASE/conf:

```
handlers = 1catalina.org.apache.juli.AsyncFileHandler, \
                2localhost.org.apache.juli.AsyncFileHandler, \
                3manager.org.apache.juli.AsyncFileHandler, \
                java.util.logging.ConsoleHandler


.handlers = 1catalina.org.apache.juli.AsyncFileHandler,
java.util.logging.ConsoleHandler


############################################################
# Handler specific properties.
# Describes specific configuration info for Handlers.
############################################################


1catalina.org.apache.juli.AsyncFileHandler.level = ALL
1catalina.org.apache.juli.AsyncFileHandler.directory = ${catalina.base}/logs
```

1catalina.org.apache.juli.AsyncFileHandler.prefix = catalina.

1catalina.org.apache.juli.AsyncFileHandler.maxDays = 90

1catalina.org.apache.juli.AsyncFileHandler.encoding = UTF-8

2localhost.org.apache.juli.AsyncFileHandler.level = ALL

2localhost.org.apache.juli.AsyncFileHandler.directory = ${catalina.base}/logs

2localhost.org.apache.juli.AsyncFileHandler.prefix = localhost.

2localhost.org.apache.juli.AsyncFileHandler.maxDays = 90

2localhost.org.apache.juli.AsyncFileHandler.encoding = UTF-8

3manager.org.apache.juli.AsyncFileHandler.level = ALL

3manager.org.apache.juli.AsyncFileHandler.directory = ${catalina.base}/logs

3manager.org.apache.juli.AsyncFileHandler.prefix = manager.

3manager.org.apache.juli.AsyncFileHandler.bufferSize = 16384

3manager.org.apache.juli.AsyncFileHandler.maxDays = 90

3manager.org.apache.juli.AsyncFileHandler.encoding = UTF-8

java.util.logging.ConsoleHandler.level = ALL

java.util.logging.ConsoleHandler.formatter = java.util.logging.OneLineFormatter

java.util.logging.ConsoleHandler.encoding = UTF-8

##

Facility specific properties.

Provides extra control for each logger.

##

org.apache.catalina.core.ContainerBase.[Catalina].[localhost].level = INFO

org.apache.catalina.core.ContainerBase.[Catalina].[localhost].handlers = \

    2localhost.org.apache.juli.AsyncFileHandler

org.apache.catalina.core.ContainerBase.[Catalina].[localhost].[/manager].level = INFO

org.apache.catalina.core.ContainerBase.[Catalina].[localhost].[/manager].handlers = \

    3manager.org.apache.juli.AsyncFileHandler

For example, set the org.apache.catalina.util.LifecycleBase logger to log

each component that extends LifecycleBase changing state:

#org.apache.catalina.util.LifecycleBase.level = FINE

Example logging.properties for the servlet-examples web application to be placed in WEB-INF/classes inside the web application:

handlers = org.apache.juli.AsyncFileHandler, java.util.logging.ConsoleHandler

##

Handler specific properties.

Describes specific configuration info for Handlers.

##

org.apache.juli.AsyncFileHandler.level = ALL

org.apache.juli.AsyncFileHandler.directory = ${catalina.base}/logs

org.apache.juli.AsyncFileHandler.prefix = ${classloader.webappName}.

org.apache.juli.AsyncFileHandler.encoding = UTF-8

java.util.logging.ConsoleHandler.level = ALL

java.util.logging.ConsoleHandler.formatter = java.util.logging.OneLineFormatter

java.util.logging.ConsoleHandler.encoding = UTF-8

Documentation references

See the following resources for additional information:

- Apache Tomcat Javadoc for the org.apache.juli package.
- Oracle Java 11 Javadoc for the java.util.logging package.

Considerations for production usage

You may want to take note of the following:

- Consider removing ConsoleHandler from configuration. By default (thanks to the .handlers setting) logging goes both to a AsyncFileHandler and to a ConsoleHandler. The output of the latter one is usually captured into a file, such as catalina.out. Thus you end up with two copies of the same messages.
- Consider removing AsyncFileHandlers for the applications that you do not use. E.g., the one for host-manager.
- Consider configuring an Access log.

Apache Portable Runtime (APR) based Native library for Tomcat

Introduction

Tomcat can use the Apache Portable Runtime to provide an OpenSSL based TLS

implementation for the HTTP connectors.

These features allows making Tomcat a general purpose webserver, will enable much better integration with other native web technologies, and overall make Java much more viable as a full fledged webserver platform rather than simply a backend focused technology.

Installation

APR support requires three main native components to be installed:

- APR library
- JNI wrappers for APR used by Tomcat (libtcnative)
- OpenSSL libraries

Windows

Windows binaries are provided for tcnative-2, which is a statically compiled .dll which includes OpenSSL and APR. It can be downloaded from here as 32bit or AMD x86-64 binaries. In security conscious production environments, it is recommended to use separate shared dlls for OpenSSL, APR, and libtcnative-2, and update them as needed according to security bulletins. Windows OpenSSL binaries are linked from the Official OpenSSL website (see related/binaries).

Linux

Most Linux distributions will ship packages for APR and OpenSSL. The JNI wrapper (libtcnative) will then have to be compiled. It depends on APR, OpenSSL, and the Java headers.

Requirements:

- APR 1.6.3+ development headers (libapr1-dev package)
- OpenSSL 1.1.1+ development headers (libssl-dev package)
- JNI headers from Java compatible JDK 1.4+
- GNU development environment (gcc, make)

The wrapper library sources are located in the Tomcat binary bundle, in

the bin/tomcat-native.tar.gz archive. Once the build environment is installed and the source archive is extracted, the wrapper library can be compiled using (from the folder containing the configure script):

./configure && make && make install

APR Components

Once the libraries are properly installed and available to Java (if loading fails, the library path will be displayed), the Tomcat connectors will automatically use APR.

APR Lifecycle Listener Configuration

See the listener configuration.

Virtual Hosting and Tomcat

Assumptions

For the sake of this how-to, assume you have a development host with two host names, ren and stimpy. Let's also assume one instance of Tomcat running, so $CATALINA_HOME refers to wherever it's installed, perhaps /usr/local/tomcat.

Also, this how-to uses Unix-style path separators and commands; if you're on Windows modify accordingly.

server.xml

At the simplest, edit the Engine portion of your server.xml file to look like this:

```
<Engine name="Catalina" defaultHost="ren">
    <Host name="ren"       appBase="renapps"/>
    <Host name="stimpy" appBase="stimpyapps"/>
</Engine>
```

Note that the directory structures under the appBase for each host should not overlap each other.

Consult the configuration documentation for other attributes of the Engine and Host elements.

Webapps Directory

Create directories for each of the virtual hosts:

mkdir $CATALINA_HOME/renapps

mkdir $CATALINA_HOME/stimpyapps

Configuring Your Contexts

General

Contexts are normally located underneath the appBase directory. For example, to deploy the foobar context as a war file in the ren host, use $CATALINA_HOME/renapps/foobar.war. Note that the default or ROOT context for ren would be deployed as $CATALINA_HOME/renapps/ROOT.war (WAR) or $CATALINA_HOME/renapps/ROOT (directory).

NOTE: The docBase for a context should never be the same as the appBase for a host.

context.xml - approach #1

Within your Context, create a META-INF directory and then place your Context definition in it in a file named context.xml. i.e. $CATALINA_HOME/renapps/ROOT/META-INF/context.xml This makes deployment easier, particularly if you're distributing a WAR file.

context.xml - approach #2

Create a structure under $CATALINA_HOME/conf/Catalina corresponding to your virtual hosts, e.g.:

mkdir $CATALINA_HOME/conf/Catalina/ren

mkdir $CATALINA_HOME/conf/Catalina/stimpy

Note that the ending directory name "Catalina" represents the name attribute of

the Engine element as shown above.

Now, for your default webapps, add:

$CATALINA_HOME/conf/Catalina/ren/ROOT.xml

$CATALINA_HOME/conf/Catalina/stimpy/ROOT.xml

If you want to use the Tomcat manager webapp for each host, you'll also need to add it here:

cd $CATALINA_HOME/conf/Catalina

cp localhost/manager.xml ren/

cp localhost/manager.xml stimpy/

Defaults per host

You can override the default values found in conf/context.xml and conf/web.xml by specifying the new values in files named context.xml.default and web.xml.default from the host specific xml directory.

Following our previous example, you could use $CATALINA_HOME/conf/Catalina/ren/web.xml.default to customize the defaults for all webapps that are deployed in the virtual host named ren.

Further Information

Consult the configuration documentation for other attributes of the Context element.

Advanced IO and Tomcat

Introduction

IMPORTANT NOTE: Usage of these features requires using the HTTP connectors. The AJP connectors do not support them.

Asynchronous writes

When using HTTP connectors Tomcat supports using sendfile to send large static

files. These writes, as soon as the system load increases, will be performed asynchronously in the most efficient way. Instead of sending a large response using blocking writes, it is possible to write content to a static file, and write it using a sendfile code. A caching valve could take advantage of this to cache the response data in a file rather than store it in memory. Sendfile support is available if the request attribute org.apache.tomcat.sendfile.support is set to Boolean.TRUE.

Any servlet can instruct Tomcat to perform a sendfile call by setting the appropriate request attributes. It is also necessary to correctly set the content length for the response. When using sendfile, it is best to ensure that neither the request or response have been wrapped, since as the response body will be sent later by the connector itself, it cannot be filtered. Other than setting the 3 needed request attributes, the servlet should not send any response data, but it may use any method which will result in modifying the response header (like setting cookies).

- org.apache.tomcat.sendfile.filename: Canonical filename of the file which will be sent as a String
- org.apache.tomcat.sendfile.start: Start offset as a Long
- org.apache.tomcat.sendfile.end: End offset as a Long

In addition to setting these parameters it is necessary to set the content-length header. Tomcat will not do that for you, since you may have already written data to the output stream.

Note that the use of sendfile will disable any compression that Tomcat may otherwise have performed on the response.

Apache Tomcat - Using Tomcat libraries with Maven

Using Tomcat libraries With Maven

Tomcat Snapshots

Tomcat snapshots are located in the Apache Snapshot Repository. The official URL is

https://repository.apache.org/content/repositories/snapshots/org/apache/tomcat/

Snapshots are done periodically, not on a regular basis, but when changes happen and the Tomcat team deems a new snapshot might useful.

Tomcat Releases

Stable releases are published to the Maven Central Repository. The URL for this is

https://repo.maven.apache.org/maven2/org/apache/tomcat/

Security Considerations

Introduction

Tomcat is configured to be reasonably secure for most use cases by default. Some environments may require more, or less, secure configurations. This page is to provide a single point of reference for configuration options that may impact security and to offer some commentary on the expected impact of changing those options. The intention is to provide a list of configuration options that should be considered when assessing the security of a Tomcat installation.

Note: Reading this page is not a substitute for reading and understanding the detailed configuration documentation. Fuller descriptions of these attributes may be found in the relevant documentation pages.

Non-Tomcat settings

Tomcat configuration should not be the only line of defense. The other components in the system (operating system, network, database, etc.) should also be secured.

Tomcat should not be run under the root user. Create a dedicated user for the Tomcat process and provide that user with the minimum necessary permissions for the operating system. For example, it should not be possible to log on

remotely using the Tomcat user.

File permissions should also be suitably restricted. In the .tar.gz distribution, files and directories are not world readable and the group does not have write access. On Unix like operating systems, Tomcat runs with a default umask of 0027 to maintain these permissions for files created while Tomcat is running (e.g. log files, expanded WARs, etc.).

Taking the Tomcat instances at the ASF as an example (where auto-deployment is disabled and web applications are deployed as exploded directories), the standard configuration is to have all Tomcat files owned by root with group Tomcat and whilst owner has read/write privileges, group only has read and world has no permissions. The exceptions are the logs, temp and work directory that are owned by the Tomcat user rather than root. This means that even if an attacker compromises the Tomcat process, they can't change the Tomcat configuration, deploy new web applications or modify existing web applications. The Tomcat process runs with a umask of 007 to maintain these permissions.

At the network level, consider using a firewall to limit both incoming and outgoing connections to only those connections you expect to be present.

JMX

The security of the JMX connection is dependent on the implementation provided by the JRE and therefore falls outside the control of Tomcat.

Typically, access control is very limited (either read-only to everything or read-write to everything). Tomcat exposes a large amount of internal information and control via JMX to aid debugging, monitoring and management. Given the limited access control available, JMX access should be treated as equivalent to local root/admin access and restricted accordingly.

The JMX access control provided by most (all?) JRE vendors does not log failed authentication attempts, nor does it provide an account lock-out feature after repeated failed authentications. This makes a brute force attack easy to mount and difficult to detect.

Given all of the above, care should be taken to ensure that, if used, the JMX

interface is appropriately secured. Options you may wish to consider to secure the JMX interface include:

- configuring a strong password for all JMX users;
- binding the JMX listener only to an internal network;
- limiting network access to the JMX port to trusted clients; and
- providing an application specific health page for use by external monitoring systems.

Default web applications

General

Tomcat ships with a number of web applications that are enabled by default. Vulnerabilities have been discovered in these applications in the past. Applications that are not required should be removed so the system will not be at risk if another vulnerability is discovered.

ROOT

The ROOT web application presents a very low security risk but it does include the version of Tomcat that is being used. The ROOT web application should normally be removed from a publicly accessible Tomcat instance, not for security reasons, but so that a more appropriate default page is shown to users.

Documentation

The documentation web application presents a very low security risk but it does identify the version of Tomcat that is being used. It should normally be removed from a publicly accessible Tomcat instance.

Examples

The examples web application should always be removed from any security sensitive installation. While the examples web application does not contain any known vulnerabilities, it is known to contain features (particularly the cookie examples that display the contents of all cookies received and allow new cookies to be set) that may be used by an attacker in conjunction with a

vulnerability in another application deployed on the Tomcat instance to obtain additional information that would otherwise be unavailable.

Manager

The Manager application allows the remote deployment of web applications and is frequently targeted by attackers due to the widespread use of weak passwords and publicly accessible Tomcat instances with the Manager application enabled. The Manager application is not accessible by default as no users are configured with the necessary access. If the Manager application is enabled then guidance in the section **Securing Management Applications** section should be followed.

Host Manager

The Host Manager application allows the creation and management of virtual hosts - including the enabling of the Manager application for a virtual host. The Host Manager application is not accessible by default as no users are configured with the necessary access. If the Host Manager application is enabled then guidance in the section **Securing Management Applications** section should be followed.

Securing Management Applications

When deploying a web application that provides management functions for the Tomcat instance, the following guidelines should be followed:

- Ensure that any users permitted to access the management application have strong passwords.
- Do not remove the use of the LockOutRealm which prevents brute force attacks against user passwords.
- Configure the RemoteAddrValve in the context.xml file for the management application which limits access to localhost by default. If remote access is required, limit it to specific IP addresses using this valve.

Security manager

Support for running under a security manager has been removed for Tomcat 11

onwards. Similar (arguably better) functionality maybe obtained by running a single web application on a dedicated Tomcat instance in a dedicated environment such as a container or VM.

server.xml

General

The default server.xml contains a large number of comments, including some example component definitions that are commented out. Removing these comments makes it considerably easier to read and comprehend server.xml.

If a component type is not listed, then there are no settings for that type that directly impact security.

Server

Setting the **port** attribute to -1 disables the shutdown port.

If the shutdown port is not disabled, a strong password should be configured for **shutdown**.

Listeners

The APR Lifecycle Listener is not stable if compiled on Solaris using gcc. If using the APR/native connector on Solaris, compile it with the Sun Studio compiler.

The JNI Library Loading Listener may be used to load native code. It should only be used to load trusted libraries.

The Security Lifecycle Listener should be enabled and configured as appropriate.

Connectors

By default, a non-TLS, HTTP/1.1 connector is configured on port 8080. Connectors that will not be used should be removed from server.xml.

AJP is a clear text protocol. AJP Connectors should normally only be used on trusted networks. If used on an untrusted network, use of the secret attribute will limit access to authorised clients but the secret attribute will be visible to anyone who can observe network traffic.

AJP Connectors block forwarded requests with unknown request attributes. Known safe and/or expected attributes may be allowed by configuration an appropriate regular expression for the allowedRequestAttributesPattern attribute.

The **address** attribute may be used to control which IP address a connector listens on for connections. By default, a connector listens on all configured IP addresses.

The **allowBackslash** attribute allows non-standard parsing of the request URI. Setting this attribute to a non-default value when behind a reverse proxy may enable an attacker to bypass any security constraints enforced by the proxy.

The **allowTrace** attribute may be used to enable TRACE requests which can be useful for debugging. Due to the way some browsers handle the response from a TRACE request (which exposes the browser to an XSS attack), support for TRACE requests is disabled by default.

The **discardFacades** attribute set to true will cause a new facade object to be created for each request. This is the default value, and this reduces the chances of a bug in an application exposing data from one request to another.

The **encodedSolidusHandling** attribute allows non-standard parsing of the request URI. Setting this attribute to a non-default value when behind a reverse proxy may enable an attacker to bypass any security constraints enforced by the proxy.

The **enforceEncodingInGetWriter** attribute has security implications if set to false. Many user agents, in breach of RFC 7230, try to guess the character encoding of text media types when the specification-mandated default of ISO-8859-1 should be used. Some browsers will interpret as UTF-7 a response containing characters that are safe for ISO-8859-1 but trigger an XSS vulnerability if interpreted as UTF-7.

The **maxPostSize** attribute controls the maximum size of a POST request that will be parsed for parameters. The parameters are cached for the duration of the request so this is limited to 2 MiB by default to reduce exposure to a DOS attack.

The **maxSavePostSize** attribute controls the saving of the request body during FORM and CLIENT-CERT authentication and HTTP/1.1 upgrade. For FORM authentication, the request body is cached in the HTTP session for the duration of the authentication so the cached request body is limited to 4 KiB by default to reduce exposure to a DOS attack. To further reduce exposure to a DoS attack by limiting the permitted duration of the FORM authentication, a reduced session timeout is used if the session is created by the FORM authentication. This reduced timeout is controlled by the authenticationSessionTimeout attribute of the FORM authenticator.

The **maxParameterCount** attribute controls the maximum total number of request parameters (including uploaded files) obtained from the query string and, for POST requests, the request body if the content type is application/x-www-form-urlencoded or multipart/form-data. Requests with excessive parameters are rejected.

The **xpoweredBy** attribute controls whether or not the X-Powered-By HTTP header is sent with each request. If sent, the value of the header contains the Servlet and JSP specification versions, the full Tomcat version (e.g. Apache Tomcat/11.0), the name of the JVM vendor and the version of the JVM. This header is disabled by default. This header can provide useful information to both legitimate clients and attackers.

The **server** attribute controls the value of the Server HTTP header. The default value of this header for Tomcat 4.1.x to 8.0.x is Apache-Coyote/1.1. From 8.5.x onwards this header is not set by default. This header can provide limited information to both legitimate clients and attackers.

The **SSLEnabled**, **scheme** and **secure** attributes may all be independently set. These are normally used when Tomcat is located behind a reverse proxy and the proxy is connecting to Tomcat via HTTP or HTTPS. They allow Tomcat to see the SSL attributes of the connections between the client and the proxy rather than the proxy and Tomcat. For example, the client may connect to the proxy over HTTPS but the proxy connects to Tomcat using HTTP. If it is necessary for Tomcat to be able to distinguish between secure and non-secure connections received by a proxy, the proxy must use separate connectors to pass secure and non-

secure requests to Tomcat. If the proxy uses AJP then the SSL attributes of the client connection are passed via the AJP protocol and separate connectors are not needed.

The **tomcatAuthentication** and **tomcatAuthorization** attributes are used with the AJP connectors to determine if Tomcat should handle all authentication and authorisation or if authentication should be delegated to the reverse proxy (the authenticated user name is passed to Tomcat as part of the AJP protocol) with the option for Tomcat to still perform authorization.

The **requiredSecret** attribute in AJP connectors configures shared secret between Tomcat and reverse proxy in front of Tomcat. It is used to prevent unauthorized connections over AJP protocol.

Host

The host element controls deployment. Automatic deployment allows for simpler management but also makes it easier for an attacker to deploy a malicious application. Automatic deployment is controlled by the **autoDeploy** and **deployOnStartup** attributes. If both are false, only Contexts defined in server.xml will be deployed and any changes will require a Tomcat restart.

In a hosted environment where web applications may not be trusted, set the **deployXML** attribute to false to ignore any context.xml packaged with the web application that may try to assign increased privileges to the web application. Note that if the security manager is enabled that the **deployXML** attribute will default to false.

Context

This applies to Context elements in all places where they can be defined: server.xml file, default context.xml file, per-host context.xml.default file, web application context file in per-host configuration directory or inside the web application.

The **crossContext** attribute controls if a context is allowed to access the resources of another context. It is false by default and should only be changed

for trusted web applications.

The **privileged** attribute controls if a context is allowed to use container provided servlets like the Manager servlet. It is false by default and should only be changed for trusted web applications.

The **allowLinking** attribute of a nested Resources element controls if a context is allowed to use linked files. If enabled and the context is undeployed, the links will be followed when deleting the context resources. Changing this setting from the default of false on case insensitive operating systems (this includes Windows) will disable a number of security measures and allow, among other things, direct access to the WEB-INF directory.

The **sessionCookiePathUsesTrailingSlash** can be used to work around a bug in a number of browsers (Internet Explorer, Safari and Edge) to prevent session cookies being exposed across applications when applications share a common path prefix. However, enabling this option can create problems for applications with Servlets mapped to /*. It should also be noted the RFC6265 section 8.5 makes it clear that different paths should not be considered sufficient to isolate cookies from other applications.

When **antiResourceLocking** is enabled, Tomcat will copy the unpacked web application to the directory defined by the java.io.tmpdir system property ($CATALINA_BASE/temp by default). This location should be secured with appropriate file permissions - typically read/write for the Tomcat user and no access for other users.

Valves

It is strongly recommended that an AccessLogValve is configured. The default Tomcat configuration includes an AccessLogValve. These are normally configured per host but may also be configured per engine or per context as required.

Any administrative application should be protected by a RemoteAddrValve (this Valve is also available as a Filter). The **allow** attribute should be used to limit access to a set of known trusted hosts.

The default ErrorReportValve includes the Tomcat version number in the response sent to clients. To avoid this, custom error handling can be configured within each web application. Alternatively, you can explicitly configure an ErrorReportValve and set its **showServerInfo** attribute to false. Alternatively, the version number can be changed by creating the file CATALINA_BASE/lib/org/apache/catalina/util/ServerInfo.properties with content as follows:

server.info=Apache Tomcat/11.0.x

Modify the values as required. Note that this will also change the version number reported in some of the management tools and may make it harder to determine the real version installed. The CATALINA_HOME/bin/version.bat|sh script will still report the correct version number.

The default ErrorReportValve can display stack traces and/or JSP source code to clients when an error occurs. To avoid this, custom error handling can be configured within each web application. Alternatively, you can explicitly configure an ErrorReportValve and set its **showReport** attribute to false.

The RewriteValve uses regular expressions and poorly formed regex patterns may be vulnerable to "catastrophic backtracking" or "ReDoS". See Rewrite docs for more details.

Realms

The MemoryRealm is not intended for production use as any changes to tomcat-users.xml require a restart of Tomcat to take effect.

The UserDatabaseRealm is not intended for large-scale installations. It is intended for small-scale, relatively static environments.

The JAASRealm is not widely used and therefore the code is not as mature as the other realms. Additional testing is recommended before using this realm.

By default, the realms do not implement any form of account lock-out. This means that brute force attacks can be successful. To prevent a brute force attack, the chosen realm should be wrapped in a LockOutRealm.

Manager

The manager component is used to generate session IDs.

The class used to generate random session IDs may be changed with the **randomClass** attribute.

The length of the session ID may be changed with the **sessionIdLength** attribute.

The **persistAuthentication** controls whether the authenticated Principal associated with the session (if any) is included when the session is persisted during a restart or to a Store.

When using the **JDBCStore**, the session store should be secured (dedicated credentials, appropriate permissions) such that only the **JDBCStore** is able to access the persisted session data. In particular, the **JDBCStore** should not be accessible via any credentials available to a web application.

Manager implementations that persist sessions to storage or replicate sessions in a cluster typically use Java serialization. While the session data is considered trusted (since the application is trusted), system administrators may wish to consider placing restrictions on the Java serialization. This can be done using the **sessionAttributeValueClassNameFilter** attribute. A safe starting value for this attribute is java\\.lang\\.(?:Boolean|Integer|Long|Number|String)|org\\.apache\\.catalina\\.realm\\.GenericPrincipal\\$SerializablePrincipal|\\[Ljava.lang.String; which can then be adjusted to meet the needs of the application. If setting a value for **sessionAttributeValueClassNameFilter** it is recommended that **warnOnSessionAttributeFilterFailure** is set to true.

Cluster

The cluster implementation is written on the basis that a secure, trusted network is used for all of the cluster related network traffic. It is not safe to run a cluster on a insecure, untrusted network.

If you require confidentiality and/or integrity protection then you can use the EncryptInterceptor to encrypt traffic between nodes. This interceptor does not protect against all the risks of running on an untrusted network, particularly

DoS attacks.

web.xml

This applies to the default conf/web.xml file, the /WEB-INF/tomcat-web.xml and the /WEB-INF/web.xml files in web applications if they define the components mentioned here.

The DefaultServlet is configured with **readonly** set to true. Changing this to false allows clients to delete or modify static resources on the server and to upload new resources. This should not normally be changed without requiring authentication.

The DefaultServlet is configured with **listings** set to false. This isn't because allowing directory listings is considered unsafe but because generating listings of directories with thousands of files can consume significant CPU leading to a DOS attack.

The DefaultServlet is configured with **showServerInfo** set to true. When the directory listings is enabled the Tomcat version number is included in the response sent to clients. To avoid this, you can explicitly configure a DefaultServlet and set its **showServerInfo** attribute to false. Alternatively, the version number can be changed by creating the file CATALINA_BASE/lib/org/apache/catalina/util/ServerInfo.properties with content as follows:

server.info=Apache Tomcat/11.0.x

Modify the values as required. Note that this will also change the version number reported in some of the management tools and may make it harder to determine the real version installed. The CATALINA_HOME/bin/version.bat|sh script will still report the correct version number.

The CGI Servlet is disabled by default. If enabled, the debug initialisation parameter should not be set to 10 or higher on a production system because the debug page is not secure.

When using the CGI Servlet on Windows with enableCmdLineArguments enabled, review the setting

of cmdLineArgumentsDecoded carefully and ensure that it is appropriate for your environment. The default value is secure. Insecure configurations may expose the server to remote code execution. Further information on the potential risks and mitigations may be found by following the links in the CGI How To.

HttpHeaderSecurityFilter can be used to add headers to responses to improve security. If clients access Tomcat directly, then you probably want to enable this filter and all the headers it sets unless your application is already setting them. If Tomcat is accessed via a reverse proxy, then the configuration of this filter needs to be co-ordinated with any headers that the reverse proxy sets.

Embedded Tomcat

When using embedded Tomcat, the typical defaults provided by the scripts, server.xml and other configuration are not set. Users of embedded Tomcat may wish to consider the following:

- The listeners normally configured in server.xml, including org.apache.catalina.security.SecurityListener, will not be configured by default. They must be explicitly enabled if required.
- The java.io.tmpdir will not be set (it is normally set to $CATALINA_BASE/temp). This directory is used for various temporary files that may be security sensitive including file uploads and a copy of the web application if anti-resource locking is enabled. Consider setting the java.io.tmpdir system property to an appropriately secured directory.

General

BASIC and FORM authentication pass user names and passwords in clear text. Web applications using these authentication mechanisms with clients connecting over untrusted networks should use SSL.

The session cookie for a session with an authenticated user is nearly as useful as the user's password to an attacker and should be afforded the same level of protection as the password itself. This usually means authenticating over SSL and continuing to use SSL until the session ends.

Tomcat's implementation of the Servlet API's file upload support may use the

directory defined by the java.io.tmpdir system property ($CATALINA_BASE/temp by default) to store temporary files. This location should be secured with appropriate file permissions - typically read/write for the Tomcat user and no access for other users.

Windows Service How-To

Tomcat monitor application

Tomcat11w is a GUI application for monitoring and configuring Tomcat services.

Command line directives

Each command line directive is in the form of //XX[//ServiceName]

If the //ServiceName component is omitted, then the service name is assumed to be the name of the file less the w suffix. So the default service name is Tomcat11.

The available command line directives are:

//ES	Edit service configuration	This is the default operation. It is called if the no option is provided. Starts the GUI application which allows the service configuration to be modified, started and stopped.
//MS	Monitor service	Starts the GUI application and minimizes it to the system tray.
//MR	Monitor & run service	Starts the GUI application and minimizes it to the system tray. Start the service if it is not currently running.
//MQ	Monitor quit	Stop any running monitor for the service.

Tomcat service application

Tomcat11 is a service application for running Tomcat 11 as a Windows service.

Command line directives

Each command line directive is in the form of //XX[//ServiceName]

The available command line directives are:

//TS	Run the service as console application	This is the default operation. It is called if the no option is provided. The ServiceName is the name of the executable without exe suffix, meaning Tomcat11
//RS	Run the service	Called only from ServiceManager
//ES	Start (execute) the service	
//SS	Stop the service	
//US	Update service parameters	
//IS	Install service	
//DS	Delete service	Stops the service if running
//PS	Print service	Prints the command to (re-)create the current configuration
//PP[//seconds]	Pause service	Default is 60 seconds
//VS	Version	Print version and exit
//?	Help	Print usage and exit

Command line parameters

Each command line parameter is prefixed with --. If the command line parameter is prefixed with ++, and the parameter supports multiple values, then it's value will be appended to the existing option. In the table below, parameters that

support multiple values are prefixed with ++.

If the environment variable with the same name as command line parameter but prefixed with PR_ exists it will take precedence. For example:

set PR_CLASSPATH=xx.jar

is equivalent to providing

--Classpath=xx.jar

as command line parameter.

Parameter Name	Default	Description
--Description		Service name description (maximum 1024 characters)
--DisplayName	ServiceName	Service display name
--Install	procrun.exe //RS//ServiceName	Install image
--Startup	manual	Service startup mode can be either **auto** or **manual**
++DependsOn		List of services that this service depend on. Dependent services are separated using either **#** or **;** characters
++Environment		List of environment variables that will be provided to the service in the form **key=value**. They are separated using

	either **#** or **;** characters. If you need to use either the **#** or **;** character within a value then the entire value must be enclosed inside single quotes.
--User	User account used for running executable. It is used only for StartMode **java** or **exe** and enables running applications as service under account without LogonAsService privilege.
--Password	Password for user account set by --User parameter
--ServiceUser	Specifies the name of the account under which the service should run. Use an account name in the form DomainName\UserName. The service process will be logged on as this user. if the account belongs to the built-in domain, you can specify .\UserName. Note that the Service Control Manager does not accept localised forms of the standard names so to use them you need to specify NT

		Authority\LocalService, NT Authority\NetworkService or LocalSystem as appropriate.
--ServicePassword		Password for user account set by --ServiceUser parameter
--LibraryPath		Directory added to the search path used to locate the DLLs for the JVM. This directory is added both in front of the PATH environment variable and as a parameter to the SetDLLDirectory function.
--JavaHome	JAVA_HOME	Set a different JAVA_HOME than defined by JAVA_HOME environment variable
--Jvm	auto	Use either **auto** (i.e. find the JVM from the Windows registry) or specify the full path to the **jvm.dll**. You can use the environment variable expansion here.
++JvmOptions	-Xrs	List of options in the form of **-D** or **-X** that will be passed to the JVM. The options are separated using either **#** or **;** characters. If you need to embed either **#** or **;** characters, put

	them inside single quotes. (Not used in **exe** mode.)
++JvmOptions9	List of options in the form of **-D** or **-X** that will be passed to the JVM when running on Java 9 or later. The options are separated using either **#** or **;** characters. If you need to embed either **#** or **;** characters, put them inside single quotes. (Not used in **exe** mode.)
--Classpath	Set the Java classpath. (Not used in **exe** mode.)
--JvmMs	Initial memory pool size in MiB. (Not used in **exe** mode.)
--JvmMx	Maximum memory pool size in MiB. (Not used in **exe** mode.)
--JvmSs	Thread stack size in KiB. (Not used in **exe** mode.)
--StartMode	One of **jvm**, **Java** or **exe**. The modes are: • jvm - start Java in-process. Depends on jvm.dll, see **--Jvm**. • Java - same as exe, but automatically uses the

		default Java executable, i.e. %JAVA_HOME%\bin\java.exe. Make sure JAVA_HOME is set correctly, or use --JavaHome to provide the correct location. If neither is set, procrun will try to find the default JDK (not JRE) from the Windows registry. • exe - run the image as a separate process
--StartImage		Executable that will be run. Only applies to **exe** mode.
--StartPath		Working path for the start image executable.
--StartClass	Main	Class that contains the startup method. Applies to the **jvm** and **Java** modes. (Not used in **exe** mode.)
--StartMethod	main	Method name if differs then main
++StartParams		List of parameters that will be passed to either StartImage or StartClass. Parameters are

		separated using either **#** or **;** character.
--StopMode		One of **jvm**, **Java** or **exe**. See **--StartMode** for further details.
--StopImage		Executable that will be run on Stop service signal. Only applies to **exe** mode.
--StopPath		Working path for the stop image executable. Does not apply to **jvm** mode.
--StopClass	Main	Class that will be used on Stop service signal. Applies to the **jvm** and **Java** modes.
--StopMethod	main	Method name if differs then main
--StopParams		List of parameters that will be passed to either StopImage or StopClass. Parameters are separated using either **#** or **;** character.
++StopTimeout	No Timeout	Defines the timeout in seconds that procrun waits for service to exit gracefully.
--LogPath	%SystemRoot%\System32\LogFiles\Apache	Defines the path for logging. Creates the directory if

		necessary.
--LogPrefix	commons-daemon	Defines the service log filename prefix. The log file is created in the LogPath directory with .YEAR-MONTH-DAY.log suffix
--LogLevel	Info	Defines the logging level and can be either **Error**, **Info**, **Warn** or **Debug**. (Case insensitive).
--LogJniMessages	0	Set this non-zero (e.g. 1) to capture JVM jni debug messages in the procrun log file. Is not needed if stdout/stderr redirection is being used. Only applies to jvm mode.
--StdOutput		Redirected stdout filename. If named **auto** then file is created inside **LogPath** with the name **service-stdout.YEAR-MONTH-DAY.log**.
--StdError		Redirected stderr filename. If named **auto** then file is created inside **LogPath** with the name **service-stderr.YEAR-MONTH-DAY.log**.

--PidFile	Defines the file name for storing the running process id. Actual file is created in the **LogPath** directory

Installing services

The safest way to manually install the service is to use the provided **service.bat** script. Administrator privileges are required to run this script. If necessary, you can use the /user switch to specify a user to use for the installation of the service.

NOTE: If User Account Control (UAC) is enabled you will be asked for additional privileges when 'Tomcat11.exe' is launched by the script.
If you want to pass additional options to service installer as PR_* environment variables, you have to either configure them globally in OS, or launch the program that sets them with elevated privileges (e.g. right-click on cmd.exe and select "Run as administrator"; on Windows 8 (or later) or Windows Server 2012 (or later), you can open an elevated command prompt for the current directory from the Explorer by clicking on the "File" menu bar). See issue 56143 for details.

Install the service named 'Tomcat11'

C:\> service.bat install

There is a 2nd optional parameter that lets you specify the name of the service, as displayed in Windows services.

Install the service named 'MyService'

C:\> service.bat install MyService

When installing the service with a non-default name, tomcat11.exe and tomcat11w.exe may be renamed to match the chosen service name. To do this, use the --rename option.

Install the service named 'MyService' with renaming

C:\> service.bat install MyService --rename

If using tomcat11.exe, you need to use the **//IS** parameter.

Install the service named 'Tomcat11'

```
C:\> tomcat11 //IS//Tomcat11 --DisplayName="Apache Tomcat 11" ^
        --Install="C:\Program Files\Tomcat\bin\tomcat11.exe" --Jvm=auto ^
        --StartMode=jvm --StopMode=jvm ^
        --StartClass=org.apache.catalina.startup.Bootstrap --StartParams=start ^
        --StopClass=org.apache.catalina.startup.Bootstrap --StopParams=stop
```

Updating services

To update the service parameters, you need to use the **//US** parameter.

Update the service named 'Tomcat11'

```
C:\> tomcat11 //US//Tomcat11 --Description="Apache Tomcat Server - https://tomcat.apache.org/ " ^
        --Startup=auto --Classpath=%JAVA_HOME%\lib\tools.jar;%CATALINA_HOME%\bin\bootstrap.jar
```

If you gave the service an optional name, you need to specify it like this:

Update the service named 'MyService'

```
C:\> tomcat11 //US//MyService --Description="Apache Tomcat Server - https://tomcat.apache.org/ " ^
        --Startup=auto --Classpath=%JAVA_HOME%\lib\tools.jar;%CATALINA_HOME%\bin\bootstrap.jar
```

Removing services

To remove the service, you need to use the **//DS** parameter.
If the service is running it will be stopped and then deleted.

Remove the service named 'Tomcat11'

```
C:\> tomcat11 //DS//Tomcat11
```

If you gave the service an optional name, you need to specify it like this:

Remove the service named 'MyService'

C:\> tomcat11 //DS//MyService

Debugging services

To run the service in console mode, you need to use the **//TS** parameter. The service shutdown can be initiated by pressing **CTRL+C** or **CTRL+BREAK**. If you rename the tomcat11.exe to testservice.exe then you can just execute the testservice.exe and this command mode will be executed by default.

Run the service named 'Tomcat11' in console mode

C:\> tomcat11 //TS//Tomcat11 [additional arguments]

Or simply execute:

C:\> tomcat11

Multiple Instances

Tomcat supports installation of multiple instances. You can have a single installation of Tomcat with multiple instances running on different IP/port combinations, or multiple Tomcat versions, each running one or more instances on different IP/ports.

Each instance folder will need the following structure:

- conf
- logs
- temp
- webapps
- work

At a minimum, conf should contain a copy of the following files from CATALINA_HOME\conf\. Any files not copied and edited, will be picked up by default from CATALINA_HOME\conf, i.e. CATALINA_BASE\conf files override defaults from CATALINA_HOME\conf.

- server.xml
- web.xml

You must edit CATALINA_BASE\conf\server.xml to specify a unique IP/port for the instance to listen on. Find the line that contains <Connector port="8080" ... and add an address attribute and/or update the port number so as to specify a unique IP/port combination.

To install an instance, first set the CATALINA_HOME environment variable to the name of the Tomcat installation directory. Then create a second environment variable CATALINA_BASE and point this to the instance folder. Then run "service.bat install" command specifying a service name.

set CATALINA_HOME=c:\tomcat_11

set CATALINA_BASE=c:\tomcat_11\instances\instance1

service.bat install instance1

To modify the service settings, you can run **tomcat11w //ES//instance1**.

For additional instances, create additional instance folder, update the CATALINA_BASE environment variable, and run the "service.bat install" again.

set CATALINA_BASE=c:\tomcat_11\instances\instance2

service.bat install instance2