AN OUTLINE PRESENTATION ON TOMCAT WEB SERVER AND ITS SECURITY ISSUES

SALVADI MAHENDER,
B.Tech,
Guru Nanak Institute of Technology, Ibrahimpatnam,
R.R Dist, Telangana.

Abstract:

By mean precisely web applications are intricate frameworks and to make these total frameworks, different segments would be incorporated, for example, working framework, database, arrange, web server, and some others. In the web condition world, there ought to be significantly more extension to establish weakness can in any of the parts. In this web age, deal with security of just specific segments and disregarding or give least consideration of other can uncover your whole framework make defenseless. Similarly investing genuine energy to making sure about the tomcat should essential as making sure about as different segments, for example, working frameworks, systems. The point of this undertaking is to build up a comprehension of Tomcat security as far as web application and web server see, and furthermore how to improve the security of tomcat especially by utilizing Java Security Manager. A less work has been done on various parts of security of Tomcat, for example, setup of Tomcat and how to make sure about the Tomcat when all is said in done manner and furthermore extremely modest quantity of research accessible to tell the best way to execute the Java security Manager with Tomcat, how JSM give security to web server. This is unequivocally the primary goal of this undertaking, also with that the job of the JSM inside Tomcat have been examined. To help that an investigation has been given by testing the diverse composed experiments.

Keywords:

Operating system, database, network, web server, the tomcat, Tomcat security.

Introduction:

The Apache Tomcat server is an open source, Java-based web application compartment that was made to run servlet and Java Server Pages (JSP) web applications. It was made under the Apache-Jakarta subproject; notwithstanding, because of its ubiquity, it is currently facilitated as a different Apache venture, where it is upheld and upgraded by a gathering of volunteers from the open source Java people group. Apache Tomcat is truly steady and has the entirety of the highlights of a business web application compartment – yet goes under Open Source Apache License. Tomcat likewise gives extra usefulness that settles on it an incredible decision for building up a total web application arrangement. A portion of the extra highlights gave by Tomcat—other than being open source and free—incorporate the Tomcat Manager application, particular domain executions, and Tomcat valves. At present upheld forms on Apache Tomcat are 5.5X, 6.0X, and 7.0X. Forms sooner than 5.5 are as yet accessible for download, yet they are filed and no help is accessible for them, so clients are urged to utilize the most recent conceivable variant of

Tomcat where accessible. Significant forms on Apache Tomcat match with renditions of the Java Servlet determination, or Java Servlet API, discharged. Along these lines, Tomcat 5.5X backings Servlet API 2.3, Tomcat 6.0X backings Servlet API 2.4, and the most recent Tomcat 7.0 is a reference usage of current Servlet API 3.0. Notwithstanding Servlet API renditions, Tomcat adaptations bolster relating JSP API variants. The JVM similarity likewise relies upon the rendition picked. Table 1-1 gives a cross-reference of Tomcat forms, bolstered JVM renditions, and Servlet API and JSP API discharges.

Apache Tomcat	Servlet API	JSP API	JDK
7.0	3.0	2.2	1.6
6.0	2.5	2.1	1.5
5.5	2.4	2.0	1.4
4.1	2.3	1.2	1.3
3.0	2.2	1.1	1.1

Table 1-1. Tomcat Versions and Supported API and JDK Versions

Tomcat is a Java servlet holder and web server from the Jakarta undertaking of the Apache Software Foundation (http://jakarta.apache.org). A web server is, obviously, the program that dishes out pages in light of solicitations from a client sitting at an internet browser. In any case, web servers aren't constrained to presenting static HTML pages; they can likewise run programs in light of client demands and return the dynamic outcomes to the client's program. This is a part of the web that Apache's Tomcat is excellent at in light of the fact that Tomcat gives both Java servlet and Java Server Pages (JSP) innovations (notwithstanding customary static pages and outside CGI programming). The outcome is that Tomcat is a decent decision for use as a web server for some applications. Also, it's a generally excellent decision in the event that you need a free, open source (http://opensource.org/) servlet and JSP motor. Tomcat can be utilized independent, however it is regularly utilized "behind" customary web servers, for example, Apache httpd, with the conventional server serving static pages and Tomcat serving dynamic servlet and JSP demands. Regardless of what we call Tomcat, a Java servlet compartment or servlet and JSP motor, we mean Tomcat gives a domain wherein servlets can run and JSP can be prepared. Essentially, we can totally say a CGI-empowered Web server is a CGI program compartment or motor since the server can oblige

CGI programs and speak with them as indicated by CGI particular. Among Tomcat and the servlets and JSP code dwelling on it, there is likewise a standard controlling their association, servlet and JSP detail, which is thus a piece of Sun's J2EE (Java 2 Enterprise Edition). Yet, what are servlets and JSP? For what reason do we need them? How about we investigate them in the accompanying subsections before we spread them in substantially more detail later on.

Web Applications of Servlets and JSP:

Advantages

Customarily, before Java servlets, when we notice web applications, we mean an assortment of static HTML pages and a couple CGI contents to create the dynamic substance bits of the web application, which were generally written in C/C++ or Perl. Those CGI contents could be written in a stage free way, in spite of the fact that they didn't should be (and thus frequently weren't). Likewise, since CGI was an acknowledged industry standard over all web server brands and usage, CGI contents could be composed to be web server execution autonomous. By and by, some are and some aren't. The most concerning issue with CGI was that the structure made it intrinsically moderate and unscalable. For each HTTP solicitation to a CGI content, the OS must fork and execute another procedure, and the structure orders this. At the point when the web server is under a high traffic load, an excessive number of procedures fire up and shut down, causing the server machine to devote the vast majority of its assets to process new companies and shutdowns as opposed to satisfying HTTP demands. Concerning versatility, CGI innately has nothing to do with it. As we probably am aware, regardless of whether order line contentions, condition factors or stdin/stdout are utilized for writing to or perusing from CGI programs, every one of them are restricted to the neighborhood machine, not including organizing or disseminated instruments by any stretch of the imagination. Contrastingly, Java servlets and their supporting surroundings are fit for versatility. I will discuss this in future classes. Another way to deal with creating dynamic substance is web server modules. For example, the Apache httpd web server permits powerfully loadable modules to run on startup. These modules can reply on pre-arranged HTTP demand designs, sending dynamic substance to the HTTP customer/program.

This elite strategy for creating dynamic web application content has appreciated some accomplishment throughout the years, however it has its issues also. Web server modules can be written in a stage autonomous way, yet there is no web server usage free standard for web server modules they're explicit to the server you keep in touch with them for, and most likely won't chip away at some other web server execution. Presently let us investigate the Java side. Java carried stage freedom to the server, and Sun needed to use that ability as a component of the arrangement toward a quick and stage autonomous web application standard. The other piece of this arrangement was Java servlets. The thought behind servlets was to utilize Java's basic and amazing multithreading to answer demands without beginning new procedures. You would now be able to compose a servlet-based web application, move it starting with one servlet compartment then onto the next or starting with one PC engineering then onto the next, and run it with no adjustment (truth be told, without recompiling any of its code).

Servlets and JSP:

Quickly, a servlet is a Java program intended to run in a servlet holder (we trust you didn't get that round definition), and a JSP is a site page that can call Java code at demand time. In case you're a framework executive or website admin, you can consider JSPs simply one more scripting and templating language for HTML pages; you can figure out how to compose JSPs that call Java protests much as you would have utilized articles in JavaScript. The thing that matters is that the Java runs on the server side, before the site page is sent to the program. It's increasingly similar to PHP, or even ASP. Composing Java classes, for example, servlets and JSP custom labels, be that as it may, is an assignment likely best left to individuals prepared in Java programming. All the more exactly, a servlet is a Java program that utilizes the javax.servlet bundle, subclasses either the javax.servlet.http.HttpServlet or javax.servlet.GenericServlet Java class, plays out some preparing (anything the software engineer needs that the servlet holder permits) in light of client input, (for example, tapping on a connection or filling in and presenting a web structure), and creates a yield that may be valuable on the Web. A servlet can, obviously, create a HTML page, yet servlets can and have been composed to produce diagrams and outlines in GIF, PNG, and JPEG

positions; printed archives in PDF; or any configuration the engineer can program. A Java Server Page is fundamentally a HTML page that can call Java language usefulness. The plan objective of JSPs is to expel "crude" Java code from the page increase and to have the Java code detached into outside modules that get stacked into the JSP at runtime.

The following gives a servlet example:

```
import java.io. *;

import java.util.*;

import javax.servlet.*;

import javax.servlet.http.*;

/** A servlet that prints a web page with the date at the top. */

public class Hello extends HttpServlet {

/** Called when the user clicks on a link to this servlet

* @parameter request Encapsulates the details about the input.

* @parameter response Encapsulates what you need to get a reply to the

* user's browser.

*/ public void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException {

// Get a writer to generate the reply to user's browser

PrintWriter out = response.getWriter( );

// Generate the HTTP header to say the response is in HTML

response.setContentType("text/html");

out.println("");
out.println( );
out.println("");
out.println("");
out.println("Time on our server is " + new Date( ) + "");
out.println("Hello from a servlet");
out.println("The rest of the actual HTML page would be here...");
out.println("");
}
}
```

The result was that the calls to out.println often outweighed the actual HTML (and when they didn't, it still felt like it to the developer). So Java Server Pages, or JSPs, were developed. You can think of JSPs mainly as HTML pages containing some Java code, instead of Java code containing some HTML. In other words, a JSP is just a servlet turned inside out! So, the above

example could be written as the following JSP, which may be named as date.jsp:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html><head><title>Hello from a JSP</title></head>
<body>
  <p>Time on our server is <%= new java.util.Date() %></p>
  <h1>Hello from a JSP</h1>
  <p>The rest of the actual HTML page would be here...</p>
</body>
</html>
```

The first time a user views the" page," the JSP engine very cleverly turns it inside out so that it can be run as a servlet. In the simplest case, all it does is put out.println calls around each piece of HTML. But then why bother with servlets at all? Excellent question. The answer, of course, is that you can do much more than just print HTML. More details about servlets and JSP is coming up in the following classes.

Installing Tomcat :

Refer to Activating Tomcat 5.0 and Servlet Applications for the instructions for installing Tomcat on Solaris platform.

Web Applications on Tomcat:

Tomcat provides an implementation of both the servlet and JSP specifications. This section discusses what a web application looks like exactly on Tomcat and how we deploy it.

1. Layout of a Web Application:

As we referenced over, a web application is an assortment of static HTML records, dynamic JSPs, and servlet classes. It is characterized as a chain of command of registries and documents in a standard design. Such a progressive system can be gotten to in its "unloaded" structure, where every registry and record exists in the filesystem independently, or in a "stuffed" structure known as a Web ARchive, or WAR document. The previous organization is progressively valuable during advancement, while the last is utilized when you disperse your application to be introduced. The top-level index of your web application chain of command is additionally the archive base of your application. Here, you will put the HTML records and JSP pages that contain your application's UI. At the point when the framework executive sends your application into a specific server, the individual doles out a setting way to your application. Accordingly, if the framework executive doles out your application to the setting way/list, at that point a solicitation URI alluding to/list/index.html will recover the index.html record from your archive root. At the point when you do tests,

you yourself are the head. To put your web application chipping away at Tomcat, you make a subdirectory under Tomcat's webapps catalog, which is the setting way where you should put your web application records. Figure 1 shows the general format of a web application, where test webapp is thought to be the setting of your web application. As should be obvious, the website pages (regardless of whether static HTML, dynamic JSP, or another dynamic templating language's substance) can go in the foundation of a web application index or in practically any subdirectory that you like. Pictures frequently go in a/pictures subdirectory, however this is a show, not a necessity. The WEB-INF registry contains a few explicit bits of substance. Initially, the classes registry is the place you place Java class records, regardless of whether they are servlets or different class documents utilized by a servlet, JSP, or other piece of your application's code. Second, the lib catalog is the place you put Java Archive (JAR) records containing bundles of classes. At long last, the web.xml document is known as a sending descriptor, which contains arrangement for the web application, a depiction of the application, and any extra customization. At the point when you introduce an application into Tomcat, the classes in the WEB-INF/classes/registry, just as all classes in JAR documents found in the WEB-INF/lib/index, are made obvious to different classes inside your specific web application. In this way, on the off chance that you incorporate the entirety of the necessary library classes in one of these spots, you will streamline the establishment of your web application – no change in accordance with the framework class way (or establishment of worldwide library records in your server) will be fundamental.

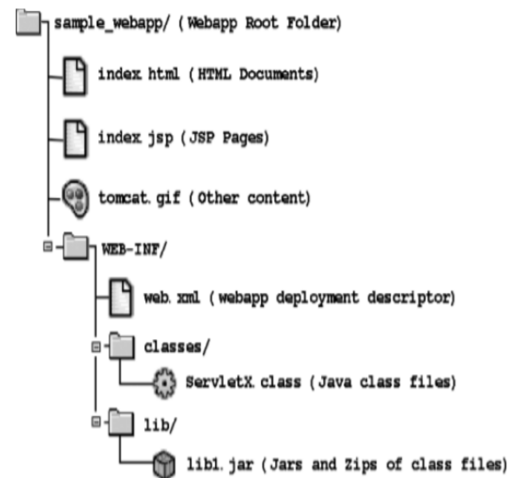

Figure 1: Servlet Web Application File Layout

Container-Managed Security :

Container-managed authentication methods control how a user's credentials are verified when a protected resource is accessed. There are four types of container-managed security that Tomcat supports, and each obtains credentials in a different way:

Basic authentication:

The user's password is required via HTTP authentication as base64-encoded text. 10 When a web application uses basic authentication (BASIC in the web.xml file's auth-method element), Tomcat uses HTTP basic authentication to ask the web browser for a username and password whenever the browser requests a resource of that protected web application. With this authentication method, all passwords are sent across the network in base64-encoded text. The following shows a web.xml excerpt from a club membership web site with a membersonly subdirectory that is protected using basic authentication. Note that this effectively takes the place of the Apache web server's .htaccess files.

```
<!--
  Define the Members-only area, by defining
  a "Security Constraint" on this Application, and
  mapping it to the subdirectory (URL) that we want
  to restrict.
 -->

  <security-constraint>
    <web-resource-collection>
      <web-resource-name>
        Entire Application
      </web-resource-name>
      <url-pattern>/members/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
      <role-name>member</role-name>
    </auth-constraint>
  </security-constraint>

  <!-- Define the Login Configuration for this Application -->

  <login-config>
    <auth-method>BASIC</auth-method>
    <realm-name>My Club Members-only Area</realm-name>
  </login-config>
```

Digest authentication:

The user's password is requested via HTTP authentication as a digest-encoded string.

Form authentication:

The client's secret word is mentioned on a website page structure. Structure validation shows a website page login structure to the client when the client demands a shielded asset from a web application. Indicate structure validation by setting the auth-strategy component's an incentive to "Structure". The Java Servlet Specification Versions 2.2 or more normalize holder oversaw login structure accommodation URIs and parameter names 11 for this kind of

use. This normalization permits web applications that utilization structure confirmation to be compact across servlet compartment executions. To execute structure based confirmation, you need a login structure page and a validation disappointment blunder page in your web application, a security-limitation component like those appeared above, and a login-config component in your web.xml document like the one appeared as follows:

```
<login-config>
  <auth-method>FORM</auth-method>
  <realm-name>My Club Members-only Area</realm-name>
  <form-login-config>
    <form-login-page>/login.html</form-login-page>
    <form-error-page>/error.html</form-error-page>
  </form-login-config>
</login-config>
```

The /login.html and /error.html in the above example refer to files relative to the root of the web application. The form-login-page element indicates the page that Tomcat displays to the user when it detects that a user who has not logged in is trying to access a resource that is protected by a security-constraint. The form-error-page element denotes the page that Tomcat displays when a user's login attempt fails.

Client-cert authentication:

The user is verified by a client-side digital certificate. The client-cert (CLIENT-CERT in the web.xml file's auth-method element) method of authentication is available only when you're serving content over SSL (i.e., HTTPS). It allows clients to authenticate without the use of a password instead, the browser presents a client-side X.509 digital certificate as the login credential. Each user is issued a unique digital certificate that the web server will recognize. Once users import and store their digital certificates in their web browsers, the browsers may present them to the server whenever the server requests them. If you want to know more about this, please refer to relating books or online manual of Tomcat.

Conclusion:

This segment of the report sums up the primary finishes of the job of JSM with Apache tomcat. A survey of tomcat security gives the security related issues of tomcat in detail. What's more, the web application Book Store is utilized to distinguish the running condition of Tomcat, and furthermore with Java Security Manager and without. By arranging the JSM with Tomcat may make to control and to usage of various security arrangements. Every security arrangement gives

extraordinary kind of security to the web application. To investigate the job of JSM, some experiments are composed and afterward tried effectively. Despite the fact that the execution and work of JSM with tomcat has not been totally tried at this point however in this proposed venture attempted to investigate the Role of JSM with tomcat by utilizing distinctive experiments. And furthermore, it very well may be feeble against assaults, for example, setting off the boundless circles and devour the majority of the CPU memory and significant time.

Future scope:

In the present proposed venture, we attempted to cover all security related issues about Tomcat and its arrangements and how to make better make sure about condition for Tomcat. The examination to dissect the Security trough in Tomcat requires a ton of time and test work. Inside the restricted time limitation, we got a few outcomes which are useful to break down the job of Security Manager. In this venture we executed SSL however it will require some more opportunity to work impeccably. Furthermore, we attempted to add the some additional fates to current web application, for example, Visa security. Inside the time limitation we tried different record consent get to controls, for example, framework properties. We utilized a few strategies and made some noxious pages which are helpful to test the continuous assaults. For the current idea of Tomcat security and Java Security Manager ought to require a great deal of research. On the off chance that we get some additional time, we can test some more experiments.

References:

[1] ACCESS DENIED. DFS Issue 55. http:// textfiles.com/magazines/DFS/dfs055.txt, 1996.

[2] AERASEC NETWORK SERVICES AND SECURITY GMBH. Decompression Bomb Vulnerabilities. http://www. aerasec.de/security/advisories/ decompression-bomb-vulnerability.html, 2009.

[3] ALUR, D., MALKS, D., AND CRUPI, J. Core J2EE Patterns: Best Practices and Design Strategies. Prentice Hall PTR, 2001.

[4] ANTUNES, J., NEVES, N., AND VERISSIMO, P. Detection and prediction of resource-exhaustion vulnerabilities. In ISSRE 2008 (Nov 2008), pp. 87–96.

[5] BURNIM, J., JUVEKAR, S., AND SEN, K. Wise: Automated test generation for worst-case complexity. In ICSE 2009 (May 2009), pp. 463–473.

[6] BUSCHER ¨ , A., AND HOLZ, T. Tracking DDoS Attacks: Insights into the business of disrupting the Web. In Proc. LEET (2012).

[7] BUTKO, A. JSONRPC Java implementation. https://code.google.com/p/libjsonrpc/, 2014.

[8] CARNEGIE MELLON UNIVERSITY. Project Cyrus. https://cyrusimap.org/, 2014.

[9] CAZI, M. CSJSONRPC - A PHP JSON-RPC Server. https://github.com/mojtabacazi/ CSJRPC, 2014.

[10] CHANG, R., JIANG, G., IVANCIC, F., SANKARANARAYANAN, S., AND SHMATIKOV, V. Inputs of coma: Static detection of denial-of-service vulnerabilities. In CSF '09 (July 2009), pp. 186–199.

[11] CRISPIN, M. Internet Message Access Protocol - Version 4rev1. RFC 3501 (Proposed

Standard), Mar. 2003. Updated by RFCs 4466, 4469, 4551, 5032, 5182, 5738, 6186, 6858.

[12] CROSBY, S. A., AND WALLACH, D. S. Algorithmic DoS. In Encyclopedia of Cryptography and Security (2nd Ed.), H. C. A. van Tilborg and S. Jajodia, Eds. Springer, 2011, pp. 32–33.

[13] DEUTSCH, P. DEFLATE Compressed Data Format Specification version 1.3. RFC 1951 (Informational), May 1996.

[14] DIERKS, T., AND ALLEN, C. The TLS Protocol Version 1.0. RFC 2246 (Proposed Standard), Jan. 1999. Obsoleted by RFC 4346, updated by RFCs 3546, 5746, 6176.

[15] DILLEY, B. JSON-RPC for Java. https:// github.com/briandilley/jsonrpc4j, 2014.