Understanding Tomcat Security

In the rush to bring products and services online, security is typically the one area that gets the least attention, even though it is arguably the most important. The reasons for neglecting security vary. In some cases, implementing good security is seen as too much work in a compressed timeline. In other cases, an organization chooses to rely on the security features built into another product, such as an operating system or application server, rather than enhance that security with additional steps.

Even if your system doesn't deal with financial transactions, security should still be a high priority. Security isn't just about protecting money anymore. In the Internet age, many other issues require strong security, from personal privacy to medical information. Organizations that have well-established privacy policies and have a track record of taking security seriously, often attract more customers and clients than those who don't.

The Apache Project's web servers are the most popular web servers in the world with over 63% of web sites, according to the ongoing survey from Netcraft: http://www.netcraft.com/Survey. That doesn't even include the number of private web servers in use, such as corporate intranets and personal desktops. While the Apache numbers are obviously due to the popularity of the Apache HTTP server, Apache Tomcat is popular in its own right, with millions of downloads since its release. In addition, many web servers use Tomcat as the servlet container of choice, so while a survey such as Netcraft's can only determine the actual server serving the request, there could be any number of Apache Tomcat installations supplying the actual content. Thus, devoting resources and effort to securing your Apache Tomcat installation is just as important as securing your applications, your operating systems, and your networks.

Vulnerability Overview

Web applications are complex systems by definition. Several different components are needed to make a complete system, including:

- The operating system and its file systems
- The web server itself, serving HTTP requests
- A dynamic content engine, such as Tomcat, executing the application and assembling responses
- A database
- A network or networks
- The Java Virtual Machine (JVM) in the case of applications designed with servlets and JSPs
- Any-third party packages or libraries

A vulnerability can be found in any one of these components. Even more frustrating, a specific combination of components can have a vulnerability not found in another combination. For example, a vulnerability found when using Tomcat with a certain relational database may not be an issue if a different database was used. Likewise, a problem with one version of a component might not be a problem with an older or newer version.

Ten Most Common Security Problems

The Open Web Application Security Project (OWASP) is an Open Source community project staffed entirely by volunteers. The group has highlighted the ten most common vulnerabilities in web applications and services, noting that these problems are as serious as network security problems and deserve just as much attention.

Vulnerability	Description
Unvalidated Parameters	Information from web requests is not validated before being used by a web application. Attackers can use these flaws to attack backend components through a web application.
Broken Access Control	Restrictions on what authenticated users are allowed to do are not properly enforced. Attackers can exploit these flaws to access other users' accounts, view sensitive files, or use unauthorized functions.

Vulnerability	Description
Broken Account and Session Management	Account credentials and session tokens are not properly protected. Attackers that can compromise passwords, keys, session cookies, or other tokens can defeat authentication restrictions and assume other users' identities.
Cross-Site Scripting (XSS) Flaws	The web application can be used as a mechanism to transport an attack to an end user's browser. A successful attack can disclose the end user's session token, attack the local machine, or spoof content to fool the user.
Buffer Overflows	Web application components in some languages that do not properly validate input can be crashed and, in some cases, used to take control of a process. These components can include CGI, libraries, drivers, and web application server components.
Command Injection Flaws	Web applications pass parameters when they access external systems or the local operating system. If an attacker can embed malicious commands in these parameters, the external system may execute those commands on behalf of the web application.
Error Handling Problems	Error conditions that occur during normal operation are not handled properly. If an attacker can cause errors to occur that the web application does not handle, they can gain detailed system information, deny service, cause security mechanisms to fail, or crash the server.
Insecure Use of Cryptography	Web applications frequently use cryptographic functions to protect information and credentials. These functions and the code to integrate them have proven difficult to code properly, frequently resulting in weak protection.
Remote Administration Flaws	Many web applications allow administrators to access the site using a web interface. If these administrative functions are not very carefully protected, an attacker can gain full access to all aspects of a site.
Web and Application Server Misconfiguration	Having a strong server configuration standard is critical to a secure web application. These servers have many configuration options that affect security and are not secure out of the box.

The OWASP Top Ten list is a "living" document. It can and will change over time. To get the latest version, and to participate in the project itself, visit the OWASP web site at http://www.owasp.org/.

It's important to remember that the list is not specific, nor can it be. Each web application is different, so instead of thinking in terms of specific problems, it is more effective to think in terms of the types of problems, as shown in the list. All web applications suffer from these types of vulnerabilities, and it is up to the development team, the systems administration team, and the organization supporting them, to make correcting these problems a priority. Above all, remember that your web application is a critical part of your security model. Web application security techniques are discussed in Chapter 6. For now, let's address the last item in the list: "Web and Application Server Misconfiguration."

Known Tomcat Vulnerabilities

Tomcat, like any other application, is not bug free. A fundamental part of any security policy is not only staying abreast of known vulnerabilities, usually through a mailing list like the **BUGTRAQ** list or one of many others, but also staying current with recent patch levels and versions of the software. Since Tomcat's version 4 was released, several vulnerabilities have been found and resolved. As of this writing (January 2003), the recommended versions of the Tomcat 4 branch are 4.0.6 and 4.1.18. The 4.0 branch is no longer actively developed, and work on Tomcat 5 has already started. If you are unable to use Tomcat 4 for some reason, the recommended version is 3.3.1.

Servlet/JSP Spec	Tomcat version
2.4/2.0	5.0.0 Alpha
2.3/1.2	4.1.18
2.2/1.1	3.3.1

Even if you have the latest version of Tomcat installed, you still need to worry about potential vulnerabilities. In a web server environment, Tomcat is only one piece of the entire system, so while you may have the latest security patches for Tomcat itself, your system can still fall prey to attacks and exploits that target web applications or other external system components.

Recently discovered Tomcat vulnerabilities have focused on problems with Tomcat installation defaults, such as the Invoker servlet being enabled by default, allowing an attacker to craft a URL that will display the source code of a JavaServer Page instead of the content that the page generates. More on web application vulnerabilities can be found in Chapter 6, Web Application Security.

Here's a list of the known vulnerabilities in Tomcat 4 through January 2003:

Date Announced	Vulnerability
2003-01-09	Invoker servlet file disclosure vulnerability
2002-12-10	Default Servlet file disclosure vulnerability
2002-12-09	Mod_jk chunked encoding denial of service vulnerability
2002-08-21	JSP request cross site scripting vulnerability
2002-06-12	JSP engine denial of service vulnerability
2002-04-23	Servlet path disclosure vulnerability
2002-04-19	System path information disclosure vulnerability

As you can see, the majority of the problems found are application or configuration oriented, not necessarily problems with Tomcat itself, such as a buffer overflow. All vulnerabilities found so far have known workarounds and solutions; it is up to the administrator to implement them in a timely manner. Any solutions to these vulnerabilities not covered by upgrading to the latest version of Tomcat are discussed in upcoming chapters.

For future reference, you can subscribe to various mailing lists to stay abreast of vulnerabilities as they are found. Two popular mailing lists are the *BUGTRAQ* mailing list, and the *Web Application Security* mailing list, known as **webappsec**. Information on both, including charters, archives, and subscription information, can be found at http://online.securityfocus.com/archive. In addition, vulnerability databases can be found at http://online.securityfocus.com/bid and http://www.kb.cert.org/vuls. The CERT® Coordination Center at Carnegie Mellon University is also an excellent source of Internet-related security information, including up-to-date alerts and status reports on the newest vulnerabilities. The CERT® Coordination Center web site can be found at http://www.cert.org.

Installation

Right from the start, the decisions you make when installing Tomcat can have a significant impact on the security of your installation. From a pre-installation planning perspective, you will
want to consider:

- How will starting and stopping Tomcat be controlled?
- Which user account is used to run Tomcat?

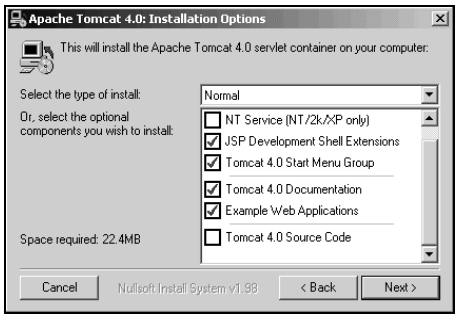
- Which ports will Tomcat use, and how will access to those ports be controlled?
- What are Tomcat installation defaults, and do they fit our security model?

Running Tomcat as a Service

Like most applications designed to provide services for many users and accept requests from other applications, Tomcat can be used in one of two ways. First, Tomcat can run as a standard user-level application. In other words, users log into their account, start up Tomcat, do their work, shut Tomcat down, and log out. In a single-developer/single-desktop scenario, this may be ideal and all that is needed. The second way is to run Tomcat as a service. By running it as a service, you are making it available to serve requests to more than just the person logged in, and you are making sure that Tomcat is available for those requests whether anyone is logged in or not. For a web server, this is the standard setup.

If you're using Linux or another type of UNIX-like operating system, you'll want to set up a script that starts and stops Tomcat as needed. Typically, this script would be called in the system's boot routine, and called again when the system shuts down. This allows Tomcat to be available to serve requests as much as possible. By including this script in your system's `init` configuration, Tomcat will be available to accept requests whenever the server is up.

If you're installing Tomcat on Microsoft's Windows operating system, you'll want to install Tomcat as a Windows service so that it stays running without requiring that a user stay logged in while it's running. Depending on the install method that you use for Tomcat, there are two different ways to set up Tomcat as a service on Windows. The first is the easiest: if you download the Tomcat self-installer package, there will be an option to install Tomcat as a service when you run it. This can be seen in the screenshot below:



By selecting the box labeled NT Service (NT/2k/XP only), the installer will take care of setting up Tomcat as a Windows service instead of as a stand-alone application controlled by a typical user. If, on the other hand, you installed Tomcat from a ZIP archive or even built it yourself from source, you can still install it as a service after the fact.

Note that installing and uninstalling Tomcat as service is not the same as starting and stopping Tomcat. You only need to do the service installation once. After that, you can use the Services control applet or the command line to start and stop your Tomcat service as needed. Running Tomcat as a service on Windows and UNIX-like platforms is discussed in Chapter 2. For more detailed information on installing Tomcat, consult *Professional Apache Tomcat* (*ISBN 1-861007-73-6)* from *Wrox Press*.

Using an Unprivileged User Account

One of the problems encountered when running web servers and other publicly accessible applications is the concept of permissions. While your application might use access controls to limit what certain users can do, that only applies to the application. What about Tomcat itself? Buffer overflows, probably the most common exploit, highlight the problems with running server applications under user accounts that have total access to the system. For example, if a web server is running as `root`, administrator, or superuser, and an attacker is able to send a request such that the web server fails and allows arbitrary commands to run within its environment, those commands will run with a superuser access.

The solution is to avoid using a privileged account to run Tomcat whenever possible. By using an unprivileged user account, you can take the first step to protecting against a future exploit that involves causing Tomcat to allow the execution of arbitrary commands issued by an attacker. If Tomcat, running as an *unprivileged* user, were to fail or for some reason allow an attacker access to the operating system, the only permission level the attacker would typically have would be the permission level of the Tomcat user, instead of administrator or superuser access.

Different operating systems have different ways of denoting the superuser account, or administrator account. On UNIX variants, the administrator account is known as `root`. On Windows systems, the administrator account is known as `SYSTEM`. UNIX-like operating systems further complicate matters by requiring that all services listening on ports less than 1024 run as `root`. This poses a problem, since popular Internet services like e-mail, web browsing, and file transfers all occur on ports less than 1024 by default. If one of those services was attacked and failed to handle the attack appropriately, an attacker might gain access to that service's environment, in this case an administrator environment. The *less than 1024* rule doesn't apply to Windows servers, since services run as `SYSTEM` by default, regardless of the port being used.

So, how do we use Tomcat as a stand-alone web server on port 80? By default, Tomcat listens on port 8080 for HTTP requests. To get Tomcat to listen on port 80, the default HTTP port, you have to do two things:

1. Change the `org.apache.coyote.tomcat4.CoyoteConnector` on port 8080 in `server.xml` to port 80
2. Start, stop, and run Tomcat as `root` if you're using a UNIX-like operating system.

As discussed earlier, running publicly available web services as `root` or superuser is typically a bad idea, so the solution is to avoid using Tomcat as a stand-alone web server on port 80 by integrating it with a standard HTTP web server such as Apache, Microsoft's Internet Information Server, or Sun Microsystem's iPlanet. In the case of Apache, the *less than 1024* port restriction is handled nicely by Apache starting up as `root`, binding to port 80, and then forking child processes to handle actual HTTP requests. These child processes do not run as `root`, but as some other unprivileged user, typically a user account known as `nobody`, set up specifically for Apache's use. Creating and running Tomcat with unprivileged user accounts on Windows and UNIX-like platforms is discussed in Chapter 2. For more detailed information on integrating Tomcat with a HTTP server, consult *Professional Apache Tomcat* (*ISBN 1-861007-73-6)* from *Wrox Press*.

Using a Firewall

While a detailed discussion of firewalls is beyond the scope of this book, they can play a key role in protecting your Tomcat installation, so we'll take a quick look at the basics: what they do, why they're important, and some simple examples of how you can use them.

Simply put, a firewall protects a computer or computer network from unauthorized access. A firewall can be software, hardware, or a combination of both. There are different types: proxy servers, packet filters, and stateful inspection firewalls. Firewalls use rules to determine which network traffic should be allowed to pass, and which shouldn't. These rules can be very simple, such as "deny all traffic to port 23," or complex, such as "deny all traffic to port 3389, unless it comes from a certain list of IP addresses, and arrives during a certain part of the day but only Monday through Friday." The rules can also determine whether certain traffic is logged or not.

Most environments include a firewall that is a distinct component from the server hosting Tomcat. Typically, the firewall is located between the public, untrusted Internet, and the trusted application network. This location allows the firewall to analyze all traffic as it passes in and out. The problem with relying on this firewall for protection of your Tomcat installation is that by default, this firewall has to include rules that apply to all nodes in the network that it is protecting. If the only node it is protecting is the machine hosting Tomcat, all is good. If the firewall is protecting a number of different networks, and dozens or even hundreds of nodes, it is possible that the firewall rule set may not be restrictive enough to provide the protection you desire for your Tomcat installation. In this scenario, you have additional tools you can use to add layers of protection.

Linux kernels since version 2.0 have included support for firewall rules. The 2.0 kernels used a tool called `ipfwadm`, while the 2.2 kernels used `ipchains`. The 2.4 kernels have yet another framework for setting up firewalls, and that is through `netfilter` and `iptables`.

Let's take a look at a simple firewall rule. Assume that the environment is Tomcat integrated with the Apache HTTP server using a web server connector such as JK or JK2. By default, Tomcat will use `org.apache.coyote.tomcat4.CoyoteConnector` to listen on port 8009 for communications from Apache. In a single server scenario, that means that traffic for port 8009 should never be received from any other machine at any other address besides localhost, or 127.0.0.1. Leaving a connector port, such as 8009, open to any traffic from anywhere is inviting trouble. So, on a Linux system, a rule for rejecting traffic on port 8009 from any address but 127.0.0.1 might look like this:

```
ipchains -A input -s ! 127.0.0.1 -d 0/0 8009 -p all -y -j REJECT
```

Going from left to right, this particular rule says, "add an input rule for all traffic from any host except localhost, destined for port 8009 on any address, any protocol, and reject it." By implementing a rule like this on the server itself, you can add one or more layers of protection to your installation beyond the protection provided by an external firewall.

The rule shown above is just an example. Your environment and requirements might need different rules, or you might not want to use a firewall on your server at all. In a load-balancing environment, your Tomcat installation might receive requests on port 8009 from any number of IP addresses on your local network. In that case, you would change the rule to use a different source parameter, like this:

```
ipchains -A input -s ! 192.168.1.0/24 -d 0/0 8009 -p all -y -j REJECT
```

As shown, the rule would reject traffic destined for port `8009` unless it came from the `192.168.1.0` Class C network, which might be the addresses used by your Apache servers. A rule for `iptables` would be similar. Either way, the point is that using a local firewall or packet filter can add protection to your Tomcat installation without requiring Tomcat to handle the network traffic.

Firewalls and packet filters are worthy of a book themselves and can't be covered thoroughly here. For more information on `ipchains` and `iptables`, consult the manual pages and documentation, or check the HOWTOs and FAQs at http://www.netfilter.org/. If you're using a UNIX variant besides Linux, consult your vendor's product pages or investigate open source options. Many different solutions exist, especially for open source operating systems like FreeBSD and OpenBSD.

If you're using a Windows server, you will need to install and configure Microsoft's Internet and Security Acceleration (ISA) server or investigate a third-party firewall package for your server, as there is no firewall or packet filter built into the Windows server kernel. For more information on using Microsoft's ISA server, consult the Microsoft MSDN HOWTO articles for ISA Server: http://msdn.microsoft.com/howto/security.asp or the ISA Server homepage at http://www.microsoft.com/isaserver/.

Connector Management

In Tomcat parlance, a *connector* is an implementation of a particular Java class that is configured to listen on a specific port with varying parameters, such as timeout or the maximum number of connections. There are two different types of connectors: those that allow browsers to connect directly to Tomcat; and those that listen for and handle traffic from web servers like Apache or Microsoft's IIS. For example, a connector of the first type would be the default HTTP connector on port 8080. A connector of the second type would be the default JK/JK2 connector on port 8009. From the start, Tomcat comes with some connectors enabled in `server.xml`, and several disabled.

For Tomcat 4.1.18, the default connectors and their initial statuses are:

Port Number	Connector Type	Default Status
8080	HTTP/1.1 `org.apache.coyote.tomcat4.CoyoteConnector`	Enabled
8443	SSL HTTP/1.1 `org.apache.coyote.tomcat4.CoyoteConnector`	Disabled
8009	JK/JK2 AJP 1.3 `org.apache.coyote.tomcat4.CoyoteConnector`	Enabled
8009	Legacy JK AJP 1.3 `org.apache.ajp.tomcat4.Ajp13Connector`	Disabled
8082	Proxied HTTP/1.1 `org.apache.coyote.tomcat4.CoyoteConnector`	Disabled
8083	Legacy HTTP/1.1 `org.apache.catalina.connector.http.HttpConnector`	Disabled
8084	Legacy SSL HTTP/1.1 `org.apache.catalina.connector.http.HttpConnector`	Disabled
8008	WARP `org.apache.catalina.connector.warp.WarpConnector`	Disabled

Once you have Tomcat installed, you will want to review `server.xml` and decide which connectors you need, and which you don't. Disable the connectors you don't need by commenting them out in `server.xml` and restarting Tomcat. The `server.xml` excerpt below shows the connector on port 8080 enabled:

```
<!-- Define a non-SSL Coyote HTTP/1.1 Connector on port 8080 -->
<Connector className="org.apache.coyote.tomcat4.CoyoteConnector"
           port="8080" minProcessors="5" maxProcessors="75"
           enableLookups="true" redirectPort="8443"
           acceptCount="100" debug="0" connectionTimeout="20000"
           useURIValidationHack="false" disableUploadTimeout="true" />
```

And the one below shows a disabled connector:

```
<!-- Define a non-SSL Coyote HTTP/1.1 Connector on port 8080 -->
<!--
<Connector className="org.apache.coyote.tomcat4.CoyoteConnector"
           port="8080" minProcessors="5" maxProcessors="75"
           enableLookups="true" redirectPort="8443"
           acceptCount="100" debug="0" connectionTimeout="20000"
           useURIValidationHack="false" disableUploadTimeout="true" />
 -->
```

At a minimum, you will need at least one connector. If you disable all connectors, Tomcat will be unable to listen for requests. If you're using Tomcat in stand-alone mode as its own HTTP server, then you'll need the HTTP/1.1 connector on port 8080 or 80, and you won't need the JK/JK2 connector on port 8009. Likewise, if you are using Tomcat with a HTTP server and not stand-alone, you won't need the HTTP/1.1 connector on port 8080. Instead, you would use the JK/JK2 connector on port 8009 or some other port of your choosing, or even the WARP connector on port 8008. For more on configuring and using the JK/JK2 and other connectors in `server.xml`, consult *Professional Apache Tomcat* (*ISBN 1-861007-73-6*) from *Wrox Press*.

If you don't disable the connectors you aren't using, you allow Tomcat to receive traffic on that port. An attacker could send all kinds of different packet types to that port, looking for a vulnerability, and you would never know it was happening since it wouldn't be interfering with connectors listening on other ports. In short, make sure Tomcat is only listening on the ports you want it to use.

Default Tomcat Applications

Just as Tomcat has various connectors enabled by default, it also has several web applications included and enabled by default. These applications consist of the following:

- The Manager application
- The Admin application
- The WebDAV application
- The examples application

The applications are there for a reason, but that doesn't mean you have to sit back and accept the way they are installed or the way they are configured by default. Analyzing the default applications, determining which ones you need and which ones you don't, and taking appropriate action might make the difference between being susceptible to a vulnerability down the road, and being protected.

The Manager Application

The Manager application included with Tomcat is designed for ease of deployment and management of web applications without having to affect the entire servlet container.

With the Manager application, you can:

- Deploy a new web application, on a specified context path, from the uploaded contents of a WAR file
- Install a new web application
- List the currently deployed web applications, as well as the sessions that are currently active for those web apps
- Reload an existing web application, to reflect changes in the contents of `/WEB-INF/classes` or `/WEB-INF/lib`
- List the available global JNDI resources
- List the available security roles defined in the user database
- Remove an installed web application, start and stop an application, and undeploy an application

At first glance, it might seem like the Manager application is not included with your Tomcat installation. That's because the `Context` element for `/manager` is not included in `server.xml`. The Manager application uses the Automatic Deployment feature of Tomcat, and is deployed as a valid `Context` at startup because of the `manager.xml` file in the `webapps` directory of your Tomcat installation. `Manager.xml` looks like the following:

```
<Context path="/manager" docBase="../server/webapps/manager"
        debug="0" privileged="true">

  <!-- Link to the user database we will get roles from -->
  <ResourceLink name="users" global="UserDatabase"
                type="org.apache.catalina.UserDatabase"/>

</Context>
```

If you'd rather have the `Context` for the Manager application in your `server.xml`, you can do this simply by copying the contents of `manager.xml` into `server.xml` and restarting Tomcat. However, this will make management for many applications and virtual hosts in Tomcat harder. By taking advantage of the Automatic Deployment feature of Tomcat, you can install the Manager application for many virtual hosts simply by copying `manager.xml` to each host's `webapps` folder.

While the Manager application is available by default, access to it is disabled by default. The Manager application requires a user account with a role of **manager** in `tomcat-users.xml`, which is located in Tomcat's `conf` directory. The `tomcat-users.xml` file does not include a user with Manager access permissions out of the box. To enable Manager application access for a specific user, edit `tomcat-users.xml` so that is looks like the following:

```
<?xml version='1.0' encoding='utf-8'?>
<tomcat-users>
  <role rolename="tomcat"/>
  <role rolename="role1"/>
  <role rolename="manager"/>
  <role rolename="admin"/>
  <user username="tomcat" password="tomcat" roles="tomcat"/>
  <user username="role1" password="tomcat" roles="role1"/>
  <user username="both" password="tomcat" roles="tomcat,role1"/>
  <user username="admin" password="tomcat" roles="admin,manager"/>
</tomcat-users>
```

Note the role of `manager` added to the user account `admin`. Obviously, in your own installation, you will want to change the password from `tomcat` to something else. You can also use JDBC Realms and JNDI Realms to hold your user account and permission information if you want to integrate with existing authentication systems and avoid having to edit `tomcat-users.xml`.

If you wanted to further restrict access to the Manager application, you could do so by restricting access to a certain IP address or host name by using a `Valve`. For example, by adding a `RemoteAddrValve` or `RemoteHostValve` to `manager.xml`, you could restrict access to only those computers on a local network or even to the Tomcat server itself. In the example below, access to the `/manager Context` is restricted to requests from IP address `127.0.0.1`:

```
<Context path="/manager" debug="0" privileged="true"
         docBase="/usr/local/kinetic/tomcat4/server/webapps/manager">
         <Valve className="org.apache.catalina.valves.RemoteAddrValve"
                allow="127.0.0.1"/>
</Context>
```

Once you have a user account with the appropriate permissions set up in `tomcat-users.xml`, and any desired address or hostname restrictions in place using a `Valve`, you can access the manager application in one of three ways:

- Via a web browser using a URL like: `http://localhost/manager/html/`
- Via HTTP requests, suitable for use in scripts set up by system administrators
- As a set of task definitions for the Ant (version 1.4 or later) build tool

For more information on the Manager application, consult the Manager HOWTO at: http://jakarta.apache.org/tomcat/tomcat-4.1-doc/manager-howto.html. For more on creating users and roles, and using alternate methods to `tomcat-users.xml`, consult Chapter 4.

The Admin Application

Similar to the Manager application, Tomcat 4 and 5 include an administration application by default. Just as is the case with the Manager application, the Admin application uses a file called `admin.xml` in Tomcat's `webapps` directory to auto-deploy the `/admin` `Context` instead of including the `Context` in `server.xml`.

The contents of `admin.xml` look like this:

```
<Context path="/admin" docBase="../server/webapps/admin"
        debug="0" privileged="true">

  <!—
  <Valve className="org.apache.catalina.valves.RemoteAddrValve"
    allow="127.0.0.1"/>
  -->

  <Logger className="org.apache.catalina.logger.FileLogger"
             prefix="localhost_admin_log." suffix=".txt"
           timestamp="true"/>

</Context>
```

As you can see, it looks very similar to `manager.xml`, including the use of a `Valve` to restrict access to a specific IP address or range of addresses. The value of `allow` in the `Valve` can be a comma-delimited list of IP addresses, or even a regular expression such as `192.168.1.*`.

What's the difference between Admin and Manager? The Manager application is for managing web application deployment. The Admin application is for managing the server itself. With the Admin application, you can do the following:

- Add and delete `Connectors`, as well as change their parameters
- Add, delete, and manage `Hosts`
- Add, delete, and manage `Contexts` within `Hosts`
- Manage `Resources`, such as `DataSources` and other `Environment` parameters
- Manage `Users` and `Roles`

In short, rather than editing `server.xml` directly, you can accomplish the same tasks by using the Admin application. It goes without saying, then, that access to the Admin application should be monitored carefully and granted judiciously. Having access to the Admin application is essentially the same as being able to edit `server.xml`.

Just like the Manager application, you can grant access to the Admin application using `tomcat-users.xml`. While the Manager application requires a user account with a role of `manager`, the Admin application requires a user account with the role of **`admin`**. In the `tomcat-users.xml` file for our Manager example, for instance, the user `admin` has both *admin* and *manager* privileges.

By default, the Admin application can be reached at the following URL for your Tomcat installation: `http://localhost:8080/admin/`. Once you've considered the ramifications of changing the defaults, you can modify `admin.xml` and `tomcat-users.xml` to grant access to the Admin application to other users and hosts. For more on creating users and roles, and using alternate methods to `tomcat-users.xml`, consult Chapter 4.

The examples Application

By default, Tomcat includes a set of JavaServer Pages and servlets as examples of how to use Tomcat to serve web applications. In a production situation, the `/examples Context` is unnecessary, and should be disabled. If left enabled, an attacker can use some of the examples as a means of discerning information about the Tomcat installation, such as pathnames and other information, all of which is better left hidden.

Disabling the `/examples Context` is easy. Simply comment out that `Context` in `server.xml`, or delete it entirely. In fact, Tomcat 4.1.18 comes with a second `server.xml` file called `server-noexamples.xml.config`. This file is identical to `server.xml`, but without the `/examples Context`. Simply rename the default `server.xml` to `server-examples.xml.config`, and then copy `server-noexamples.xml.config` to `server.xml` and restart Tomcat. You now have a default Tomcat installation without the `/examples Context`.

Because of the Automatic Deployment feature, performing the operation above won't have any effect on the Manager or Admin applications. Both will still be available to the hostnames where they're installed.

The WebDAV Application

The fourth and final application included with Tomcat 4 is the WebDAV application. WebDAV is a method of allowing **remote authoring** of a web site using a compatible WebDAV client. Tomcat 4.1.18 supports WebDAV 2, and compatible clients include:

- Adobe GoLive 5.0 (and other WebDAV-enabled Adobe products like Photoshop)
- Internet Explorer 5 and 5.5 (Windows 2000)
- Jakarta Slide 1.0 WebDAV client library
- Office 2000 (Windows 2000)

By default, the `/webdav Context` is distributed with **read-only** access enabled. In order to use WebDAV for a particular `Context`, you'll need to make it **read-write** instead. For example, to make the `/webdav Context` included with Tomcat 4.1.18 read-write instead of read-only, you would edit the `web.xml` file found in `webapps/webdav/WEB-INF/` and change the following:

```
<!--
    <init-param>
      <param-name>readonly</param-name>
      <param-value>false</param-value>
    </init-param>
-->
```

to this:

```
    <init-param>
      <param-name>readonly</param-name>
      <param-value>false</param-value>
    </init-param>
```

Also in `web.xml`, you can configure particular roles as listed in `tomcat-users.xml`, granting them particular access to the `/webdav Context`. For example, to enable access to `/webdav` for the user `tomcat` as defined in `tomcat-users.xml`, you would set up the following in `web.xml`:

```
  <security-constraint>
    <web-resource-collection>
      <web-resource-name>The Entire Web Application</web-resource-name>
      <url-pattern>/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
      <role-name>tomcat</role-name>
    </auth-constraint>
  </security-constraint>

  <login-config>
    <auth-method>BASIC</auth-method>
    <realm-name>Tomcat Supported Realm</realm-name>
  </login-config>

  <security-role>
    <description>
      An example role defined in "conf/tomcat-users.xml"
    </description>
    <role-name>tomcat</role-name>
  </security-role>
```

Once you have granted and enabled read-write access, you can reach the `/webdav Context` by pointing to the `/webdav` path of your Tomcat installation using your WebDAV client.

Tomcat 4 uses the Jakarta Slide WebDAV module, implemented as a servlet, to provide WebDAV services. For more info on Jakarta Slide, visit http://jakarta.apache.org/slide/index.html and for more info on WebDAV itself, visit http://www.webdav.org/. For other links and more information on Tomcat's implementation of WebDAV, consult the http://localhost:8080/webdav/index.html URL of your Tomcat installation.

Summary

As pointed out by the Open Web Application Security Project, misconfiguration and improper installation of web and application servers can be a serious problem and adversely affect the integrity of your web application and its components. In addition, neglecting to analyze things like remote administration applications that may be included can be another cause for problems. In this chapter we discussed:

- The top ten web application vulnerabilities as listed by the Open Web Application Security Project
- Proper installation of Tomcat as a service running under an unprivileged user account
- Using a local firewall to add additional layers of security to network traffic
- Managing `Connectors` in `server.xml` so only those needed are enabled
- Proper management of default applications and `Contexts` included with Tomcat to minimize possible entry points for attacks

In short, it is up to the server administrator to be proactive and manage the default installation of any web or application server under their control. By merely accepting the defaults, an administrator leaves themselves open to more potential attack points and may even compromise their installation, since they are simply trusting someone else to ship the product with secure defaults.