HP OpenView Select Access

Integration Guide for Compiling Apache with Tomcat and/or SSL-enabled systems

Software Version: 6.0

for HP-UX, Linux, Solaris, and Windows operating systems

March 2004

Legal Notices

Warranty

Hewlett-Packard makes no warranty of any kind with regard to this document, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Hewlett-Packard shall not be held liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.

A copy of the specific warranty terms applicable to your Hewlett-Packard product can be obtained from your local Sales and Service Office.

Restricted Rights Legend

Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause in DFARS 252.227-7013.

Hewlett-Packard Company
United States of America

Rights for non-DOD U.S. Government Departments and Agencies are as set forth in FAR 52.227-19(c)(1,2).

Copyright Notices

No part of this document may be copied, reproduced, or translated into another language without the prior written consent of Hewlett-Packard Company. The information contained in this material is subject to change without notice.

HP OpenView Select Access includes software developed by third parties. The software HP OpenView Select Access uses includes:

- The OpenSSL Project for use in the OpenSSL Toolkit.
- Cryptographic software written by Eric Young.
- Cryptographic software developed by The Cryptix Foundation Limited.
- JavaService software from Alexandria Software Consulting.
- Software developed by Claymore Systems, Inc.
- Software developed by the Apache Software Foundation.
- JavaBeans Activation Framework version 1.0.1 © Sun Microsystems, Inc.
- JavaMail, version 1.2 © Sun Microsystems, Inc.
- SoapRMI, Copyright © 2001 Extreme! Lab, Indiana University.
- cURL, Copyright © 2000 Daniel Stenberg.
- Protomatter Syslog, Copyright © 1998-2000 Nate Sammons.
- JClass LiveTable, Copyright © 2002 Sitraka Inc.

For expanded copyright notices, see HP OpenView Select Access's `<install_path>/3rd_party_license` directory.

Trademark Notices

- Intel® and Pentium® are registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.
- Java™ is a US trademark of Sun Microsystems, Inc.
- Linux is a U.S. registered trademark of Linus Torvalds.
- Microsoft®, Windows®, and Windows NT® are U.S. registered trademarks of Microsoft Corporation.
- Oracle® is a registered US trademark of Oracle Corporation, Redwood City, California.
- UNIX® is a registered trademark of The Open Group.

Support

Please visit the HP OpenView Select Access web site at:

http://www.openview.hp.com/products/select/index.html

There you will find contact information and details about the products, services, and support that HP OpenView Select Access offers.

You can also go directly to the HP OpenView support web site at:

http://support.openview.hp.com/

The support site includes:

- Downloadable documentation
- Troubleshooting information
- Patches and updates
- Problem reporting
- Training information
- Support program information
- Security bulletins

Contents

Chapter 1
About this Integration Paper

What is it about?

This Integration Paper describes how to integrate Tomcat with Select Access.

Table 1 is an overview of this document's contents.

Table 1: Integration Paper overview

This chapter...	Covers these topics...
Chapter 2, *Technologies overview*	• Introduces Select Access: what it is, what it does, and how it works. • Introduces Tomcat: what it is and what integration issues exist.
Chapter 3, *Integrating Select Access with Tomcat over Apache*	Describes what you need to do with Tomcat and Select Access to integrate these technologies with an Apache Web server.

Who is it for?

This Integration Paper is intended to instruct individuals or teams responsible for:

- Integrating Select Access with the Tomcat servlet over Apache.
- Using Select Access to manage access to dynamic content that is dynamically created by the Tomcat servlet engine.

What does it assume you already know?

This Integration Paper assumes a working knowledge of:

- ***Select Access***—Particularly Select Access's Enforcer plugin technology. This ensures that you understand how integration with Tomcat affects the Select Access components.

- ***Servlets*—** A background of servlets and the principals on which they work help you to predict what particular issues exist for your specific deployment.
- ***LDAP directory servers*—** Helps ensure that information in the Policy Builder is set up correctly.
- ***Building modules/libraries on Apache*—** Familiarity with the build process on different operating systems (that is, Linux and Solaris), ensures you know the basics required to complete many of the tasks outlined in this paper.

Related references

Before you begin to integrate Select Access with Tomcat, you may want to begin by familiarizing yourself with the contents of the following documents:

- *HP OpenView Select Access 6.0 Installation Guide*, © Copyright 2000-2004 Hewlett-Packard Development Company, L.P. (`installation_guide.pdf`)
- *HP OpenView Select Access 6.0 Network Integration Guide*, © Copyright 2000-2004 Hewlett-Packard Development Company, L.P. (`network_integration_guide.pdf`)
- *HP OpenView Select Access 6.0 Policy Builder Guide*, © Copyright 2000-2004 Hewlett-Packard Development Company, L.P. (`policy_builder_guide.pdf`)
- *HP OpenView Select Access 6.0 Developer's Tutorial Guide*, © Copyright 2000-2004 Hewlett-Packard Development Company, L.P. `(dev_tut_guide.pdf)`
- *HP OpenView Select Access 6.0 Developer's Reference Guide*, © Copyright 2000-2004 Hewlett-Packard Development Company, L.P. (`dev_ref_guide.pdf`)
- Hewlett-Packard, Application/portal servers *Integration Papers*, © Copyright 2000-2004 Hewlett-Packard Development Company, L.P.

Chapter 2
Technologies overview

This chapter introduces you to Select Access and Tomcat. It gives you an overview of the products: what they do, what components are installed with these products, and what Tomcat integration issues exist.

What is Select Access?

Select Access is a centralized access management system that provides you with a unified approach to defining authorization policies and securely managing role-based access to on-line resources. It uses a collection of components that integrate with your network, to give you and your partners the ability to capitalize on the potential of extranets, intranets and portals. These components, along with the access policies you set, offer your Web and wireless users a seamless user experience by connecting them to dispersed resources and applications.

What does Select Access do?

Several features of Select Access extend its functionality beyond that of a simple authorization administration tool. It is a complete access management system, offering you a set of features to support your online relationships with your users and your content partners:

- *Supports single sign-on*
- *Enables user profiling*
- *Provides user password and profile management*
- *Delegates administration*
- *Provides an end-to-end auditing system*
- *Automates the discovery and maintenance of corporate resources*

Together, this extended functionality provides a simplified experience for both the end user and those responsible for managing what the user sees and interacts with.

Supports single sign-on

To improve user satisfaction, Select Access incorporates a Web Single Sign-On (SSO) capability. This means users can sign on once to access

all permitted resources and have their information stored for future access. Select Access supports transparent navigation between:

- Multiple proprietary domains: For organizations with ownership of multiple Web sites.
- Multiple partnering domains: For on-line business partners, so they can securely share authentication and authorization information across corporate boundaries that have separate:
  - user databases
  - authorization policies
  - access management products

Using SSO means that users do not have to remember multiple passwords or PINs, thereby reducing the amount of help desk support.

Enables user profiling

A user is represented as a user entry that is stored in a directory server. When you create a user entry, you can also define a set of attributes that describe that user, which become part of the user's profile. The values contained in the attribute can be used in two ways:

- *To determine level-of-access with roles*: Role-based access allows you to configure and apply policies automatically, according to the attribute values stored in the user's profile.
- *To determine delivery-of-content*: Select Access exports user attributes and their values as environment variables, so that applications can use the profile information to personalize Web pages and to conduct transactions.

A user's profile dynamically changes as a user conducts transactions with your organization. As attributes in the profile change, so too can the role the user belongs to. For example, a customer's profile may contain his current bank balance, date of last transaction, and current credit limit—any of which can change from moment to moment.

This capability of Select Access makes development of Web applications much easier, because programmers do not have to develop (or maintain) complex directory or database access codes to extract entitlement information about each user.

Provides user password and profile management

Select Access's password and profile management feature makes it easy for users to conduct business and minimize the demand on technical resources that can best be employed elsewhere. This feature includes the following principles:

- *Password administration*: Allows you to set the policies and expiration times for user passwords. Select Access automates reminders and messages. Other administration features include:

  - Profile lockout and re-activation
  - Password history lists
- *Self-servicing*: Allows users to initiate:
  - The definition of new or existing passwords, which are controlled by the password policy you create.
  - The modification of profile data, which is predefined by the attributes you select. Typically, these attributes are the same attributes the user provides when they register with your organization. If the user is already known to you (like an employee or a supplier), you can pre-populate the values for them.

By allowing users to self-manage passwords and profile data, you reduce the amount of help desk support.

Delegates administration

Delegated Administration allows for delegation of both user and policy management, providing more control for decentralized administrators. Select Access's delegation is highly efficient: it supports sub-delegation to multiple tiers of administrators, which mimics real-world organization charts. This decentralized approach to administration:

- Reduces administrative bottlenecks and costs.
- Puts the power to manage users in the hands of those who best understand those users.

Provides an end-to-end auditing system

Select Access can record all access and authorization actions, as well as all policy administrative changes to any number of outputs, such as:

- The HP Secure Audit server
- JDBC-compliant databases
- Local files
- Platform-specific log files
- Email

Of all output choices, the Secure Audit server is the most useful: not only does it collect messages from different components on a distributed network, but it also allows you to digitally-sign all audit entries and ultimately create a report from the outputs collected.

Automates the discovery and maintenance of corporate resources

In order to define and enforce authorization, Select Access must be aware of all the resources on your network, as well as the users who want to access them. Select Access uses the directory server as the central repository for policy data, which includes the resource listing. You can deploy special HTTP/HTTPS-specific plugins to automatically scan any given network, thereby enumerating available services and resources. As services and resources are enumerated by the plugin, it adds them hierarchically in the Policy Builder's Policy

Matrix. Unlike other products that require manual data input (where a simple typing error can put the security of resources at risk) Select Access saves administrators' time and improves accuracy.

How does Select Access work?

Select Access delivers the core of its authorization and authentication functionality with the following technical components:

- *Policy Builder*: Allows full or delegated administrators to define the authentication methods and authorization policies with an easy-to-use administration grid.
- *Policy Validator*: Serves the access decision to the Enforcer plugin after it accepts and evaluates the user's access request with the policy information retrieved from the directory server that holds your Policy Store.
- *Enforcer plugin*: Acts as the agent for Select Access on the Web/application server. The Enforcer plugin enforces the outcome of the access request that has been evaluated by the Policy Validator.
- *SAML server*: Handles the logistics of transferring users between your web sites and those of your partners.

These core components form a sophisticated and consistent architecture that easily adapts to any existing network infrastructure. Primarily XML and Java-based, you can readily extend Select Access to meet the needs of future security requirements.

The authentication process

Select Access's authentication and authorization of Web-based or wireless users takes place within a small number of basic steps. Select Access components communicate via XML documents known as queries and responses. XML offers Select Access complete flexibility for data transmission and integration into existing and future applications, whether Web or non-Web based. Select Access's authentication and authorization process follows these steps:

1. A user makes a request to access a resource.
2. The Enforcer plugin passes details of the request to the Policy Validator, including any authentication information provided.
3. The Policy Validator collects user and policy data from the directory and then caches it for future retrieval.
4. Based on this combination of information, the Policy Validator returns a policy decision to the Enforcer plugin, including relevant information to dynamically personalize the user experience.

Other Select Access components

Other Select Access components provide the support system for Select Access's core components:

- *Administration server & Setup Tool*: As a mini Web server, the Administration server is responsible for the configuration of core components and deployment of the Policy Builder applet in a user's browser. The Setup Tool is a client of the Administration server: it is the interface that allows you to quickly set up and deploy Select Access.
- *Secure Audit server*: Collects and manages incoming log messages from Select Access components on a network.

Third-party components Select Access integrates with

Other third-party components that are integral to an effective Select Access solution:

- *Directory server – LDAP v3.0 compliant*: is the foundation of a Select Access-protected system. It acts as the repository of information. Depending on how you have set up your directory system, Select Access can use one or more directory servers to store:
  - A single policy data location
  - One or more user data locations
- *Web/Application/Portal/Provisioning servers*: are third-party technologies that use Select Access as their authorization and access management system. Depending on your server technology, you can use Select Access's native SSO and/or personalization solution rather than use the server's built-in alternative for a more robust solution.

Figure 1 illustrates how Select Access and third-party components interact with each other.

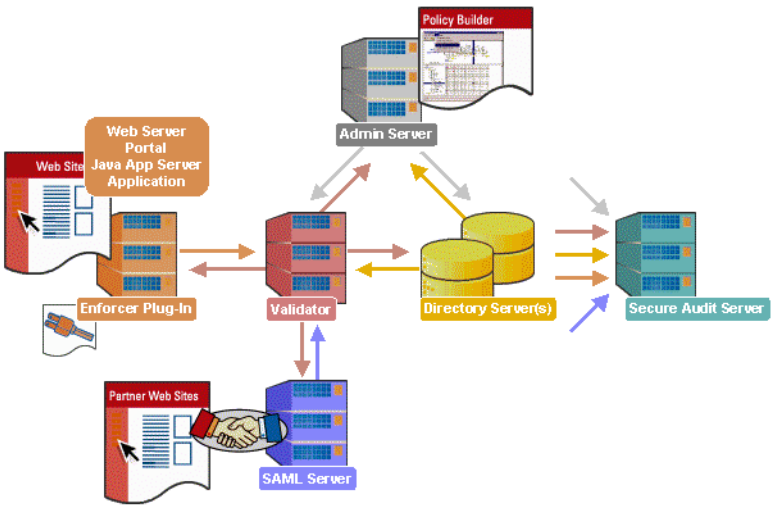

Figure 1: Select Access system architecture

Custom plugins you can customize functionality with

To more efficiently capture your organization's business logic, you can use Select Access's APIs to build custom plugins. Plugins that you can customize functionality with include:

- *Authentication plugins*: A custom Policy Builder authentication plugin allows you to tailor which kinds of authentication methods are available to better meet the needs of your organization. A Policy Builder authentication method plugin allows administrators to use and configure the authentication server for this method via a dialog box. As with the decision point plugin, this dialog box is a property editor that allows security administrators to configure the authentication server.
- *Decision point plugins*: A custom Rule Builder decision point plugin allows you to tailor how rules are built to better meet the needs of your organization. A Rule Builder decision point plugin allows administrators to use and configure the criteria for the decision point via:
  - The icons that represent that decision point on both the toolbar and the rule tree.
  - The dialog box, known as a property editor, that allows security administrators to configure it.
- *Policy Validator decider plugins*: The Validator-specific counterpart of a decision point plugin, the decider plugin allows you to capture the evaluation logic for your custom decision point (described above), so that the Policy Validator can evaluate users based on the information it collects.
- *Resource discovery plugins*: These plugins allow you to customize how resources are scanned on your network.
- *Enforcer plugins*: A new Enforcer plugin allows you to customize the backend application logic by enforcing the decision that the Policy Validator returns to the Enforcer plugin's query.
- *Additional Web/Application/Portal/Provisioning server specific plugins*: These plugins can be included to handle specific integration details between the third-party technology and Select Access. For example, the Domino server requires a `site_data` plugin if you need to transfer site data between Select Access and Domino.

What is Tomcat?

Tomcat is the implementation of Java Server Pages (JSP) and Servlets, which is developed and maintained by the Jakarta Project (http://jakarta.apache.org/). Tomcat is used to build dynamic Web content when:

- User attributes drive personalization of the content served.

- Frequent content changes drive the need for easily renewable content.
- Content data sources such as databases are used as the source for pages that are rendered dynamically.

Because the servlet creates server-side pages and/or applications, your Web site becomes more efficient (because it is persistent), is more convenient to the user (pages are built on demand), and is more powerful (does things that many other applications cannot do).

In most deployments, Tomcat is integrated as a plugin for the regular Apache Web server. This document only addresses this typical deployment scenario. It does not address Tomcat on commercial Web servers.

How the Select Access and Tomcat solution works

With a collection of Java technologies (JSP, Java API, and so on), Select Access with Tomcat v4.0.3 over Apache v1.3.22 allows you to build a three-tier system, as shown in Figure 2. This system:

- Makes the deployment of dynamic JSP pages and servlets easier.
- Allows for enhanced transaction management.

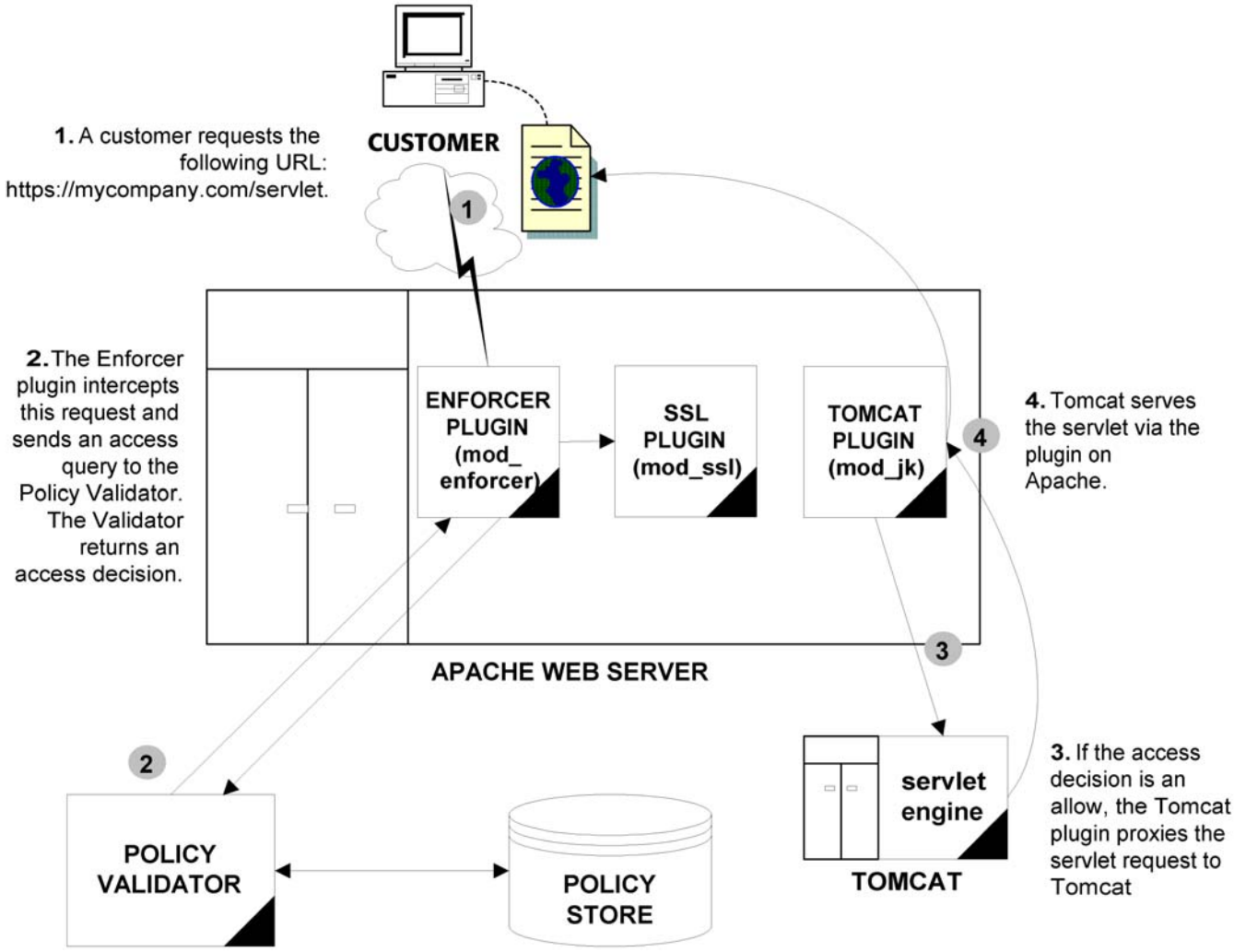

Figure 2: Select Access and Tomcat with Apache

Issues that affect Tomcat

There are two main issues that affect Tomcat with Apache:

- To enforce security across this dispersed architecture, you need to configure Tomcat so it always forces resource requests through the Apache Web server (using a Tomcat plugin to proxy servlet requests), rather than accept resource requests indirectly through its built-in Web server. This configuration ensures that requests are always handled by Apache Enforcer plugin. Consequently, you must disable all services and connectors, keeping only ajp13 active.
- To avoid receiving harmless warning messages, HP recommends that you rebuild your Apache Enforcer plugin—especially if you enable SSL on a Solaris host.

The benefits of Select Access's solution

Integrating Select Access with Tomcat offers the following main benefits:

- *Secures Tomcat from direct access*—With services disabled and with the Apache Enforcer plugin intercepting all requests, direct resource requests are prevented which minimizes risk to your Tomcat servlet engine.
- *Multiple levels of access*—Using a combination of allow, deny and conditional policies ensures user access is restricted to meet your business requirements.
- *Multiple authentication methods*—Select Access's multiple authentication methods (digital certificates, RADIUS, SecurID, registration, and password authentication) extend Tomcat's security architecture. This allows administrators to do the following:
  - Tier security based on the sensitivity of the resource.
  - Use more than one authentication method to create a more secure multi-authentication mechanism.

Chapter 3

Integrating Select Access with Tomcat over Apache

Integrating these two technologies requires that you configure Apache, Tomcat, and Select Access with specific properties and parameters to ensure they function properly as a unit. For details, see the corresponding section below:

Tomcat v4.0.3 requires that you install the JDK. When you install Select Access, JRE v1.3 is installed with the product. However, HP recommends that you download and install JDK v1.4 to facilitate the integration process. Ensure that you set the `PATH` so that both `java -version` and `javac -help` return the correct values.

The subsequent sections assume that you have downloaded the binaries for Tomcat v4.0.3 and the source for Apache v1.3.22—although you can use more recent versions. If you do use a more recent version, you simply need to rebuild the Apache Enforcer plugin.

Integrating the Apache Web server with Tomcat

When you integrate Tomcat on an Apache Web server, you are building it to run with new plugins as well as to enable it to run over SSL. Table 1 outlines the high-level setup tasks and their

corresponding implementational steps that you must consider when setting up Apache to run with the Tomcat servlet engine.

Table 1: Integrating Apache with Tomcat—procedure summary

This setup task...	Steps to accomplish it...
Step 1: Build Apache with packages that enable it to run over SSL. **Note**: This step assumes that Perl is already installed on Apache's computer. If Perl is not installed, please do so before attempting to build packages.	1. Download and extract the following packages: — `apache_1.3.22.tar.gz from` http://www.apache.org/dist/httpd/old/ — `mod_ssl-2.8.5-1.3.22.tar.gz` from http://modssl.org/source/OBSOLETE/ — `openssl-0.96d.tar.gz` from http://openssl.org/source/ — `mm-1.1.3.tar.gz` from http://engelschall.com/sw/mm/ — `perl-5.0.6.tar.gz` from http://cpan.org/src/5.0/ 2. Build the following packages in their corresponding order, as described in *Building required packages* on page 15. — `openssl` — `mm` — `mod_ssl`
Step 2: Integrate the Tomcat plugin (`mod_jk`) with the Apache Web server. `mod_jk` works as a compatibility layer—sending servlet requests to Apache.	1. Download and extract the following: — Tomcat binaries: `jakarta-tomcat-4.0.3.zip` from http://jakarta.apache.org/builds/jakarta-tomcat-4.0/release/v4.0.3/bin — The Tomcat plugin source package: `jakarta-tomcat-connectors-4.0.2-01-src.tar.gz` from http://jakarta.apache.org/builds/jakarta-tomcat-4.0/release/v4.0.3/src/ **Tip:** The version of `mod_jk` used on Apache has a direct correlation to the version of Tomcat you intend to use. **Note:** To avoid receiving harmless warning messages, HP recommends that you rebuild your Apache Enforcer plugin—especially if you enable SSL on a Solaris host. 2. Change to the Tomcat plugin directory, and build `mod_jk`. The procedure for this varies depending on your OS. For details, see *Building the mod_jk plugin* on page 17 3. Modify your Web server so it knows to load this plugin. Do this by editing the `LoadModule` section of your `httpd.conf` file so that the `mod_jk` is loaded after the `mod_ssl` plugin. Code example 1 shows typical lines that you might add at the end of the `LoadModule` section.

Code example 1: Adding `mod_jk` to the end of the `LoadModule` section

```
# load the jk_module
LoadModule    jk_module  libexec/mod_jk.so
AddModule     mod_jk.c

# jk_module configuration properties
JkWorkersFile /usr/local/jakarta-tomcat-4.0.3/conf/workers.properties
JkLogFile     /usr/local/apache/logs/mod_jk.log
JkLogLevel    info
JkLogStampFormat "[%a %b %d %H:%M:%S %Y] "

# directories you want to redirect to Tomcat
JkMount /*.jsp ajp13
JkMount /examples/servlet/* ajp13
```

Building required packages

The packages you downloaded and expanded need to be built into your Apache Web server before SSL is enabled and plugins/modules interact more efficiently.

- OpenSSL is an open source SSL library that allows Apache to communicate over SSL. For details, see *To build OpenSSL* on page 15.
- `mm` is an abstraction library that simplifies the usage of shared memory between forked processes. For this reason, it is needed on the Apache server as base library for providing shared memory pools to Apache modules. For details, see *To build mm* on page 16.
- `mod_ssl` is the interface counterpart to the OpenSSL libraries for Apache. This module provides strong cryptography via SSL/TSL protocols. For details, see *To build mod_ssl* on page 16.

To build OpenSSL

1. Change to the directory in which you expanded the OpenSSL libraries. For example, using the default directory, the command for this is:

   ```
   cd openssl-0.9.6d
   ```

2. To parse the configuration file so that it creates general parameters from it, run the command shown below from this directory.

   ```
   sh config no-threads -fPIC
   ```

3. To compile the library, run the `make` command.
4. To test the results of the compiled library, run the `make test` command.

5. If the results are positive, change out of the OpenSSL directory with the `cd..` command.

To build mm

1. Change to the directory in which you expanded the `mm` abstraction libraries. For example, using the default directory, the command for this is:

   ```
   cd mm-1.1.3
   ```

2. Run the `configure` script with the following flag:

   ```
   ./configure --disable-shared
   ```

3. To compile the library, run the `make` command.
4. Change out of the `mm` directory with the `cd..` command.

To build mod_ssl

1. Change to the directory in which you expanded the `mod_ssl` source code. For example, using the default directory, the command for this is:

   ```
   cd mod_ssl-2.8.5-1.3.22
   ```

2. Run the `configure` script with the following parameters so that the module is configured to run with:
   - The Apache Web server.
   - The libraries you have just built.

   ```
   ./configure --with-apache=..<apache_dir>
   --with-ssl=..<openssl_dir> --with-mm=..<mm_dir>
   --enable-module=so --enable-shared=ssl
   ```

3. Change to the Apache directory you included as a parameter in step 2.
4. To compile Apache with this module, run the `make` command.
5. To build an SSL test certificate that tests Apache with the `mod_ssl` plugin, run the following `make certificate` command.

You can skip this step if you would like to use Apache's `mod_ssl` plugin with a real certificate. However, you need to install it manually.

6. To install the plugin files to their required locations, run the following `make` command.

   ```
   make install
   ```

7. Open the server's `httpd.conf` file. Do this with the following command:

   ```
   vi <apache_install_path>/conf/httpd.conf
   ```

8. Ensure that you configure the correct values for Apache's HTTP ports (both SSL and non-SSL).

Building the mod_jk plugin

Building this plugin is an important integration step to get the Tomcat servlet engine running with Apache. Depending on which operating system your Web server is running on, the procedure varies. For specific details for Solaris or Linux, see the corresponding section that follows.

To build mod_jk on Solaris

1. Change to the Tomcat plugin directory. Using Tomcat's defaults, this directory is:

   ```
   cd jakarta-tomcat-connectors-4.0.2-014-
   src/jk/native/apache-1.3
   ```

2. Open the `build-solaris.sh` script.
3. Set the `APACHE_HOME` and `JAVA_HOME` variables to reflect your specific installation folder for Apache and your JDK.
4. Go to line 39 of the `build-solaris.sh` script and edit the line so it appears like the one shown below (changes to the line appear in bold):

   ```
   $APXS -DEAPI -S CFLAGS="-DSOLARIS -DUSE_EXPAT -
   I../lib/expat-lite" -o mod_jk.so $INCLUDE -lposix4 -c
   $SRC
   ```

   This flag tells the script that Apache was built with EAPI.
5. Execute the `build-solaris.sh` script with the following command:

   ```
   sh ./build-solaris.sh
   ```

To build mod_jk on Linux

1. Change to the Tomcat plugin directory. Using Tomcat's defaults, this directory is:

   ```
   cd jakarta-tomcat-connectors-4.0.2-014-
   src/jk/native/apache-1.3
   ```

2. Export the `APACHE_HOME` and `JAVA_HOME` values as environment variables to reflect your specific installation folder for Apache and the JDK. For example, using default installation directories, you would set values with the following command:

   ```
   export APACHE_HOME =usr/local/apache
   export JAVA_HOME=usr/local/java
   ```

3. Execute the `build-unix.sh` script with the following command:

   ```
   sh ./build-unix.sh
   ```

Configuring Tomcat to minimize the number of connections

Tomcat uses an entity called "workers" to run servlets on the Apache Web server's behalf. Worker definitions are added to a `workers.properties` file. This repository of worker definitions is used by Tomcat so it knows where different workers are and which

workers it needs to forward requests to. Table 2 outlines the high-level setup tasks that you must consider when configuring the number of connections the Tomcat servlet engine keeps active.

Table 2: Configuring Tomcat—procedure summary

This setup task...	For details, see...
1. In your `tomcat/conf` directory, create a `workers.properties` file.	*Creating a workers.properties file* on page 18
2. Configure your Tomcat server so you disable its built-in Web server and only keep the ajp13 connector active.	*Configuring Tomcat* on page 19

Creating a workers.properties file

A worker is an instance of a Tomcat process. A worker can use a number of protocols so that it can process servlets that the Apache Web server requests. There are many different protocols you can use with Tomcat. However, when integrating Tomcat over Apache with Select Access, there is only one you need to concern yourself with: the ajp13 worker. This worker is the one that forwards requests to out-of-process Tomcat workers using the ajp13 protocol.

To create a ajp13 worker definition

1. Define the Tomcat install directory. The syntax for this is:

   `workers.tomcat_home=<tomcat_install_path>`

2. Define the Java install directory. This allows Tomcat workers to find the java files they need to serve Java servlets. The syntax for this definition is:

   `workers.java_home=<jdk_install_path>`

3. Create the worker list. Your worker list only includes the ajp13 worker. For example:

   `worker.list=ajp13`

4. Define the properties for the ajp13 worker. These properties define the connection information for it. The syntax for a property definition is:

   `worker.<worker_name>.<property_name>=<value>`

   For example, typical properties you would want to define include:

   ```
   worker.ajp13.port=8009
   worker.ajp13.host=localhost
   worker.ajp13.type=ajp13
   ```

   For more details on other properties available for the ajp13 worker, see Tomcat's *Worker Definition* guide.

Configuring Tomcat

Because the Apache Enforcer plugin only intercepts resource requests for the Apache Web server, you need to disable Tomcat's built-in Web server. This action prevents unwanted direct connections to the Tomcat servlet engine. Only the connector required by the `mod_jk` plugin must remain active so it can proxy servlet requests to Tomcat. However, keeping even one connector open still exposes Tomcat to direct servlet requests. Therefore, to fully secure Tomcat requires that you:

- Deactivate all connectors, with the exception of the ajp13. For details, see *To deactivate connectors that disable Tomcat's built-in Web server* below.
- Heavily firewall Tomcat's host machine to limit direct ajp13 requests to those originating from Apache's host computer.

To deactivate connectors that disable Tomcat's built-in Web server

1. Open the `server.xml` configuration file. By default, you can locate this file in the following directory:

   `<tomcat_install path>/conf/`
2. Disable all HTTP listeners and other unwanted connectors so that Tomcat cannot use its built-in Web server. Do this by commenting out all corresponding lines in this file so only the ajp13 connector remains active. XML requires that you use the tags to wrap the lines as shown below:

   `<!-- Disable this line -->`

   Code example 2 illustrates a non-SSL section that has been disabled with this syntax.
3. Save your changes.

Code example 2: HTTP listeners section

```
<!-- Define a non-SSL HTTP/1.1 Connector on port 8080 -->
  <!--
  <Connector className="org.apache.catalina.connector.http.HttpConnector"
             port="8080" minProcessors="5" maxProcessors="75"
             enableLookups="true" redirectPort="4443"
             acceptCount="10" debug="0" connectionTimeout="60000"/>
  -->
```

Configuring Select Access to proxy requests with the Enforcer plugin

Table 3 outlines the high-level setup tasks and their corresponding implementational steps that you must consider when setting up Select Access to run with the Tomcat servlet engine.

Table 3: Setting up Select Access

This step...	Details on how to do it...
Step 1: Install the Apache Enforcer plugin and configure it to minimize the number of extra forked Apache processes that are created when the server receives multiple requests in parallel. This is because production Apache Enforcer plugins need to be configured to be more forgiving on slower networks, than on a test deployment. For details on the setup wizard, see Chapter 8, *Configuring the Enforcer plugins* in the *HP OpenView Select Access 6.0 Installation Guide*.	1. After you have installed the plugin, run the Setup Tool. Click **Next** until you reach the setup wizard for the Apache Enforcer plugin. 2. On the **General** setup screen, choose the **Custom** configuration option. 3. Configure the Apache Enforcer plugin with the following **Tuning Parameter** values—especially if your Apache Web server is using International Components for Unicode (ICU) libraries. For reference purposes, XML parameter name equivalents are included in the brackets. — **Stop Validator connection attempt after** (`ctimeout`) = 60 — **Retry unavailable Validators every** (`rtimeout`) = 30 — **Wait for Validator reply** (`KeepAliveTimeout`) = 3-5 4. If you are automatically allowing the Setup Tool to modify Apache's `httpd.conf` file, ensure that the plugin is the last module listed. 5. Using an LDAP browser, modify the Apache Enforcer plugin properties in the directory server that holds your Policy Store information. You must make these changes manually because these parameters are not configurable with the Setup Tool. — `MaxClients` = 60 — `MaxRequestsPerChild` = 1000 **Note:** This step assumes that a Select Access Administration server and Policy Validator are already running on your network. 6. Open the server's `httpd.conf` file and ensure that the Apache Enforcer plugin is the last module to be loaded. Do this with the following command: `vi <apache_install_path>/conf/httpd.conf`
Step 2: If you require SSL EAPI support, rebuild your Apache Enforcer plugin.	1. Change to Select Access's installation directory. Using Select Access's default location, this command is: `cd /opt/OV/SelectAccess` 2. Run the `make` command to rebuild the Apache Enforcer plugin.

Table 3: Setting up Select Access

This step...	Details on how to do it...
Step 3: Add the Apache Web server and the Tomcat servlet engine to the Resources Tree as service branches.	1. Right-click a folder or the root of the Resources Tree. 2. Click **Run Discovery>Services.** The **Discover Networks Services** dialog box appears. 3. Provide the required information on the **Networks** and **Protocols** tabs and then click **OK**. For details, see Chapter 4, *Building your Users and Resources Trees* in the *HP OpenView Select Access 6.0 Policy Builder Guide*. **Note:** Ensure you configure HTTPS if you have configured Apache/Tomcat to run over SSL.
Step 4: Add the resources you want to protect under the services you just created.	1. Make sure you have configured the network resource plugin. For details, see Chapter 4, *Building your Users and Resources Trees*. 2. On the Resources Tree, right-click the service you want to scan for available resources. 3. Click **Run Discovery>Resources**. The **Discover Network Resources** dialog box appears. — Information about the service's representative server is entered automatically in the **Protocol**, **Hostname**, and **Port Number** fields. (This information is taken from the service's properties.) — If you have configured a plugin for the protocol used by the service, the plugin's configuration details are entered automatically in the **Plugin Settings** field. 4. Select **Run Resource Discovery Plugin** and fill in any empty fields. 5. Select the location on the Resources Tree to add the resources. Do the following: a. Click the **Browse** button beside the **Network Resources Tree Destination** field. The **Select Resource Destination** dialog box appears. b. Select a service, then click **OK**. 6. Click **OK.** For details, see *Automatically generating a list with a discovery plugin* on page 44 in the *HP OpenView Select Access 6.0 Policy Builder Guide*.

Testing your deployment

Testing your integrated deployment is important when integrating any combination of technologies—but especially when it is a three-way deployment. HP recommends that you check your Apache/Tomcat server combination to ensure that you can:

- Start your Tomcat servlet engine. Do this by running the following script: `<Tomcat_install_path>/bin/startup.sh`.

Ensure you set all scripts in the `<Tomcat_install_path>/bin/` directory to executable, by using the following command: chmod + x *.sh

- Start Apache and ensure that all recently-installed plugins are behaving as expected.
- Access any HTML and/or JSP pages you require. Testing the latter shows that the Java compiler and the Java Virtual Machine are installed and running correctly.

Also check that your level of access matches the access policies you have configured in the Policy Builder.

- Test a wide range of servlets and ensure that they function correctly:
  - a servlet without packages
  - a servlet with packages
  - a servlet with packages and utility classes