JASPERSERVER

EXTERNAL AUTHENTICATION

COOKBOOK

RELEASE 3.5

http://www.jaspersoft.com

This is version 0409-JSP35-1 of the *JasperServer External Authentication Cookbook*.

TABLE OF CONTENTS

1 INTRODUCTION

JasperServer builds on JasperReports as a comprehensive family of Business Intelligence (BI) products, providing robust static and interactive reporting, report server, and data analysis capabilities. These capabilities are available as either stand-alone products, or as part of an integrated end-to-end BI suite utilizing common metadata and providing shared services, such as security, a repository, and scheduling. JasperServer exposes comprehensive public interfaces enabling seamless integration with other applications and the capability to easily add custom functionality.

This cookbook describes the customization of JasperServer to use an external authentication mechanism in place of the built-in authentication of users. This document covers the following authentication mechanisms:

- LDAP - Lightweight Directory Access Protocol
- CAS - Central Authentication Service

The benefits of external authentication include:

- Users accounts shared by all applications within your enterprise.
- Depending on the authentication mechanism, single sign-on capabilities with other applications within your enterprise.

For deployments that also include JasperAnalysis running within JasperServer, external authentication applies transparently to JasperAnalysis users.

The procedures in this document assume you are familiar with JasperServer installation, deployment, and administration. You must have system administrator privileges within JasperServer and unlimited access to the host machine.

This chapter contains the following sections:

1.1 Overview and Terminology

Authentication is the verification of a user's identity to allow access to JasperServer. By default, anonymous authentication is disabled, and all users must present valid credentials to login: a username, a password, and in certain cases an organization ID. Authentication is more than gathering the user credentials, it is a process that ensures that every page is secure, either viewed by a recognized user or denied access until valid credentials are provided.

JasperServer users are either people who accesses the web-based interface or applications that accesses the same content through web services. Authentication for the two types of access is slightly different but operates on the same principles. For simplicity, this document refers to users as people accessing the web-based interface.

User accounts are created by administrators and stored in an internal database known as the JasperServer database. For example, when you install JasperServer with the default settings, it requires a MySQL database server where it stores the JasperServer database tables containing user information. The JasperServer database is independent of database that stores the data that JasperServer uses to fill your reports, although they may be on the same database server.

Authentication happens only once at the beginning of the user's session. After a user is authenticated, the user's session is represented by an in-memory instance referred to as the principal object in this document. The existence of the principal object determines that the user is logged on, and the user can access multiple pages throughout the application. The principal object also stores the roles and organization ID of the user, which are required for authorization within JasperServer.

External authentication is the process of gathering and verifying user credentials through a third-party application; for example a corporate LDAP directory. The external application is called an authority because it is trusted to contain valid user information. The process of external authentication depends on the nature of the external authority, and the various configurations create different usage scenarios. For example, with simple external authentication, users log into the JasperServer page, but their credentials are verified externally; in a single sign-on configuration, the user may log into another application, then navigate to JasperServer, without seeing the JasperServer login page.

Authorization is the verification of a user's roles and organization ID to access JasperServer features, repository objects, and, in some cases, data. For every page that the user requests, JasperServer determines which menu items, repository contents, and report contents the user may access, based on the principal object. Authorization happens every time a user accesses a resource.

In the JasperServer architecture, which is based on the Spring Framework and Acegi Security, authentication may be configured through an external authority, but authorization is always performed by internal mechanisms. Part of configuring external authentication is the mapping of external roles and organization IDs into the principal object so that authorization can proceed internally. The information for authorization is determined when the session of an external user is established and is also stored in an account representing the external user in the JasperServer database. The information is synchronized every time the external user begins a new session with JasperServer, so that the JasperServer database contains the user's latest information.

1.2 Multiple Organizations in JasperServer

JasperServer Professional 3.5 introduced a new architecture to manage several distinct organizations within the same server. Each organization has its own users and roles, and possibly a hierarchy of sub-organizations, all of which are invisible to other organizations. For more information about deploying multiple organizations, see the latest *JasperServer Professional Administration Guide*.

Configuring external authentication for servers hosting multiple organizations is similar to servers without the new architecture, but requires extra steps. The procedures in this book include the basic configuration for all 3.x servers, and when feasible, the additional steps to support multiple organizations.

The default installation of JasperServer 3.5 Professional includes a single organization that mimics the behavior of previous releases without organizations. However, even when using a single organization, the architecture still requires the additional configuration steps to perform external authentication.

1.3 Default Internal Authentication

Understanding how JasperServer performs internal authentication will help you implement external authentication. The following diagram shows the general steps involved in the default internal authentication:

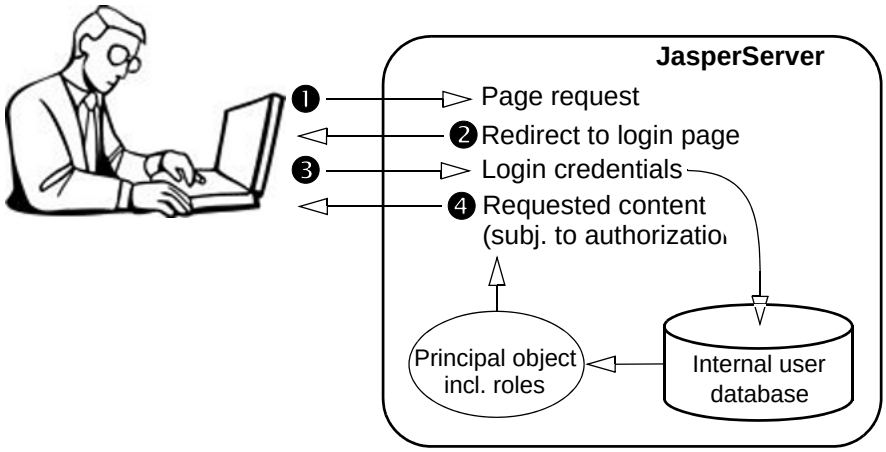

Figure 1-1 General Steps of Internal Authorization

The following steps explain the interaction between the user's browser and JasperServer:

1. An unauthenticated user requests any page in JasperServer.

   Often, users bookmark the login page and begin directly at step 3, but this step covers the general case and secures every possible access to JasperServer. For example, this step applies when a user clicks the page of an expired session or if a user is given the direct URL to a report within JasperServer.

2. JasperServer detects that the user is not logged in and replies with a redirect to the login page.

   For convenience, JasperServer includes the original request in the login screen URL so that the user goes directly to the requested page after logging in.

3. The user enters a username, password, and possibly an organization ID.

   JasperServer compares these credentials with the existing user accounts in the internal user database, and if they are valid, creates a principal object. The user is now authenticated, and the principal object represents the user session, including any roles found in the user database.

4. JasperServer sends the requested content to the user, or if none was specified, the home page appropriate for the user.

   Content that is sent to the user is subject to authorization. For example the home page has different options for administrators than for end-users, as determined by the roles of the user in the principal object. If the user is viewing the repository, the folders and objects that are returned are based on the organization ID and roles in the principal object.

1.4 Acegi Security Framework

JasperServer relies on Acegi Security 1.0.1, an open source security solution that provides the mechanisms to authenticate and authorize users. Acegi Security is integrated with the Spring Framework, the open source Java application platform that can be customized through Java beans defined in configuration files.

JasperServer's default configuration represents the most common use cases, but Acegi's flexibility can provide very fine-grained control of the security system. However, since Acegi can be configured in so many ways, you may find its configuration somewhat daunting. This cookbook presents the most common scenarios for implementing external authentication.

For more information about Acegi beyond the scope of this cookbook, refer to the documentation for the Acegi Security 1.0.x, available at:

- http://www.acegisecurity.org/guide/springsecurity.html
- http://jasperforge.org/espdocs/docsbrowse.php?group_id=20&fid=24 (PDF)

If you plan to make extensive customizations, JasperSoft recommends that you delve more deeply into Acegi by visiting its SourceForge project and participating in its community.

The following sections explain the Acegi configuration and flow for each of the four steps shown in **Figure 1-1**.

1.4.1 Acegi Filter Chain

The main entry point for both authentication and authorization is the `securityFilter`, which is configured in WEB-INF/web.xml. As with all J2EE servlet filters, the `securityFilter` is invoked for every HTTP request for paths in this web application, whether the request comes from users or automated agents. The `securityFilter` is essentially a proxy for Acegi's `FilterChainProxy` class, which handles the actual logic that checks and grants authorization. In JasperServer, this class is instantiated as a Spring bean configured in the WEB-INF/applicationContext-security.xml configuration file:

```
<bean id="filterChainProxy" class="org.acegisecurity.util.FilterChainProxy">
  <property name="filterInvocationDefinitionSource">
    <value>
      CONVERT_URL_TO_LOWERCASE_BEFORE_COMPARISON
      PATTERN_TYPE_APACHE_ANT

      /xmla=httpSessionContextIntegrationFilter,
      basicProcessingFilter,JIAuthenticationSynchronizer,anonymousProcessingFilter,
      basicAuthExceptionTranslationFilter,filterInvocationInterceptor

      /services/**=httpSessionContextIntegrationFilter,
      basicProcessingFilter,JIAuthenticationSynchronizer,anonymousProcessingFilter,
      basicAuthExceptionTranslationFilter,filterInvocationInterceptor

      /**=httpSessionContextIntegrationFilter,authenticationProcessingFilter,
      basicProcessingFilter,JIAuthenticationSynchronizer,anonymousProcessingFilter,
      exceptionTranslationFilter,filterInvocationInterceptor
    </value>
  </property>
</bean>
```

The configuration of the `filterInvocationDefinitionSource` property of the `filterChainProxy` gives a list of URL request patterns, and for each pattern a sequence or chain of filters. Each filter in the list is a Java servlet bean, meaning it is given the request and response context and has a chance to take action before control is passed to the next bean in the chain.

Before the patterns, there are two directives that determine how URLs are matched. The first patterns take precedence, so the most specific patterns are listed before more general ones. The pattern associated with the last expression is usually `/**` with the Apache Ant patterns type, because it matches anything that wasn't caught by a different pattern.

By changing the chain of filters, you can change the behavior for the URLs that match the pattern. You can even define and add your own filter if you need to implement a custom authentication schema, but that is beyond the scope of this document. Acegi Security includes support for external authentication by reconfiguring the built-in filters, which is sufficient in almost every case. The filters themselves are configured through other beans, as described in the following sections.

1.4.2 Step 1: Receiving a Page Request

When JasperServer receives a page request, it is processed by the Acegi filter chain. The URL of the request is matched to one of the patterns and then the request is processed by the corresponding filter chain.

The `/xmla` and `/services/**` patterns represent the JasperServer XML for Analysis (XML/A) servlet and the web services, respectively. They are configured with one set of filters designed to receive SOAP requests from other servers. All other URLs

matched by the final pattern are designed for people using web browsers and have a different set of filters; for example, only people are asked to proceed to the login page.

1.4.3 Step 2: Redirecting to the Login Page

As the request is being processed, Acegi security within JasperServer calls each of the filters in the chain. The standard filter beans have the following functions:

- `httpSessionContextIntegrationFilter` - Stores the `SecurityContext` in the `HttpSession`.
- `authenticationProcessingFilter` - Redirects the user to the login page if not yet authenticated, then processes the organization ID, username, and password parameters returned from the login page.
- `basicProcessingFilter` - Handles HTTP BASIC style authentication, where the username and password are passed as an HTTP header.
- `JIAuthenticationSynchronizer` - A custom filter that notifies JasperServer which user has logged in. This filter ensures that the principal object in memory is synchronized with the information in the user database.
- `anonymousProcessingFilter` - Marks the user as anonymous if no other authentication occurred.
- `exceptionTranslationFilter` - Sets a 403 Forbidden HTTP response code if the user is not authenticated to view the requested page.
- `filterInvocationInterceptor` - Integrates the filter chain with other Acegi components.

In the example of a user requesting a page, the `authenticationProcessingFilter` detects that the user is not yet authenticated and redirects to the login page.

1.4.4 Step 3: Processing Login Credentials

When the user submits credentials from the login page, either spontaneously or after being redirected by the filter chain, the `authenticationProcessingFilter` also processes the login request. To do so, it calls `authenticationManager` bean that Acegi provides. The configuration of the `authenticationManager` bean in the WEB-INF/applicationContext-security.xml file establishes the different methods by which a user can be authenticated.

In the default installation of JasperServer the `authenticationManager` bean is configured as follows:

```
<bean id="authenticationManager" class="org.acegisecurity.providers.ProviderManager">
  <property name="providers">
    <list>
      <ref local="daoAuthenticationProvider"/>
      <ref local="anonymousAuthenticationProvider"/>
    </list>
  </property>
</bean>
```

The list of authentication providers determines the authentication mechanisms that are invoked:

- `daoAuthenticationProvider`. Compares the organization ID, username, and password from the login page to those stored in the JasperServer database. This is JasperServer's default form of authentication.
- `anonymousAuthenticationProvider`. Grants anonymous access to certain pages, by default only to the login page.

As we will see in the chapter for each external authentication mechanism, this list must be updated to configure JasperServer to use alternate providers.

1.4.5 Step 4: Sending Requested Content

When the authentication provider has successfully checked the user's credentials, the `authenticationManager` returns control to the `authenticationProcessingFilter`, but now the principal object has been created to establish the user session. The other filters in the chain can detect the user session and usually have no further action to perform.

The `JIAuthenticationSynchronizer` does perform a significant task for external authentication, as explained in the next section. But in the case of internal authentication, the job of the filter chain is done when the principal object has been created.

The user is now logged in with an active session. JasperServer processes the requested URL, or in the case of a spontaneous login, replies with the home page that is appropriate for the user. In fulfilling any request now, JasperServer knows the identity of the user and generates content based on the user's organization ID and roles found in the principal object. For example, administrators see a home page with items to manage JasperServer, and users within each organization can only see repository objects in their organization's folder.

When fulfilling any request after the user is authenticated, JasperServer still processes the entire filter chain in Acegi Security. However, in **Step 1: Receiving a Page Request**, the user session is recognized by its active principal object, no filter will be triggered, and JasperServer immediately processes the request to generate the content.

1.5 External Users in the JasperServer Database

When performing external authentication, JasperServer obtains all the information about a user from the external authority. After the `authenticationProvider` for the external authority has replied and the `AuthenticationProcessingFilter` is finished, the user session is represented by a principal object in run-time memory. This principal object includes any roles mapped from the external authority, as well as an organization ID, if necessary.

As a result, users, roles, and organizations are all defined externally. However, in order to create and enforce permissions based on these entities, JasperServer must store their values in the JasperServer database. For each external user, role, and organization, JasperServer creates a local user account, role definition, and organization folder to represent it and allow it to interact with JasperServer.

There are two aspects to managing these external values:

- Automatic synchronization of external users, roles, attributes, and organizations with the JasperServer database. This is the purpose of the filter called `JIAuthenticationSynchronizer`.
- Manual initialization of JasperServer to set up permissions based on the external credentials.

You must perform the initialization first, during the deployment of JasperServer in a production environment, but you must understand the synchronization process to perform the initialization. These two processes are explained in the following sections.

1.5.1 Automatic Synchronization of External Users

When a user is authenticated by an external authority, JasperServer has the principal object that contains the username, role names, and organization ID, if necessary, of the external user. The `JIAuthenticationSynchronizer` filter uses this information to update or create the corresponding structures in the JasperServer database:

- The organization ID may include a hierarchy of organizations to specify the parental lineage of the designated organization. If there is no organization with the given ID in the designated place in the organization hierarchy, that organization is created. Any parent organizations that do not exist are created as well. Organizations are created with the templates currently defined in the JasperServer repository.
- Any role names are compared to existing roles. If an organization ID is specified, only the roles in the organization are checked. If the roles do not exist, they are created. If an organization ID is specified, they are created within the organization.
- The username is compared to the user ID of existing user accounts in the JasperServer database. If an organization ID is specified, only the usernames in that organization are checked. If the user account exists, its list of assigned roles is replaced with the list of roles in the principal object. This is how the external user account is synchronized with the external authority.

  If the user does not exist, a user account is created. If an organization ID is specified, the user is created within the organization. Finally, all of the roles in the principal object are assigned the new user account.

For more information about how organizations, roles, and users are created, see the latest *JasperServer Professional Administration Guide*.

A user account created for an external user has the same structure as an internal user account but differs in the following ways:

- It does not store the password.
- It does not have any values for the optional fields of a user, such as the full name or email.

- It has a flag that marks it as an externally defined, which is used in the web interface to identify external users.

After synchronization, the external user fits in coherently with all the structures and mechanisms of JasperServer, in particular those required to verify authorization.

1.5.2 Initialization of JasperServer for External Users

Knowing how external users are synchronized, the system administrator must create the permissions in the repository that will perform the desired authorization in the production environment.

Your deployment procedure must include the following steps:

1. Configure the mapping of usernames, roles, and possibly organization IDs from your external authority into JasperServer. The mapping will depend on the external authentication mechanism, as described in the chapter for each mechanism.
2. Create test users in the external authority that are typical of users you expect to connect through JasperServer. The test users should be from every organization and have every role that you expect in your initial production environment.
3. Verify the external authentication and user mapping mechanisms by accessing JasperServer as the test users. Ensure that the data structure in the external authority conforms to the mapping in such a way to create the external users, roles and organizations that you expect.
4. Log into JasperServer with all of the test users, thereby creating all of the organizations and roles in your environment through the automatic synchronization.
5. Log into JasperServer as the system administrator to initialize your repository:
   a. Define your repository structure around the organization folders that were created, if applicable.
   b. Define all of your repository permissions using the roles that were created.

Creating and logging in as test users is not strictly required, but it validates the authentication and mapping mechanisms. You may use the administrative pages in JasperServer to create all the organizations and roles that your externally authenticated users will need, but you must ensure that their IDs will match exactly the values that are determined by the mapping.

The above procedure is necessary because the roles must exist in the JasperServer database before you can create permissions based on them. And if you are using organizations, roles must be defined within organizations, so they must exist as well.

When JasperServer enters into production, the external user accounts will be populated by the synchronization mechanism as the users log in. When the mapping is correct and consistent, the user population will have the same roles and possibly organizations that exist in your external authority, without having to manually import users or roles into JasperServer.

1.5.3 Maintenance of External Users

The advantage of an external authority is that it provides a single place to manage user accounts. Due to the mapping and synchronization, you do not need to manage the external users or roles in JasperServer, with small exceptions documented below.

1.5.3.1 User Management

The following paragraphs describe the impact on JasperServer when managing users in the external authority.

- Creating a new user - JasperServer will automatically create the new external user account when the user first accesses JasperServer. As long as the user relies on existing roles in an existing organization, the user will require no changes on JasperServer. If the user is associated with new roles, see **Role Management** below.
- Modifying a user - Various modifications are possible, depending on your external authority. The following list gives the impact for the most common modifications:
  - JasperServer does not store the passwords of external users, and is not impacted by changes to user passwords or user password policies on the external authority.
  - Even though JasperServer user accounts have a field for the user's full name, this value is not mapped. Only changes to values that are mapped to JasperServer will impact the external user. And because the mapping determines the username, roles, or organization ID, such changes will affect how the user is synchronized. Such changes are likely to be undesirable side-effects that can be detected by testing the mapping mechanism in **1.5.2, "Initialization of JasperServer for External Users," on page 11**.

- When changing the roles assigned to a user, the roles will be synchronized automatically at the beginning of the user's next session in JasperServer, as described in **1.5.1, "Automatic Synchronization of External Users," on page 10**. Roles that were previously unknown to JasperServer will be treated like new roles, and roles that are no longer assigned to any user will be treated like deleted roles, both described in **Role Management** below.

- Disabling or deleting a user - When the user can no longer authenticate with the external authority, he or she cannot access JasperServer. The external user account synchronized during the last user session will remain in the JasperServer database. This user account has no impact on JasperServer, and you can safely delete it, if desired.

1.5.3.2 Role Management

The following paragraphs describe the impact on JasperServer when modifying role definitions in the external authority.

- Creating a new role - Role definitions are not directly mapped to JasperServer, only roles that are assigned to users who log in are mapped. When you create a new role and assign it to a user who accesses JasperServer, determine which case applies:
  - The role is significant to access control that impacts JasperServer. You will need to initialize this role in JasperServer with a test user and define all necessary repository authorization rules to secure your data before you deploy this role to real users. This case is similar to the initialization during the deployment phase, describe in **1.5.2, "Initialization of JasperServer for External Users," on page 11**.
  - The role is not significant to users within JasperServer. The synchronization will automatically create the role and assign it to users according to their mapping, but with no authorization rules based on the role, it will have no impact.

  In practice, you will find that only a subset of the rules in your external authority will be applicable to JasperServer. Some of these roles may be used in other applications, some may be created specifically for managing users in JasperServer. Because maintenance of your enterprise-wide user database may affect JasperServer, you should identify the user management procedures that impact JasperServer, such as modifications to roles, and document the procedures for adding them to JasperServer as described above.
- Modifying role membership - Changes in role membership will be reflected the next time the role members start a new session on JasperServer. Roles that were previously unknown to JasperServer will be treated like new roles described above, and roles that are no longer assigned to any user will be treated like deleted roles described below.
- Deleting a role - When the role is deleted, users will no longer have the role, and it will be removed from each external user during the synchronization at the beginning of the next session. The external role definition will remain in the JasperServer database, as well as any permissions that reference the role. The defunct role may still be assigned to external users if they were deleted or have not logged in since the role was removed. This unused role definition has no impact on JasperServer, and you can safely delete it, if desired.

1.5.3.3 Organization Management

The following paragraphs describe the impact on JasperServer when modifying organizations defined in the external authority.

- Adding an organization - Organization definitions are not directly mapped to JasperServer, rather a new organization ID will be mapped when synchronizing the first user in the organization that accesses JasperServer. At that time, the synchronization will create the organization and create the user within it, along with any roles assigned to the user. You may want to create test users in the new organizations first, so you can configure the new organization folder and assign repository permissions before actual users have access. However, if you use roles with the same name in every organization, you may also configure the organization folder templates so that the default contents work with the known role names (there will be roles with the same name in each organization).
- Modifying an organization - Users who are added to an organization will be handled like new users described in **1.5.3.1, "User Management," on page 11**. Users who are removed from an organization will be handled like deleted users, provided they can no longer authenticate with the external authority. If users are moved to a different organization, or the mapping for their new status still succeeds, their external user accounts will be synchronized the next time they access JasperServer and authenticate successfully.
- Deleting an organization - Because organization definitions are not mapped directly, deleting an organization has the same effect as removing each of its users, which is described above.

2 LDAP AUTHENTICATION

Lightweight Directory Access Protocol (LDAP) is one of the most popular architectures for enterprise directories. By centralizing all user management in an LDAP directory, applications across the enterprise can share the same user database, and administrators do not need to duplicate user accounts in every application.

This chapter shows how JasperServer's default authentication mechanism using Acegi Security can be configured to perform external authentication with LDAP. As part of the authentication process, JasperServer also synchronizes the user information such as roles and organization ID from the LDAP structure into its own user database.

LDAP authentication does not provide single sign-on (SSO) functionality. You must implement additional mechanisms and configure their use within JasperServer to enable SSO with LDAP, the details of which are beyond the scope of this document.

This chapter assumes you are familiar with LDAP servers and the structure of the data they contain, in particular the format of distinguished names (DNs) and relative distinguished names (RDNs) that create structure and identify entries in LDAP.

This chapter contains the following sections:

2.1 Overview of External LDAP Authentication

This section explains how JasperServer performs external authentication with an LDAP server, highlighting the differences with section 1.3, "Default Internal Authentication," on page 7. The following diagram shows the general steps involved in external LDAP authentication:

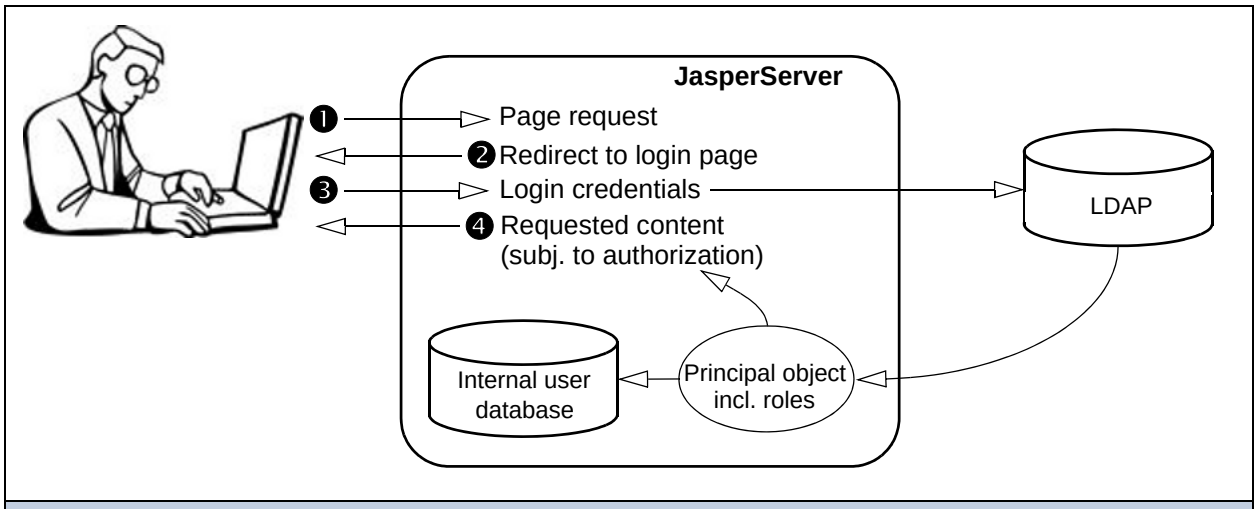

Figure 2-1 General Steps of External LDAP Authentication

The following steps explain the interaction between the user's browser, JasperServer, and the LDAP server:

1. An unauthenticated user requests any page in JasperServer.

   Often, users will bookmark the login page and begin directly at step 3, but this step covers the general case and secures every possible access to JasperServer. For example, this step applies when a user clicks on the screen of an expired session or if a user is given the direct URL to a report within JasperServer.

2. JasperServer detects that the user is not logged in and replies with a redirect to the login page.

   Users log into JasperServer's own login page even though the user credentials are not verified internally.

3. The user enters a username and password. Even on a server with multiple organizations, the organization ID must be left blank because it will be determined from the external LDAP authority.

   JasperServer performs a search on the LDAP server with the given credentials. If they are valid, the search will return the user's entry in the LDAP directory, which JasperServer uses to create a principal object. The user's roles and organization ID are mapped from the LDAP entry and stored in the principal object.

   The username, roles and organization information are also synchronized with the JasperServer database, where the user account is marked as an external user. The user is now authenticated, the principal object represents the user session, and the JasperServer environment is coherent with the user's roles and organization.

   For more information about the external user account that JasperServer synchronizes from the user information returned by the external authority, see section 1.5, "External Users in the JasperServer Database," on page 10.

4. As with the default internal authorization, JasperServer now sends the requested content to the user, or if none was specified, the home page appropriate for the user.

   Content that is sent to the user is subject to authorization. For example the home page has different options for administrators and regular users, as determined by the roles of the user in the principal object. Or if the user is viewing the repository, the folders and objects that will be returned will be based on the organization ID and roles in the principal object.

When comparing these steps with those in section 1.3, "Default Internal Authentication," on page 7, there are three significant differences, all in step 3:

- JasperServer verifies the credentials through LDAP instead of with its internal user database.
- The roles and organization ID in the user's principal object are mapped from the LDAP response.
- The JasperServer database must be synchronized with any new information in the user's principal object.

2.2 Configuring JasperServer for LDAP Authentication

The differences listed in the previous section reflect the changes that need to be made in the Acegi Security configuration:

- Add LDAP as an authentication provider in the `authenticationManager` bean and configure it to access your LDAP server and search for users.
- Configure the LDAP authentication provider to map roles and organizations, if applicable.
- The synchronization is already handled by the `JIAuthenticationSynchronizer`.

Because the response to page requests is the same as with internal authentication, there is no need to change the default filter chain. However, as user requests are processed by the filter chain, it will behave differently in the following two ways:

- The `authenticationProcessingFilter` will now call the LDAP authentication provider.
- The `JIAuthenticationSynchronizer` will now synchronize every external user instead of being mostly inactive.

The following sections explain how to make the configuration changes presented above.

2.2.1 Files to Modify

All configuration for external LDAP authentication is made in the following files:

- <application-server-path>/jasperserver[-pro]/WEB-INF/applicationContext-security.xml
- <application-server-path>/jasperserver-pro/WEB-INF/applicationContext-multiTenancy-security.xml if you installed JasperServer 3.5 Professional that has the new organization architecture. This file does not exist in other releases.

Where:

- <application-server-path> is the location of the application server in which you deployed JasperServer.
- -pro is part of the default pathname if you installed JasperServer Professional.

The rest of this document refers to file names alone. Refer to the paths above to find the file in your deployment.

Configuring JasperServer for external LDAP authentication is a common customization that is provided by Acegi Security. As a result, Acegi provides the beans to implement it, and the applicationContext-security.xml file contains commented code for many of the modifications given in the following sections. You will need to uncomment the code and make modifications according to your LDAP configuration.

2.2.2 Beans to Configure

This section explains the beans that are involved in external LDAP authentication and how they relate to each other. They are listed here in the order they appear in the files above.

- `authenticationManager` - Name of a bean of class `ProviderManager` for performing authentication, configured with:
  - `ldapAuthenticationProvider` - Bean for performing LDAP authentication, configured below.
- `initialDirContextFactory` - Name of a bean of class `DefaultInitialDirContextFactory` configured with the LDAP connection parameters.
- `userSearch` - Name of a bean of class `FilterBasedLdapUserSearch` configured with user search parameters.
- `ldapAuthenticationProvider` - Name of a bean of class `LdapAuthenticationProvider`, configured with:
  - `BindAuthenticator` - Class of an anonymous bean for finding user entries, configured with:
    - `initialDirContextFactory` - Required for accessing the LDAP server.
    - User DN patterns and/or `userSearch` - Required for finding the user entry.
  - `DefaultLdapAuthoritiesPopulator` - Class of an anonymous bean configured with:
    - `initialDirContextFactory` - Required for accessing the LDAP server.
    - Role search parameters - Required for finding user roles to map.
- `ldapExternalUserProcessor` - Name of a bean o f class `LdapExternalUserProcessor` in JasperServer 3.5 Professional, configured with:
  - `initialDirContextFactory` - Required for accessing the LDAP server.
  - Organization mapping parameters - Required for finding the user's organization ID.

2.3 Adding the ldapAuthenticationProvider

In applicationContext-security.xml, locate the `authenticationManager` bean and add `ldapAuthenticationProvider` to the top of the list of providers. The `ldapAuthenticationProvider` bean is provided in the Acegi Security package.

```
<bean id="authenticationManager"class="org.acegisecurity.providers.ProviderManager">
  <property name="providers">
    <list>
      <ref local="ldapAuthenticationProvider"/>
      <ref local="daoAuthenticationProvider"/>
      <ref local="anonymousAuthenticationProvider"/>
    </list>
  </property>
</bean>
```

As shown in the example above, you can list other authentication providers with LDAP. The `authenticationManager` bean will attempt to authenticate a user session with each provider in the list, in the order they appear. When one of the providers successfully authenticates the user, the rest of the providers are skipped.

The `daoAuthenticationProvider` is the default internal authentication using the JasperServer database, so in this example, any user credentials that are not recognized by the LDAP directory will be compared to those of locally defined users. For example, during a default installation, the `jasperadmin` and `superuser` accounts are created for JasperServer administrators. If you wish to continue using these accounts after configuring LDAP authentication for your end users, keep the `daoAuthenticationProvider` in the list of authentication providers.

The JasperServer database contains accounts for all of the external users that have previously logged into JasperServer. However, these external accounts do not contain passwords and cannot be used for authentication, for example in the case that the LDAP server in unavailable.

If LDAP is your primary authentication mechanism for the majority of users, you should place it first in the list of providers for performance reasons. Alternatively, if you wish to enforce LDAP as the only authentication mechanism, remove all other authentication providers except for the `anonymousAuthenticationProvider`. Anonymous authentication is necessary to display the JasperServer login page, however anonymous access to all other pages is disabled by default.

2.4 Setting the LDAP Connection Parameters

Several beans need to access the LDAP directory and must be configured with the connection parameters for the LDAP server. This is done through the configuration of a helper bean called `initialDirContextFactory` that will later be given to the constructors of the beans that need it.

In applicationContext-security.xml, locate the `initialDirContextFactory` bean and uncomment it if necessary. Specify the following information:

- The URL of your LDAP server, including the base DN.
- The distinguished name (DN) of your LDAP administrator.
- The password of your LDAP administrator.

If your LDAP server is configured to allow anonymous user lookup, you do not need to specify the `managerDn` or `managerPassword` properties.

The following example shows the syntax of the bean's constructor and properties:

```
<bean id="initialDirContextFactory"
      class="org.acegisecurity.ldap.DefaultInitialDirContextFactory">
  <constructor-arg value="ldap://hostname:389/dc=example,dc=com" />
  <property name="managerDn"><value>cn=Administrator,dc=example,dc=com</value></property>
  <property name="managerPassword"><value>password</value></property>
</bean>
```

2.5 Setting the LDAP Search Parameters

The `ldapAuthenticationProvider` bean must be initialized with a bean of the class `BindAuthenticator` that encapsulates search parameters for finding users in the LDAP directory. There are two ways of finding users in the directory:

- Matching RDN patterns based on the login name provided by the user. Use this method if the login name appears in the DN of your user entries, and your user entries are in a fixed branch of your LDAP directory.
- Performing a search for the login name provided by the user. Use this method if the login name is the value of an attribute that does not appear in the RDN, or if your user entries are located in a more complex structure. In particular, if you are authenticating users for one or more organizations, it is likely that user entries are not in a single branch of your directory.

In terms of performance, matching patterns is faster because doing so checks for the existence of a DN in the LDAP directory as opposed to performing a search. You can configure both matching and searching by combining the instructions in the following subsections. In this case, the patterns will be matched first, and the search will be performed only if no match is found.

2.5.1 Specifying userDnPatterns

If you have a fixed structure of user entries and the login name of the user appears in the RDN of your user entries, you can easily configure the `BindAuthenticator` bean with patterns to match them.

In applicationContext-security.xml, locate the `ldapAuthenticationProvider` bean and uncomment it if necessary. The `BindAuthenticator` bean is the first constructor argument.

In the properties that configure `BindAuthenticator`, the `userDnPatterns` property contains a list of values that give patterns for matching the RDNs of user entries. For each value in the list, JasperServer will substitute the login name for the `{0}` placeholder, then create a DN by appending the base DN from the LDAP URL. The LDAP URL is the one specified in section **2.4, "Setting the LDAP Connection Parameters," on page 16**.

JasperServer will attempt to bind to the LDAP directory with the DN created with each pattern, in the order they are given.

In the example below, JasperServer will look for a user whose given login name appears in the `uid` attribute of the RDN in two different branches of the LDAP directory:

```
<bean id="ldapAuthenticationProvider"
      class="org.acegisecurity.providers.ldap.LdapAuthenticationProvider">
  <constructor-arg>
    <bean class="org.acegisecurity.providers.ldap.authenticator.BindAuthenticator">
      <constructor-arg><ref local="initialDirContextFactory"/></constructor-arg>
      <property name="userDnPatterns"><list>
        <value>uid={0},ou=administrators</value>
        <value>uid={0},ou=people</value></list></property>
    </bean>
  </constructor-arg>
  ...
</bean>
```

When you specify the `userDnPatterns` property in the `BindAuthenticator` bean, you do not need to uncomment the `userSearch` bean described in the next section, unless you want to enable both types of user search.

2.5.2 Specifying userSearch

To perform a search to locate user entries, configure the helper bean called `userSearch` that is in turn used to configure the `BindAuthenticator` bean.

In applicationContext-security.xml, locate the `userSearch` bean and uncomment it if necessary. Specify the following information:

- An optional branch DN where user entries are located. If not specified, the search will include your entire LDAP directory starting from the root.
- The attribute to compare with the login name. This expression will substitute the login name entered by the user for the `{0}` placeholder to perform the search.
- Whether or not the search should extend to all subtrees beneath the branch DN or beneath the root when no branch DN is specified.

The following example shows the syntax of the bean's constructor and property:

```
<bean id="userSearch"
      class="org.acegisecurity.ldap.search.FilterBasedLdapUserSearch">
  <constructor-arg index="0"><value>ou=people</value></constructor-arg>
  <constructor-arg index="1"><value>(uid={0})</value></constructor-arg>
  <constructor-arg index="2"><ref local="initialDirContextFactory" /></constructor-arg>
  <property name="searchSubtree"><value>true</value></property>
</bean>
```

The combination of these three parameters lets you optimize the search for your user entries and reduce the load on your LDAP directory. For example, if your users are located in a dedicated branch of your LDAP structure, specify it in the first constructor argument to avoid searching the entire tree.

In an environment where you are authenticating users for one or more organizations in JasperServer, the search parameters must be able to locate all users for all organizations.

Finally, in applicationContext-security.xml, locate the `ldapAuthenticationProvider` bean and uncomment it as well. Specify the `userSearch` bean in the `userSearch` property to configure the `BindAuthenticator` bean in the first constructor argument of the `ldapAuthenticationProvider` bean. The following example shows the `BindAuthenticator` bean configured with `userSearch`:

```
<bean id="ldapAuthenticationProvider"
      class="org.acegisecurity.providers.ldap.LdapAuthenticationProvider">
  <constructor-arg>
    <bean class="org.acegisecurity.providers.ldap.authenticator.BindAuthenticator">
      <constructor-arg><ref local="initialDirContextFactory"/></constructor-arg>
      <property name="userSearch" ref="userSearch"/>
    </bean>
  </constructor-arg>
  ...
</bean>
```

2.5.3 Alternate to Bind Authentication

In Acegi Security, the task of the `BindAuthenticator` bean is to access the LDAP directory to determine the DN of the user. To do this, it performs a "bind" authentication on the LDAP directory, which consists of the following steps:

1. Using either the pattern matching or the search described above, find a candidate user entry based on the login name.
2. Attempt to login to the LDAP server, known as binding, as the candidate with the login password.
3. A successful bind indicates the right user was found.

Bind authentication with the `BindAuthenticator` bean is default behavior when configuring Acegi Security for LDAP authentication. However, Acegi Security provides an alternate authentication method based on password comparison:

1. Use the administrator credentials in the `initialDirContextFactory` bean to log into the LDAP server.
2. Find a candidate user entry.
3. Retrieve the candidate's password attribute and compare it to the login password, or send the login password for comparison by the LDAP server.

The alternate authentication method is implemented by Acegi Security in the `PasswordComparisonAuthenticator` class. However, configuring Acegi with this class is beyond the scope of this documentation. For more information, see the Acegi documentation and the Acegi Javadoc.

2.6 Mapping the User Roles

The roles that an external user will have in JasperServer are based on the static groups to which the user belongs in LDAP. The mapping defines the location of the group definitions in LDAP, how to find the groups to which the user belongs, and any transformation of the group name for use in JasperServer as a role name. The mapping for user roles is configured in a bean of the class `DefaultLdapAuthoritiesPopulator`, itself part of the configuration of the `ldapAuthenticationProvider` bean.

Some LDAP servers also support other user-grouping mechanisms, such as `nsrole` in the Sun Directory Server. These mechanisms may be mapped into JasperServer roles through the configuration parameters below, by extending the `DefaultLdapAuthoritiesPopulator` class, or a combination of both. As another example, you can customize the populator bean to map administrators of your LDAP server into system administrators of JasperServer. Such configurations are beyond the scope of this document.

In applicationContext-security.xml, locate the `ldapAuthenticationProvider` bean that you uncommented when specifying the search parameters. In the second constructor argument, specify the following information:

- An optional branch DN where group entries are located. If not specified, the search will include your entire LDAP directory starting from the root.
- The group role attribute, which is the attribute name in the RDN of the group entry. The value of this attribute will be the name that is mapped to a role.
- A group search filter that will locate entries representing groups to which the user belongs. For static groups, this filter should detect entries with the `groupofuniquename` object class and with a `uniqueMember` value that matches the login name given by the user.
- Whether or not the search should extend to all subtrees beneath the branch DN or beneath the root when no branch DN is specified.
- Whether or not the group name should be converted to all upper case when mapped to a role name in JasperServer.
- An optional prefix to add to the group name when mapped to a role name in JasperServer.

The following example shows the syntax of the constructor arguments and properties:

```
<bean id="ldapAuthenticationProvider"
      class="org.acegisecurity.providers.ldap.LdapAuthenticationProvider">
  <constructor-arg>
    ...
  </constructor-arg>
  <constructor-arg>
    <bean class="org.acegisecurity.providers.ldap.populator.
                 DefaultLdapAuthoritiesPopulator">
      <constructor-arg index="0"><ref local="initialDirContextFactory"/></constructor-arg>
      <!-- optional branch DN for roles -->
      <constructor-arg index="1"><value></value></constructor-arg>
      <property name="groupRoleAttribute"><value>cn</value></property>
      <property name="groupSearchFilter"><value>
        (&amp;(uniqueMember={0})(objectclass=groupofuniquenames))</value></property>
      <property name="searchSubtree"><value>true</value></property>
      <!-- Other possible properties
      <property name="convertToUpperCase"><value>true</value></property>
      <property name="rolePrefix"><value></value></property>
      -->
    </bean>
  </constructor-arg>
</bean>
```

With the `DefaultLdapAuthoritiesPopulator`, all roles found for a user will be converted to uppercase and given the prefix `ROLE_` when created in JasperServer. To change this mapping, specify other values for the optional `convertToUpperCase` and `rolePrefix` properties.

2.7 Mapping the User Organization (JasperServer 3.5 Professional Only)

In JasperServer 3.5 Professional, all users and roles belong to an organization. In turn, the organization determines the folders that the user may access in the repository. Therefore, the final part of mapping is determining the ID of the organization in which the external user and roles should be placed.

LDAP is well suited to mapping users into organizations, because LDAP itself has a hierarchal structure of user entries that is often used to represent separate organizations such as the internal departments of a company. The LDAP structure is reflected in the elements of the RDN of each user entry, and JasperServer maps this RDN into an organization or hierarchy or organizations for the external user. For example, the users `uid=jack,ou=audit,ou=fincance,dc=example,dc=com` and `uid=jill,ou=accounting,ou=finance,dc=example,dc=com` could be mapped to the organizations `audit` and `accounting`, respectively, both of which are sub-organizations of `finance`.

In order to ensure consistency, JasperServer must create the organization of any external user if the organization does not already exist. JasperServer will also create any organization that does not exist in the hierarchy of organizations mapped from the user RDN. To avoid “stray” organizations that are outside of your intended hierarchy, test your mapping against all potential user DNs in your LDAP directory. For more information about mapping external users, including strategies to initialize your JasperServer instance, see section **1.5, “External Users in the JasperServer Database,” on page 10**.

In the default installation of JasperServer 3.5 Professional, there is a single, transparent organization that mimics the appearance of previous versions without organizations. However, you must still specify a mapping to its organization ID, so that external users are not placed in new, separate organizations. The configuration for this case is presented separately after the general case below.

As with role mapping, you can create custom organization mappings by extending the class `LdapExternalUserProcessor`. Such a configuration is beyond the scope of this document.

2.7.1 Mapping to an Organization Hierarchy

In applicationContext-multiTenancy-security.xml, locate the `mtUserAuthorityServiceTarget` bean and its `userProcessors` property. In the list of user processors, uncomment the reference to the `ldapExternalUserProcessor` bean.

Further in the file, locate the `ldapExternalUserProcessor` bean and uncomment it. Specify the following information to map the RDN of the user to the hierarchy of organizations in JasperServer:

- Whether or not the base DN, also called root DN, should be mapped along with the RDN. For example, if you do not exclude the root DN of `dc=example,dc=com`, it will map to a top-level organization whose ID is `com` containing a sub-organization whose ID is `example`. The base DN is part of the LDAP URL specified in section **2.4, "Setting the LDAP Connection Parameters," on page 16**.
- A list of attributes names that determines which RDN values should be mapped to organization names. The attributes in this list determine the RDNs that will create a hierarchy of organizations in JasperServer. For example, if you specify the value `ou`, each RDN with `ou=<name>` creates an organization in the hierarchy.
- A root organization ID under which any mapped organizations are created as sub-organizations. If the root organization ID is not specified, and the DN of a user does not map to any organization ID, the user will have a null organization ID. The null organization ID is usually reserved for special users such as the system administrator, so you should avoid this sort of mapping either by specifying a root organization ID or ensuring that every user maps to an organization ID.

The following example shows the syntax of the beans and their properties:

```
<bean id="mtUserAuthorityServiceTarget"
  ...
  <property name="userProcessors">
    <list>
      <!-- For LDAP authentication -->
      <ref bean="ldapExternalUserProcessor"/>
    </list>
  </property>
</bean>

<bean id="ldapExternalUserProcessor" class="com.jaspersoft.jasperserver.
          multipleTenancy.ldap.LdapExternalUserProcessor">
  <property name="initialDirContextFactory" ref="initialDirContextFactory" />
  <property name="multiTenancyService">
    <ref bean="internalMultiTenancyService"/></property>
  <property name="excludeRootDn" value="true"/>
  <!-- only following RDNs will matter in creating the organization hierarchy -->
  <property name="organizationRDNs">
    <list>
      <value>o</value>
      <value>ou</value>
    </list>
  </property>
  <property name="rootOrganizationId" value=""/>
</bean>
```

2.7.2 Mapping to a Single Organization

In the default installation of JasperServer 3.5 Professional, there is a single organization that mimics the behavior of JasperServer without organizations. To map all external LDAP users to the single, default organization, regardless of any

structure in the LDAP directory, follow the instructions in section 2.7.1, “Mapping to an Organization Hierarchy,” on page 21, but specify the following property values:

```
<bean id="ldapExternalUserProcessor" class="com.jaspersoft.jasperserver.
          multipleTenancy.ldap.LdapExternalUserProcessor">
  <property name="initialDirContextFactory" ref="initialDirContextFactory" />
  <property name="multiTenancyService">
    <ref bean="internalMultiTenancyService"/></property>
  <property name="excludeRootDn" value="true"/>
  <property name="organizationRDNs"><list></list></property>
  <property name="rootOrganizationId" value="organization_1"/>
</bean>
```

When there is no list of organization RDNs, the user DN does not map to any organization hierarchy. Finally, `organization_1` is the ID of the single, default organization, so all external LDAP users are placed there.

2.8 Restarting JasperServer

When you have configured all the beans in the appropriate files, restart JasperServer for the changes to take effect.

To test your configuration, navigate to the JasperServer login page. If JasperServer and LDAP are configured correctly, you can log into JasperServer with credentials that are stored in your LDAP directory.

2.9 Authentication with Computer Associates SiteMinder

Computer Associates SiteMinder is an LDAP-based user directory that can be used to authenticate users through the `ldapAuthenticationProvider` provided by Acegi Security. Simply follow the configuration instructions in this chapter and substitute the connection parameters, search parameters, role mappings, and organization mappings that are specific to your SiteMinder directory structure.

2.10 Authentication with Microsoft Active Directory

Microsoft Active Directory can also be used to authenticate users through the `ldapAuthenticationProvider` provided by Acegi Security.

For the purposes of basic external authentication, the only difference in configuration between Active Directory and a standard LDAP server is the need to search for the `sAMAccountName` attribute containing the user’s login name. Because of this specificity, you must use the `BindAuthenticator` bean, along with the `userSearch` bean and corresponding property in `BindAuthenticator`.

The following example shows all the configuration beans in applicationContext-security.xml for LDAP authentication, including the special syntax for Active Directory:

```
<bean id="authenticationManager" class="org.acegisecurity.providers.ProviderManager">
  <property name="providers">
    <list>
      <ref local="ldapAuthenticationProvider"/>
      <ref local="daoAuthenticationProvider"/>
      <ref local="anonymousAuthenticationProvider"/>
    </list>
  </property>
</bean>

<bean id="initialDirContextFactory"
      class="org.acegisecurity.ldap.DefaultInitialDirContextFactory">
  <constructor-arg value="ldap://hostname:389/dc=ADexample,dc=com"/>
  <property name="managerDn"><value>cn=Administrator,dc=ADexample,dc=com</value></property>
  <property name="managerPassword"><value>password</value></property>
</bean>

<bean id="userSearch" class="org.acegisecurity.ldap.search.FilterBasedLdapUserSearch">
  <constructor-arg index="0"><value>cn=Users</value></constructor-arg>
  <constructor-arg index="1"><value>(sAMAccountName={0})</value></constructor-arg>
  <constructor-arg index="2"><ref local="initialDirContextFactory"/></constructor-arg>
  <property name="searchSubtree"><value>true</value></property>
</bean>

<bean id="ldapAuthenticationProvider"
      class="org.acegisecurity.providers.ldap.LdapAuthenticationProvider">
  <constructor-arg>
    <bean class="org.acegisecurity.providers.ldap.authenticator.BindAuthenticator">
      <constructor-arg><ref local="initialDirContextFactory"/></constructor-arg>
      <property name="userSearch"><ref local="userSearch"/></property>
    </bean>
  </constructor-arg>
  <constructor-arg>
    ...
  </constructor-arg>
</bean>
```

In the example above, the role mapping is omitted, as is the organization mapping. You must include a role mapping for any roles you wish to import into JasperServer, and you must include an organization mapping if you use JasperServer 3.5 Professional.

3 CAS AUTHENTICATION

Central Authentication Service (CAS) is an open source, Java-based authentication server that includes a mechanism for single sign-on (SSO) across web applications, including those running on different application servers. When a user requests a page from a CAS-enabled web application, the application redirects the user to the CAS server login page. Thereafter, logged-in users can navigate between all participating applications without needing to log in again. Each application communicates with the CAS server in the background to verify the user is valid before providing access to its resources.

With the CAS protocol, the client application such as JasperServer never receives or transmits the user's password. As a result, the client application does not need to apply any encryption to protect passwords.

This chapter shows how JasperServer's default authentication mechanism using Acegi Security can be configured to perform external authentication with CAS. In order to participate in SSO, the JasperServer configuration is more complex than with simple authentication. In particular, you must modify the filter chain, replace certain filter implementations, and add helper beans to the configuration files.

In addition to the beans provided by Acegi Security to support CAS, JasperServer provides additional helper beans for CAS integration. The implementation of these custom beans is sufficient to enable CAS authentication but may not provide enough functionality in a complex deployment. However, further customization of these helper beans is beyond the scope of this document.

This chapter assumes you are familiar with security concepts such as certificates, tokens, and cookies that are exchanged with CAS. The first section explains how to install a CAS server for testing, and all examples refer to the test server. All examples assume you are using the Apache Tomcat application server. When configuring JasperServer to use CAS in production, you must take into account any difference between your test and production environments, such as differences in application servers and the contents of certificates.

This chapter contains the following sections:

3.1 CAS Server for Testing

CAS is maintained and distributed by Jasig, a consortium of educational institutions and commercial affiliates sponsoring open source software projects. Jasig provides a CAS server packaged as a web application that includes a built-in authentication module that is suitable for testing. The built-in authentication module will accept any username and password combination where the username and password are the same. You can download the server from the JA-SIG download page http://www.jasig.org/cas/download.

As described in the next section, the CAS validation service only accepts requests using a secure transport. This means that you must have a valid certificate on your CAS server machine, and your CAS client, the JasperServer JVM, must be configured to trust that certificate. There are two important points to keep in mind:

- Test with the CAS server on a separate machine, not the localhost where JasperServer is installed. For this purpose, a virtual machine is acceptable.
- Most issues in configuring CAS are caused by the improper use of certificates. The single most common failure is that the hostname in the server's certificate doesn't match the actual hostname.

To create a certificate for the server you will need to use the Java `keytool` utility. Run the following command on the host machine for the CAS server:

```
keytool -genkey -alias tomcat -keyalg RSA -validity 365 -keystore <filename>
```

You will be prompted for several pieces of information, two of which are critical. When prompted for “Your first and last name” you must put in the hostname of the CAS server. When asked for the keystore password use `changeit` to match what Apache Tomcat uses by default.

After installation of the CAS server, configure the Apache Tomcat application server that is running the CAS server so that it uses the certificate in the keystore created above. Modify the file $CATALINA_HOME/conf/server.xml, locate the commented section about setting up a secure HTTPS connector and follow the instructions it contains. Restart the Tomcat server and test that it accepts HTTPS connections.

For further information about CAS, including deployment information, documentation, and community links, refer to the Jasig CAS website http://www.jasig.org/cas. In particular, the page http://www.jasig.org/cas/server-deployment/solving-ssl-issues can help you deploy your certificates.

CAS server also happens to be based on Spring Security, a newer version of Acegi Security. In a production environment, you need to replace the built-in authentication for testing with an external authority that validates your users when they log into CAS. As with JasperServer, you can configure CAS with a variety of external authorities to suit your needs. Follow the CAS documentation to ensure you create a secure and robust configuration on your CAS server.

3.2 Overview of External CAS Authentication

This section describes how JasperServer integrates the Acegi Security mechanisms to perform external CAS authentication, including SSO.

The original CAS implementation, now called the CAS 1 architecture, had a centralized CAS server that processed requests according to a protocol. The overview in this section explains the major steps involved in the protocol between the CAS server and JasperServer, as well as the Acegi Security beans involved.

The newer CAS 2 architecture adds decentralized proxy authentication for the CAS server, but the underlying protocols are conceptually the same from the JasperServer perspective. This chapter does not explain CAS proxies, but does cover the Acegi Security beans that are used to configure JasperServer’s response to CAS proxies. Some explanation about how Acegi Security beans interact with proxy responses is given in section **3.11, “Detailed Protocol Exchange,” on page 33**.

For more information, see http://www.jasig.org/cas/cas1-architecture and http://www.jasig.org/cas/cas2-architecture.

The general protocol during external CAS authentication is shown in the following diagram:

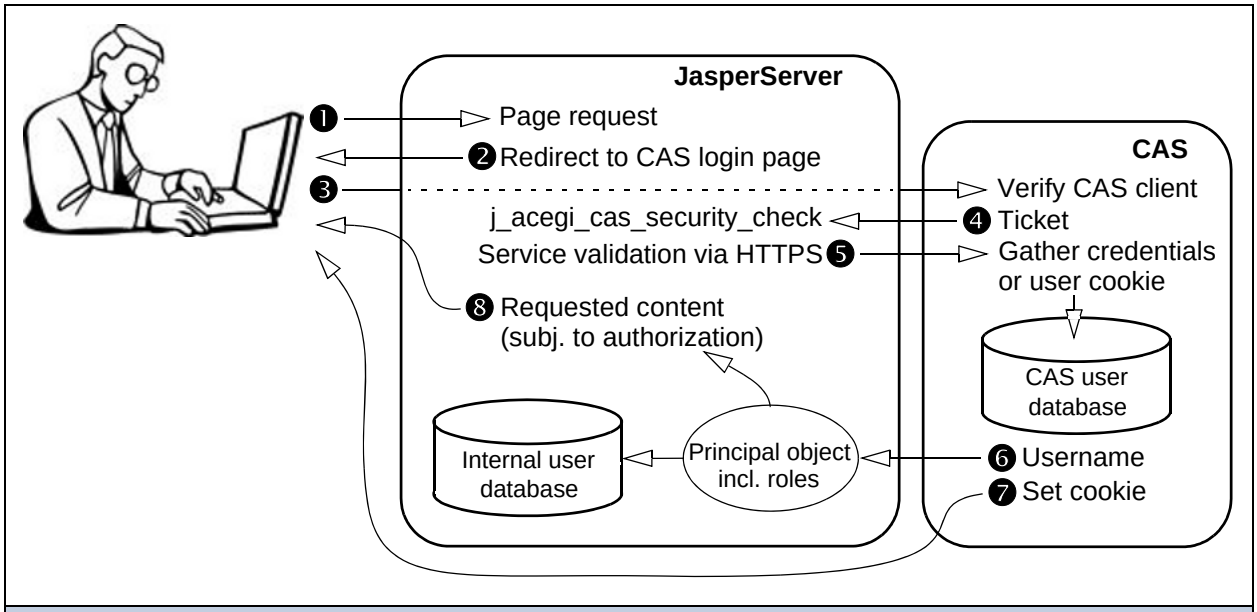

Figure 3-1 General Steps of External CAS Authentication

The following steps explain the interaction between the user's browser, JasperServer, and the CAS server:

1. An unauthenticated user requests any page in JasperServer.

   With SSO, JasperServer does not have a login page for users to bookmark. Instead, users can bookmark their home page or any page that allows it in JasperServer. As a result, every user goes through this step for every page they request from JasperServer, thereby securing every possible access to JasperServer.

2. JasperServer detects that the user is not logged in and replies with a redirect to the CAS login page.

   The redirect URL contains a parameter which is another URL that tells CAS how it can request service validation from JasperServer. For example:

   http://hostname1/cas/login?service=http://hostname2/jasperserver/j_acegi_cas_security_check.

   URLs may appear translated when viewed in browsers or in log files. For example, the previous URL can be written http://hostname1/cas/login?service=http%3A%2F%2Fhostname2%2Fjasperserver%2Fj_acegi_cas_security_check.

3. The user's browser requests the CAS login page from the CAS server.
4. However, before serving the CAS login page to the user, CAS requests the service validation from JasperServer.

   It extracts the URL parameter from the redirect URL and sends it a randomly generated number called a ticket, for example: http://hostname2/jasperserver/j_acegi_cas_security_check?ticket=ST-8670-123buTvFFjo980

5. The JasperServer filter chain processes the service validation request and the `casAuthenticationProvider` is invoked. It takes the ticket and establishes a secure connection to the CAS server to verify the credentials of the user associated with the ticket, who is only known to the CAS server.

   Only once the service is validated by the handshake over the secure connection does the CAS server attempt to get the user's credentials. If the CAS server detects its own session cookie on the user's browser, this indicates the user has logged in successfully before, and the CAS can retrieve the credentials. If there is no cookie, CAS serves its own login page to the user, finally. When the user enters a username and password, CAS verifies them against its own database.

6. In either case, the CAS server replies to JasperServer with the validated username.

   JasperServer maps the username to a predefined set of roles to create the principal object that establishes the user's session. The username and roles are also synchronized with the JasperServer database, where the user account is marked as an external user. For more information about the external user account, see section **1.5, "External Users in the JasperServer Database," on page 10**.

7. If the user submitted a valid username and password on the login page, the CAS server sets a session cookie on user's browser to be used for single sign-on with other applications that use the same CAS server.

   If the user has disabled cookies, the CAS login protocol still works, but the single sign-on mechanism fails. When no cookie is detected on the user's browser, the user is prompted to log into the CAS server every time he or she accesses a client application.

8. As with the default internal authorization, JasperServer now sends the requested content to the user.

   Content that is sent to the user is subject to authorization. For example the home page has different options for administrators and regular users, as determined by the roles of the user in the principal object.

When comparing these steps with those in section **1.3, "Default Internal Authentication," on page 7**, there are several significant differences:

- JasperServer must redirect to the CAS login page instead of its own.
- JasperServer must receive the ticket as part of the security check.
- JasperServer must process the ticket and respond to the security check over HTTPS.
- The roles in the user's principal object are mapped from the username in the CAS response.
- The JasperServer database must be synchronized with any new information in the user's principal object.

3.3 Configuring JasperServer for CAS Authentication

The differences listed in the previous section reflect the changes that need to be made in the Acegi Security configuration and on the JasperServer host. The items in the following list address each of the differences in the list above, in same order:

- To perform the redirect upon receiving an unauthenticated request, replace the pre-filter for the `authenticationProcessingFilter` with one tailored for CAS.
- To receive the ticket through the security check URL, add a filter chain for that specific URL, and then modify the behavior of the `authenticationProcessingFilter` in that chain.
- To process the ticket, add CAS as an authentication provider in the `authenticationManager` bean and configure it to receive the response from the CAS server or its proxy.

  To establish the HTTPS connection, configure the JVM on the JasperServer host to trust the CAS server certificate.
- To map roles, configure a helper bean for the CAS authentication provider. The default implementation maps administrator roles to a static list of usernames, and all others have the minimal user roles.
- The synchronization is already handled by the `JIAuthenticationSynchronizer`.

The following sections explain how to make the configuration changes presented above.

3.3.1 Files to Modify

The configuration for external CAS authentication is made in the following files:

- $CATALINA_HOME/bin/catalina.sh or .bat, or the equivalent file for your application server.
- <application-server-path>/jasperserver[-pro]/WEB-INF/applicationContext-security.xml

Where:

- <application-server-path> is the location of the application server in which you deployed JasperServer.
- -pro is part of the default pathname if you installed JasperServer Professional.

The rest of this document refers to file names alone. Refer to the paths above to find the file in your deployment.

3.3.2 Beans to Configure

This section explains the beans that are involved in external CAS authentication and how they relate to each other. Acegi security provides all the beans to configure CAS authentication, except for the `UserDetailsServiceImpl` helper class that JasperServer has provided since the 2.0.1 release.

However, unlike the configuration of JasperServer for LDAP, the configuration beans for CAS are not included by default in comments in the configuration files. As a result, you will need to add the following beans to the configuration file and edit them for your particular settings. In many places you must find the existing configuration and replace it with the example.

The beans are listed here in the order they appear or should appear logically in the applicationContext-security.xml file.

- `filterChainProxy` - Name of a bean of class `FilterChainProxy` for defining the how page requests are processed and order of filters invoked on them.
- `authenticationManager` - Name of a bean of class `ProviderManager` for performing authentication, configured with:
  - `casAuthenticationProvider` - Name of a bean of class `CasAuthenticationProvider`, configured with:
    - `casDaoAuthorityPopulator` - Name of a bean of class `DaoCasAuthoritiesPopulator`, configured with:
      - `casUserAuthorityService` - Name of a bean of class `UserDetailsServiceImpl` that configures the mapping of roles to username for external CAS users.
    - `acceptAnyCasProxy` - Class of an anonymous bean that determines the response to proxy CAS servers.
    - `casTicketValidator` - Name of a bean of class `CasProxyTicketValidator` configured with:
      - The HTTPS URL on the CAS server to respond to service validation requests.
      - `authenticationServiceProperties` - Name of bean configured below.
    - `ehCacheBasedTicketCache` - Class of an anonymous bean that configures where to store tickets:
      - `ticketCache` - Name of a bean of class `EhCacheFactoryBean` configured with:
        - `ticketCacheManager` - Name of a bean of class `EhCacheManagerFactoryBean` that gives the location of the configuration file for the ticket cache.
        - A name for the ticket cache.
- `authenticationServiceProperties` - Name of a bean of class `ServiceProperties` that specifies the URL to which the CAS server should send the ticket to initiate service validation.
- `authenticationProcessingFilterEntryPoint` - Name of a bean of class `CasProcessingFilterEntryPoint` that specifies the URL of the CAS login page for redirection.
- `authenticationProcessingFilter` - Name of a bean of class `CasProcessingFilter` that implements a new authentication processing filter that only process tickets from the CAS server.

The remaining sections in this chapter give the configuration details for these beans. The sections are shown in the logical order of the beans in the file, not in the order of the steps in the protocol with which they are associated.

3.4 Configuring Java to Trust the CAS Certificate

The CAS protocol requires that the response to the service validation be established over HTTPS for security. This connection is established from the Java classes of Acegi Security, and therefore it is the Java security system that must be configured. Java security must trust the certificate it receives from the CAS server, otherwise it will refuse to connect. This trust is based on two factors:

1. The host name in the certificate has to match the host name in the URL of the connection. Certain JVMs require hostnames as opposed to IP addresses, even if the IP addresses match. See section **3.1, “CAS Server for Testing,” on page 26** for instructions to create a certificate in a keystore.
2. You must tell Java to trust the signing certificate:
   a. Copy the file you used as the keystore for the CAS server to the JasperServer host machine.
   b. On the JasperServer host machine, edit the file $CATALINA_HOME/bin/catalina.sh or .bat, or the equivalent file for your application server. In the section for setting JAVA_OPTS, add the following option:

      ```
      "-Djavax.net.ssl.trustStore=<keystore-path>"
      ```

3.5 Modifying the Acegi Filter Chain

In applicationContext-security.xml, locate the `filterChainProxy` bean and add a pattern for matching the URL that the CAS server will request during service validation. The pattern must match the one you define in the following properties in section **3.9, "Configuring the Filters for CAS," on page 32**:

- The `service` property in the `authenticationServiceProperties` bean.
- The `filterProcessesUrl` in the `authenticationProcessingFilter` bean.

Set the filter chain for this pattern as shown in the following example.

```
<bean id="filterChainProxy" class="org.acegisecurity.util.FilterChainProxy">
  <property name="filterInvocationDefinitionSource">
    <value>
      CONVERT_URL_TO_LOWERCASE_BEFORE_COMPARISON
      PATTERN_TYPE_APACHE_ANT

      /xmla=httpSessionContextIntegrationFilter,basicProcessingFilter,
      JIAuthenticationSynchronizer,anonymousProcessingFilter,
      basicAuthExceptionTranslationFilter,filterInvocationInterceptor

      /services/**=httpSessionContextIntegrationFilter,basicProcessingFilter,
      JIAuthenticationSynchronizer,anonymousProcessingFilter,
      basicAuthExceptionTranslationFilter,filterInvocationInterceptor

      /j_acegi_cas_security_check=httpSessionContextIntegrationFilter,
      authenticationProcessingFilter,anonymousProcessingFilter,
      exceptionTranslationFilter,filterInvocationInterceptor

      /**=httpSessionContextIntegrationFilter,userPreferencesFilter,
      authenticationProcessingFilter,basicProcessingFilter,JIAuthenticationSynchronizer,
      anonymousProcessingFilter,exceptionTranslationFilter,filterInvocationInterceptor
    </value>
  </property>
</bean>
```

3.6 Adding the casAuthenticationProvider

In applicationContext-security.xml, locate the `authenticationManager` bean and add `casAuthenticationProvider` to the top of the list of providers.

```
<bean id="authenticationManager" class="org.acegisecurity.providers.ProviderManager">
  <property name="providers">
    <list>
      <ref local="casAuthenticationProvider"/>
      <ref local="anonymousAuthenticationProvider"/>
    </list>
  </property>
</bean>
```

Unlike other forms of external authentication, CAS is not compatible with multiple authentication providers. You must remove all other providers from the list, leaving only the `anonymousAuthenticaionProvider` as shown above. As a consequence of this configuration, you must have users defined in CAS that will be your administrators, you cannot use the default `jasperadmin` user account. See section **3.8, "Mapping the User Roles," on page 32** for instructions on designating which CAS user will be mapped to the administrator roles.

3.7 Configuring the CAS Authentication Behavior

Below the `authenticationManager` in the applicationContext-security.xm file, add the `casAuthenticationProvider` and its helper beans as shown in the following example. In particular, the `casTicketValidator` bean is configured with the HTTPS URL where tickets are validated on the CAS server.

```
<bean id="casAuthenticationProvider"
      class="org.acegisecurity.providers.cas.CasAuthenticationProvider">
  <property name="casAuthoritiesPopulator">
    <ref local="casDaoAuthorityPopulator"/></property>
  <property name="casProxyDecider">
    <bean class="org.acegisecurity.providers.cas.proxy.AcceptAnyCasProxy"/></property>
  <property name="ticketValidator"><ref local="casTicketValidator"/></property>
  <property name="statelessTicketCache">
    <bean class="org.acegisecurity.providers.cas.cache.EhCacheBasedTicketCache">
      <property name="cache"><ref local="ticketCache"/></property></bean></property>
  <property name="key"><value>lam_or_lame</value></property>
</bean>

<bean id="ticketCacheManager"
      class="org.springframework.cache.ehcache.EhCacheManagerFactoryBean">
  <property name="configLocation">
    <value>classpath:/ehcache-failsafe.xml</value></property>
</bean>

<bean id="ticketCache" class="org.springframework.cache.ehcache.EhCacheFactoryBean">
  <property name="cacheManager"><ref local="ticketCacheManager"/></property>
  <property name="cacheName"><value>casTicketCache</value></property>
</bean>

<bean id="casTicketValidator"
      class="org.acegisecurity.providers.cas.ticketvalidator.CasProxyTicketValidator">
  <property name="casValidate">
    <value>https://hostname1/cas/proxyValidate</value></property>
  <property name="serviceProperties">
    <ref local="authenticationServiceProperties"/></property>
</bean>
```

The helper beans for the ticket cache use default settings provided by Acegi Security and do not need to be further modified.

3.8 Mapping the User Roles

Among the helper beans for the `casAuthenticationProvider` in the applicationContext-security.xm file, add the `casDaoAuthorityPopulator` and the `casUserAuthorityService` beans that map roles to users.

```
<bean id="casDaoAuthorityPopulator"
      class="org.acegisecurity.providers.cas.populator.DaoCasAuthoritiesPopulator">
  <property name="userDetailsService"><ref local="casUserAuthorityService"/></property>
</bean>

<bean id="casUserAuthorityService"
class="com.jaspersoft.jasperserver.api.metadata.user.service.impl.UserDetailsServiceImpl">
  <property name="adminUsers">
    <list><value>tomcat</value></list></property>
  <property name="defaultAdminRoles">
    <list><value>ROLE_USER</value>
          <value>ROLE_ADMINISTRATOR</value></list></property>
  <property name="defaultInternalRoles">
    <list><value>ROLE_USER</value></list></property>
</bean>
```

This simple implementation of the `userDetailsServiceImpl` class is configured with the following:

- `adminUsers` - A list of usernames that will be granted administrator privileges in JasperServer. The username values must match exactly the usernames authenticated and returned by the CAS server.
- `defaultAdminRoles` - A list of JasperServer role names that will be assigned to every user in the list of administrators.
- `defaultInternalRoles` - A list of JasperServer role names that will be assigned to every user not in the list of administrators.

3.9 Configuring the Filters for CAS

Finally, we need to replace the `authenticationProcessingFilter` and the `authenticationProcessingFilterEntryPoint` for CAS. Notice that we keep the same bean names in the `id` attribute and substitute a different class with its specific properties. Because these beans are referenced by name in the beans that call them, the new classes will automatically be used instead. Alternatively, we could give the new beans new names, but then we would also need to re-configure the beans that call them. The documentation for Acegi Security takes the latter approach.

Locate and replace the existing entries in the applicationContext-security.xm file with the following beans:

```
<bean id="authenticationServiceProperties"
      class="org.acegisecurity.ui.cas.ServiceProperties">
  <property name="service">
    <value>http://hostname2/jasperserver/j_acegi_cas_security_check</value></property>
  <property name="sendRenew"><value>false</value></property>
</bean>

<bean id="authenticationProcessingFilterEntryPoint"
      class="org.acegisecurity.ui.cas.CasProcessingFilterEntryPoint">
  <property name="loginUrl"><value>http://hostname1/cas/login</value></property>
  <property name="serviceProperties"><ref local="authenticationServiceProperties"/>
  </property>
</bean>

<bean id="authenticationProcessingFilter"
      class="org.acegisecurity.ui.cas.CasProcessingFilter">
  <property name="authenticationManager"><ref local="authenticationManager"/></property>
  <property name="authenticationFailureUrl"><value>/loginerror.html</value></property>
  <property name="defaultTargetUrl"><value>/loginsuccess.html</value></property>
  <property name="filterProcessesUrl"><value>/j_acegi_cas_security_check</value></property>
</bean>
```

The `service` property of the `authenticationServiceProperties` bean specifies the JasperServer URL that will receive the ticket when the CAS server performs service validation. The `loginUrl` property of the `authenticationProcessingFilterEntryPoint` bean specifies the login URL for the CAS server, the one to which JasperServer will redirect all users for authentication.

3.10 Restarting JasperServer

When you have configured all the beans in the appropriate files, restart JasperServer. Because of the modification to the application server configuration in **3.4, “Configuring Java to Trust the CAS Certificate,” on page 29**, you need to restart the application server, which will also restart JasperServer.

Instead of navigating to the JasperServer login page, go directly to another page in JasperServer; your home page, for example:

http://hostname2:8080/jasperserver/flow.html?_flowId=homeFlow

If JasperServer, CAS, and your certificates are configured correctly, you should be prompted to log into the CAS server, and when you do so, you should be redirected to the JasperServer home page for the user you logged in as.

3.11 Detailed Protocol Exchange

This section gives a more detailed view of the protocol exchange between the Acegi components of JasperServer and the CAS server. It shows which beans respond to the various communications and how they exchange information to authenticate a user.

This section is based on the Acegi Security documentation. For links to the source, see section **1.4, “Acegi Security Framework,” on page 7**.

1. The user requests a page that is either secure or that includes secure beans. Acegi Security's `ExceptionTranslationFilter` detects an `AuthenticationException`. Because the user's Authentication object (or lack thereof) caused an `AuthenticationException`, the `ExceptionTranslationFilter` calls the configured `AuthenticationEntryPoint`. If using CAS, this is the `CasProcessingFilterEntryPoint` class.

2. The `CasProcessingFilterEntry` point redirects the browser to the CAS server and specifies a service parameter, which is the callback URL for Acegi Security service. For example, the browser could be redirected to http://hostname1/cas/login?service=https%3A%2F%2Fhostname2%2Fjasperserver%2Fj_acegi_cas_security_check. Both URLs are configurable.

3. After the browser redirects to CAS, the user is prompted for their username and password, unless they have a session cookie that indicates they've logged in previously. In this case, they are not prompted to login again (with a single exception, which is discussed later). CAS uses the `PasswordHandler` (or `AuthenticationHandler` if using CAS 3.0) to decide whether the username and password is valid.

4. If login is successful, CAS redirects the browser to the callback URL as requested. It also includes a ticket parameter, which is an opaque string representing the "service ticket." For example, the browser might be redirected to: http://hostname2/jasperserver/j_acegi_cas_security_check?ticket=ST-0-ER94xMJmn6pha35CQRoZ

5. In the web application, the `CasProcessingFilter` always listens for requests to /j_acegi_cas_security_check. The processing filter constructs a `UsernamePasswordAuthenticationToken` representing the service ticket. The principal is equal to `CasProcessingFilter.CAS_STATEFUL_IDENTIFIER`, and the credentials are the opaque "service ticket" string. The authentication request is sent to the configured `AuthenticationManager`.

6. The `AuthenticationManager` implementation is the `ProviderManager`, which in turn is configured with the `CasAuthenticationProvider`. The latter only responds to `UsernamePasswordAuthenticationTokens` containing the CAS-specific principal (such as `CasProcessingFilter.CAS_STATEFUL_IDENTIFIER`) and `CasAuthenticationTokens` (discussed below).

7. `CasAuthenticationProvider` validates the service ticket using a `TicketValidator` implementation. In Acegi, this is implemented as `CasProxyTicketValidator`. This implements a ticket validation class included in the CAS client library. The `CasProxyTicketValidator` sends a service ticket validation HTTPS request to the CAS server. The `CasProxyTicketValidator` may also include a proxy callback URL. For example: https://hostname1/cas/proxyValidate?service=https%3A%2F%2Fhostname2%2Fjasperserver%2F j_acegi_cas_security_check&ticket=ST-0-ER94xMJmn6pha35CQRoZ&pgtUrl=https://hostname2/ jasperserver/casProxy/receptor

8. When the CAS Server receives the proxy validation request, it checks to see if the service ticket matches the service URL to which it was issued. If they match, CAS returns an affirmative XML response that specifies the username. If any proxy was involved in the authentication (discussed below), the list of proxies is also included in the XML response.

9. Optionally, when the request sent to the CAS validation service includes the proxy callback URL in the `pgtUrl` parameter, CAS includes a `pgtIou` string in the XML response. This string represents a proxy-granting ticket IOU. The CAS server then creates its own HTTPS connection to `pgtUrl`. This authenticates both the CAS server and the claimed service URL. The systems uses the HTTPS connection to send a proxy-granting ticket to the original web application. For example: https://hostname2/jasperserver/casProxy/receptor?pgtIou=PGTIOU-0-R0zlgrl4pdAQwBvJWO3vnNpevwqStbSGcq3v &pgtId=PGT-1-si9YkkHLrtACBo64rmsi3v2nf7cpCResXg5MpES.

   If you require proxy-granting tickets, use CAS's `ProxyTicketReceptor` servlet to receive them.

10. The `CasProxyTicketValidator` parses the XML it received from the CAS server. It returns a `TicketResponse` to the `CasAuthenticationProvider`, including the username (mandatory), proxy list (if any were involved), and proxy-granting ticket IOU (if the proxy callback was requested).

11. Next, `CasAuthenticationProvider` calls a configured `CasProxyDecider`, which indicates whether the proxy list in the `TicketResponse` is acceptable. Several implementations are provided with Acegi Security System: `RejectProxyTickets`, `AcceptAnyCasProxy`, and `NamedCasProxyDecider`. The first two implementations are self-explanatory, while `NamedCasProxyDecider` can be configured with a list of trusted proxies.

12. `CasAuthenticationProvider` requests a `CasAuthoritiesPopulator` to advise the `GrantedAuthority` objects that apply to the user contained in the `TicketResponse`. Acegi Security includes a `DaoCasAuthoritiesPopulator` that uses the `UserDetailsService` infrastructure to find the `UserDetails` and the associated `GrantedAuthority` objects. Note that the system ignores the password and enabled/disabled status of `UserDetails` returned by the `UserDetailsService`, as the CAS server is responsible for authentication decisions. `DaoCasAuthoritiesPopulator` only retrieves the `GrantedAuthority` objects.

13. If no problems were encountered, `CasAuthenticationProvider` constructs a `CasAuthenticationToken`, including the details contained in the `TicketResponse` and the `GrantedAuthority` objects. The token contains the hash of a key, so that the `CasAuthenticationProvider` knows it created the response.
14. Control is returned to `CasProcessingFilter`, which places the created `CasAuthenticationToken` into the `HttpSession` attribute named `HttpSessionIntegrationFilter.ACEGI_SECURITY_AUTHENTICATION_KEY`.
15. The browser is redirected to the original page that caused the `AuthenticationException`.
16. Since the `Authentication` object is now in the well-known location, it is handled like any other authentication approach. Usually the `HttpSessionIntegrationFilter` is used to associate the `Authentication` object with the `SecurityContextHolder` for the duration of each request.