Blackboard Academic Suite™

Authentication Manual

Release 6.1–6.3
Blackboard Learning System™
Blackboard Community System™

Publication Date: February 9, 2004
Date of last revision: June 28, 2005

Worldwide Headquarters
Blackboard Inc.
1899 L Street, NW, 5th Floor
Washington, DC 20036-3861 USA
800-424-9299 toll free US & Canada
+1-202-463-4860 telephone
+1-202-463-4863 facsimile
www.blackboard.com

International Headquarters
Blackboard International B.V.
Keizersgracht 106
1015 CS Amsterdam
The Netherlands
+31 20 5206884 (NL) telephone
+31 20 5206885 (NL) facsimile
global.blackboard.com

Table of Contents

Introduction

Overview

The *Blackboard Academic Suite Authentication Manual* provides background information, specific instructions, and resources for using the different types of end-user authentication with *Blackboard Academic Suite™* (Release 6.1) and higher.

The *Blackboard Learning System* controls authentication for all of the system in the *Blackboard Academic Suite™*. Thus, authenticating against the *Blackboard Learning System* will also identify users to the *Blackboard Community system* and *Blackboard Content System*, if licensed.

End-user Authentication connects the *Blackboard Academic Suite* to institutional systems that manage user login information. By doing so, users can login to the *Blackboard Academic Suite* with the same login that they use to access other institutional systems. This cuts down on the maintenance and risk of managing user authentication information.

Manual Updates

Please note that the *Blackboard Academic Suite Authentication Manual* is updated periodically. Check the Date Last Update at the beginning of the manual to ensure that it is the most recent copy. Any updates are listed in the Appendix.

To report any comments or suggestions regarding this manual, please contact Blackboard Support.

Authentication Overview

In the *Blackboard Learning System* authentication refers to identifying a user by their login credentials, such as the user name and password. These credentials are submitted to the *Blackboard Learning System* for user identification. *Blackboard Learning System* (Release 6.1) and higher includes three authentication modules that will validate Blackboard users against an institution's existing authentication model:

- use the default authentication that comes with the *Blackboard Learning System*
- use Blackboard's implementation for a supported authentication type (LDAP, Passport, or Web server delegation)
- create a custom implementation for a supported authentication type

Authentication Types

The table below describes the authentication modules available in *Blackboard Academic Suite* (Release 6.1) and higher:

Authentication Type	Description
Blackboard's authentication	*Blackboard Learning System's* default authentication authorizes the user against the login credentials (for example, username and password information) stored in the *Blackboard Learning System* database.

LDAP authentication	LDAP is an Internet standard that provides access to information through a set of protocols that accesses information directories and retrieves information. For instructions on configuring LDAP see the section LDAP Configuration.
Web Server delegation	All of the work related to authentication is managed by the Web server. This includes obtaining the user's credentials and verifying the credentials. An example of Web server delegation is Active Directory®.
Passport authentication	The Microsoft®.NET Passport service allows users to use a single name and password for all participating Web sites. For information on configuring the properties files see the Passport Properties section.

All the files necessary to support authentication through LDAP or Active Directory® are included with *Blackboard Academic Suite*. To install Passport, Administrators must download an encryption key and the Passport 1.4.2 SDK from the Passport website.

Institutions may create a customized implementation for any of the above authentication types. See Custom Authentication for more information.

Set Authentication Type

To configure the system for one of the authentication types the `bbconfig.auth.type` property must be set by editing the `bb-config.properties` file. This allows the *Blackboard Learning System* at start up to select the appropriate set of authentication configuration settings. After this is completed, the `authentication.properties` file may be edited to change the default settings for the authentication type.

The following sections include details on configuring the different types of authentication. Each section includes a Set Authentication Settings topic that explains how the properties are edited.

Introduction to *Blackboard Academic Suite* Authentication

Overview

The default authentication for *Blackboard Academic Suite* authenticates the user's login credentials against the *Blackboard Learning System* database.

Customize the default authentication

To customize the default authentication for the *Blackboard Learning System* the Administrator modifies the `authentication.properties` file. Tables in the *Blackboard Learning System* database do not need to be changed.

Return to the default authentication

To return to the *Blackboard Learning System* default authentication (rdbms), the authentication type (`bbconfig.auth.type`) must be set via the command line. This allows the *Blackboard Learning System*, at start up, to select the appropriate set of `auth.type*.*` entries. The following are instructions for setting the default authentication:

Step 1 Edit the `authentication.properties` file

- change to the following directory

```
cd BB6_DEPLOY_DIR\blackboard\config
```

- see Authentication Properties for details

Step 2 Edit the `bb-config.properties` file

- change the property to `bbconfig.auth.type=rdbms`

Step 3 Deploy configuration updates

```
cd BB6_DEPLOY_DIR\tools\admin
PushConfigUpdates.bat
```

Note: `PushConfigUpdate` will over write any customizations to the configuration of the application.

Authentication properties

The table below details the properties applicable to the default authentication model. These properties are configured through the `authentication.properties` file. The `authentication.properties` properties file is found in the directory BB6_DIR/bbservices/config, where BB6_DIR is the directory that the *Blackboard Learning System* is installed in.

Property	Description
`auth.type.rdbms.impl`	Defines the class which must conform to the HttpAuthModule interface. The default value, `blackboard.platform.security.authentication.BaseAuthenticationModule`, should not be changed unless the institution builds and implements its own class for *Blackboard Learning System* default authentication.

`auth.type.rdbms.use_challenge`	Defines the encryption setting where a value of “false” indicates the password is encrypted with base 64 and a value of “true” indicates the password is encrypted with MD5. The default value is “true”. MD5 encryption offers stronger security for passwords. Base64 is similar to sending the password in plain text.

Example

Below is an example of how the `authentication.properties` file is configured for *Blackboard Learning System* default authentication.

```
auth.type.rdbms.impl=blackboard.platform.security.authentication.BaseAuthenticationMod
ule
auth.type.rdbms.use_challenge=true
```

LDAP Authentication

Overview

Blackboard Learning System includes an LDAP (Lightweight Directory Access Protocol) module that will authenticate users against an institution's directory server or servers using LDAP. All the files necessary to support LDAP authentication are included with the *Blackboard Learning System*.

LDAP authentication

LDAP is an Internet standard that provides access to information from different computer systems and applications. LDAP uses a set of protocols to access information directories and retrieve information. A directory is like a database, but contains information that is more descriptive and attribute-based. Information in a directory is generally read more often then it is written or modified. LDAP allows an application, running on the institution's computer platform, to obtain information such as user names and passwords.

Centralizing this type of information is very beneficial. It simplifies the job of the System Administrator by providing a single point of administration. It also provides a single location for user information, reducing the storage of duplicate information. This, in turn, reduces maintenance needs. LDAP authentication also enables users to have a single login and password to access a number of different applications.

LDAP topics

This section includes the following topics:

LDAP Module

Overview

Standard LDAP authentication is fully integrated with *Blackboard Learning System*. All necessary .jar files, including those for setting up an SSL connection between the *Blackboard Learning System* application server and the directory servers, are provided in the /systemlib directory. The application server startup executables include the .jar files in their classpath. Note that all configuration options in the `authentication.properties` file are set to default values. Some of these default values are place holders and must be changed by the Administrator for LDAP authentication to work successfully.

To begin authenticating against an LDAP server or servers, set the properties found in the `authentication.properties` file. The SSL Configuration topic has specific information on enabling the *Blackboard Learning System* application server and the directory servers to communicate over SSL.

Limitations

The limitations of this version of the LDAP module are summarized in the following list.

- The module only supports authentication through a successful bind with the directory server using the FDN for this Blackboard user—the module cannot retrieve any information from the directory.
- The module only supports binding anonymously or binding with a privileged user and then performing a search for the user's FDN.

Check with Blackboard Technical Support if you have any questions regarding these limitations. For installation problems or questions using this document contact Blackboard Technical Support at www.behind.blackboard.com. For planning, architectural analysis, best practices, or assistance with implementation call Blackboard Technical Solutions.

LDAP Configuration Overview

Overview

This topic provides an overview of the LDAP Installation process. This process consists of a set of steps that will enable the System Administrator to use LDAP authentication.

LDAP configuration

The following steps outline the LDAP configuration process:

Step 1 Edit the `authentication.properties` file
- located at `cd BB6_DEPLOY_DIR\blackboard\config`
- see the LDAP Property Configuration section for more detail

Step 2 Configure the `bbconfig.auth.type` property to LDAP. This must be done for the configuration to proceed correctly. Make the following change in the `bb-config.properties` file: `bbconfig.auth.type=ldap`

Step 3 Deploy the configuration updates

```
cd BB6_DEPLOY_DIR\tools\admin
PushConfigUpdates.bat
```

Note: `PushConfigUpdate` will overwrite any customizations to the configuration of the application.

Open LDAP – UNIX Operating Systems only

Blackboard has two versions of LDAP client authentication modules, the default and OpenLDAP.

Two modules exist because the default LDAP client does not release file descriptors when it is under heavy load. A file descriptor is used by UNIX Operating Systems to keep track of open files and network connections. If the system continually accumulates file descriptors, the server will reach a maximum number of allowed file descriptors, at which point no more files can be opened and no more network connections can be accepted.

Administrators of UNIX Operating Systems who experience this file descriptor issue under heavy load may deploy OpenLDAP as a workaround. If OpenLDAP is used, the .jar files must be updated so the command line tools do not fail. A copy of the jar file should be in /systemlib. Additionally, edit /system/build/bin/launch-tool.sh and append the .jar files to the BB_CP variable. Otherwise, command line tools that bootstrap the core services (for example, LogRotation or PurgeAccumulator) will not work.

LDAP Properties

Overview

The properties set in the `authentication.properties` file include general properties for LDAP configuration, as well as properties for individual directory servers. When adding multiple servers the variable *x* represents the sequence number. Parameters must be set for each directory server that the *Blackboard Learning System* will authenticate against. The LDAP module will access the servers according to the sequence number.

File format

The `authentication.properties` and `bb-config.properties` files contain a series of properties that must be set before authentication against the institution's directory server or servers can occur. Each property is listed with an equal sign followed by the corresponding value.

Editing the properties file

Open the `authentication.properties` file in an editor and set the LDAP specific properties to match the institution. Descriptions of the properties appear in the following section.

Properties that are suffixed with a number are properties that are associated with an individual directory server. To add information for additional directory servers, add a group of properties suffixed with the next available sequence number. The LDAP module will access the servers in the order in which they are sequenced.

Debugging LDAP Authentication

Administrators may debug the LDAP authentication as part of troubleshooting. The steps below explain how to debug LDAP authentication:

Step 1 Modify `/blackboard.config/service-config.properties`. Under Logging Service in the `service-config.properties` file set:

```
blackboard.service.log.param.logdef.default.verbosity=debug
```

Step 2 Restart the services.

```
/blackboard/tools/admin/ServiceController services.stop
/blackboard/tools/admin/ServiceController services.stop
```

Step 3 Login to the system again.

Step 4 Search `/blackboard/logs/bb-services-log.txt` for references to "LDAPAuthModule".

Windows users- open the log file in a text editor and search for "LDAPAuthModule".

UNIX users – execute the following:

```
tail -f -n200
/blackboard/config/service-config.properties | grep
"LDAPAuthModule"
```


LDAP Property Configuration

The table below details the LDAP properties configured through the `authentication.properties` file.

Property	Description
`auth.type.ldap.impl`	Defines the class which must conform to the HttpAuthModule interface. The default value, `blackboard.platform.security.authentication.LDAPAuthModule`, should not be changed unless the institution builds and implements its own class for LDAP authorization.
`auth.type.ldap.use_challenge`	Defines the encryption setting where a value of “false” indicates base 64 encryption and a value of “true” indicates MD5 encryption. The default value is “false”. MD5 encryption should only be used if the LDAP servers use MD5 encryption in the same manner as Blackboard. In most cases, using base 64 encryption and securing the connection between *Blackboard Learning System* and the LDAP servers with SSL is the best approach.

Property	Description
auth.type.ldap.num_servers	Defines the number of directory servers in use. For each server, there must be a corresponding set of server properties. This property must be kept current; update it each time a new server's entries are added to the authentication.properties file.
auth.type.ldap.user_not_found_fallback	Can be set to "true" or "false". By default, this property is set to "false" due to the security considerations outlined in LDAP Security Considerations. If set to "true" the module will attempt to authenticate the user using the password in the *Blackboard Learning System* database if the user is not found in any of the directory servers.
auth.type.ldap.error_fallback_to_bb	Can be set to "'true" or "false". By default, this property is set to "false" due to the security considerations outlined in LDAP Security Considerations. If set to "true" the module will attempt to authenticate the user using the password in the *Blackboard Learning System* database if there is an error connecting to any of the directory servers.

Property	Description
Server Specific Properties	
auth.type.ldap.server_url.x	The URL of the directory server including port. Example: ldap://directory.university.edu:389 If the LDAP server is setup to communicate over SSL, then the URL should be: ldaps://directory.university.edu:636
auth.type.ldap.server_ssl.x	Must be set to "true" or "false". If set to "true" the module will attempt to connect to the LDAP directory using SSL. The LDAP server must be set up to handle SSL connections. **Note**: See the SSL Configuration section for more information.
auth.type.ldap.use_priv_user.x	Must be set to "true" or "false". If set to "true" the module will bind to the LDAP server as a privileged (specific) user when searching for the FDN of the user to authenticate.
auth.type.ldap.user_fdn.x	The user binds as this FDN. Leave as "(none)" if not applicable.
auth.type.ldap.user_pwd.x	The password of the privileged user. Leave as "(none)" if not applicable.
auth.type.ldap.deref_aliases.x	Set this property to configure how

	aliases are dereferenced during search operations. The following values are defined for this property: • **always:** Always dereference aliases. • **never:** Never dereference aliases. • **finding:** Dereference aliases only during name resolution (that is, while locating the target entry). • **searching:** Dereference aliases once name resolution has been completed (that is, after locating the target entry).
`auth.type.ldap.user_tag.x`	Set this property to the attribute containing the *Blackboard Learning System* User Name.
`auth.type.ldap.server_error_fatal.x`	Must be set to "true" or "false". If set to "true" the module will exit with a fatal error if there is an error connecting to the server.
`auth.type.ldap.referral.x`	The value of this property is a string that specifies how referrals should be handled by the module. The following values are defined for this property: • **Follow:** Automatically follow any referrals. • **Throw:** Throw a Java ReferralException for each referral. **Note:** This will result in an error condition for this server. • **Ignore:** Ignore referrals if they appear in results. In debug mode, a log message will be generated to indicate an incomplete result, but this will not result in an error condition for this server.
`auth.type.ldap.referral_limit.x`	The value of this property is a string of decimal digits specifying the maximum number of referrals to follow in a chain of referrals. A setting of zero indicates that there is no limit.
`base_search_fdn`	The starting point in the LDAP directory structure for searching for a *Blackboard Learning System* user.

Example

Below is an example of the LDAP properties configured through the `authentication.properties` file.

```
auth.type.ldap.impl=blackboard.platform.security.authentication.LDAPAuthModule
auth.type.ldap.use_challenge=false

auth.type.ldap.error_fallback_to_bb=false
```

```
auth.type.ldap.user_not_found_fallback_to_bb=false
auth.type.ldap.log_level=error
```

*Acceptable entries for log_level are: fatal, error, warning, information, debug

```
auth.type.ldap.num_servers=2
```

*This value must be updated for every server configuration that is added below

Server #1 Configuration

```
auth.type.ldap.server_ssl.1=false
```

* This variable indicates whether interaction between Blackboard installation server and LDAP server should be over SSL

```
auth.type.ldap.base_search_fdn.1=dc=dc,dc=blackboard,dc=com
auth.type.ldap.deref_aliases.1=never
auth.type.ldap.server_url.1=ldap://lsvr1
auth.type.ldap.use_priv_user.1=true
auth.type.ldap.user_fdn.1=uid=UserA,ou=Special Users,dc=dc,dc=blackboard,dc=com
auth.type.ldap.user_pwd.1=test1
auth.type.ldap.user_tag.1=uid
auth.type.ldap.referral.1=ignore
auth.type.ldap.referral_limit.1=0
auth.type.ldap.server_error_fatal.1=true
```

Server #2 Configuration

```
auth.type.ldap.server_ssl.2=false
```

* This variable indicates whether interaction between Blackboard installation server and LDAP server should be over SSL

```
auth.type.ldap.base_search_fdn.2=dc=dc,dc=blackboard,dc=com
auth.type.ldap.deref_aliases.2=never
auth.type.ldap.server_url.2=ldap://lsvr2
auth.type.ldap.use_priv_user.2=true
auth.type.ldap.user_fdn.2=uid=UserB,ou=Special Users,dc=dc,dc=blackboard,dc=com
auth.type.ldap.user_pwd.2=test2
auth.type.ldap.user_tag.2=uid
auth.type.ldap.referral.2=ignore
auth.type.ldap.referral_limit.2=0
auth.type.ldap.server_error_fatal.2=true
```

Configuring LDAP Authentication with SSL

This section explains how to configure the `authentication.properties` file settings that enable the *Blackboard Learning System*, using LDAP authentication, to communicate with an LDAP Server over SSL. No extra entries need to be added to the `authentication.properties` file; the Administrator simply needs to set the appropriate properties correctly (see table below).

Note: The SSL Choice option in the System Control Panel is used to secure communication between the *Blackboard Learning System* and the client machine. This document does not provide information on this function. For more information, see the *Blackboard Learning System Administrator Manual*.

Property	Description
`auth.type.ldap.server_url.x`	When the LDAP server is setup to communicate over SSL, this property should be: ldap://directory.university.edu Administrators may need to append the port number depending upon the configuration.
`auth.type.ldap.server_ssl.x`	Must be set to “true” or “false”. If set to

	"true" the module will attempt to connect to the LDAP directory using SSL. The LDAP server must be set up to handle SSL connections.

Run `PushConfigUpdates` after editing the properties file. Finally, copy the JSSE, JNET, and JCERT files from `apps/tomcat/shared/lib` and paste these files into the `$JAVA_HOME/jre/lib/ext` directory.

Configuring LDAP Authentication with SSL for the JAVA runtime environment (JRE)

Copy the following three files to the JAVA_HOME\jre\lib\ext directory:

```
\blackboard\systemlib\jcert-1.0.2.jar
\blackboard\systemlib\jnet-1.0.2.jar
\blackboard\systemlib\jsse-1.0.2.jar
```

All Administrators should add the following to the JAVA_HOME /jre/lib/security/java.security file:

```
security.provider.1=sun.security.provider.Sun
security.provider.2=com.sun.net.ssl.internal.ssl.Provider
```

If there are already security providers listed, and the first one is "`sun.security.provider.Sun`", then a "`security.provider.X`" entry should be added to the end of the list.

The signed public SSL certificate must also be imported. Administrators configuring a fresh install of *Blackboard Learning System* should import a certificate for each LDAP server to the applications server's repository of trusted certificates. This is done through the 'keytool' utility.

Sun's instructions for this on Windows are found at: http://java.sun.com/j2se/1.5.0/docs/tooldocs/windows/keytool.html.

Sun's instructions for this on Unix are found at: http://java.sun.com/j2se/1.5.0/docs/tooldocs/solaris/keytool.html.

Note: The certificate must be imported whenever the J2SE version is updated.

Configuring Contextual Error Messages for LDAP

The default error message does not report the cause of the error to the user when LDAP Authentication fails. Use the auth.type.ldap.context_factory property to display contextual error messages to users.
For each LDAP server configured, set the auth.type.ldap.context_factory property. This property references a class to use for creating contexts, which need to be server specific. The options are:

`blackboard.platform.security.authentication.ResponseControlContextFactory`

Any server that supports the Netscape password policy response controls spec - any breed of Netscape Directory Server including RedHat, SunONE, OpenLDAP, and others.

`blackboard.platform.security.authentication.PasswordPolicyContextFactory`

Any server that supports the IETF standard password policy attributes (passwordExpirationTime) but not response controls, for example, Novell Directory Server.

The PasswordPolicyContextFactory is used by default. If the PasswordPolicyContextFactory is used the authentication module must be configured to use a valid privileged user (one that can search and access the passwordExpirationTime attribute of any user) in order for the proper error message to be shown.

Troubleshooting LDAP

Overview

The LDAP module should function with minimal maintenance if the `authentication.properties` file is configured properly. This topic includes information on how to troubleshoot configuring the properties files and on maintenance for LDAP authentication.

Troubleshooting LDAP Authentication Properties for Windows

For Administrators using a Windows workstation, the LDP executable may be used to troubleshoot LDAP authentication properties. The LDP executable, found on the Windows 2000 Server CD in the \SUPPORT\TOOLS folder, is used to search for specific data against the Active Directory and includes a graphical user interface. For users not using Active Directory, this tool may be used in the same way against other LDAP servers.

The following steps explain how to use the LDP Tool:

Step 5 Go to the **Connection** menu, uncheck the NTLM/Kerberos check box, and select **Bind**.

Step 6 Enter the LDAP privileged user DN in the User: field and the LDAP password in the Password: field.

Step 7 Locate the 'defaultNamingContext' attribute.

Step 8 Go to the **View** menu and select **Tree**.

Step 9 Enter the defaultNamingContext attribute value into the BaseDN: field and click **OK**.

Step 10 Locate the container for user records (by default, the DN for this container starts with 'CN=Users'; however the user records may be located elsewhere; try to locate the DN that contains all faculty and student user records)

Step 11 Record the DN that contains all faculty and student user records

Step 12 Double-click on the tree view of this container to see all user records.

Step 13 Go to the **Options** menu and select **Search**.

Step 14 Customize fields in the user records returned from the search. (This step may not be necessary)

Step 15 With the container selected, go to the **Browse** menu and select **Search**.

Step 16 Enter the user field to search by. The user field is the user tag property, for example, (CN=jsmith).

Step 17 Record the "distinguishedName" attribute for this user record.

Step 18 Verify that you can find a sample user. Enter the baseDN from Step 7 and (user_tag=someUserValue) where user_tag is the name of the LDAP

user record field that the client expects users to enter in the Blackboard login form. (For example, if the client expects users to login by entering their email address as the 'username' in the Blackboard login form, then the user_tag should be the name of the field that stores the user's email address).

Next, Administrators must update `authentication.properties`:

- Set `auth.type.ldap.base_search_fdn.1` to the DN for the container for user records (See Step 7 above).
- Set `auth.type.ldap.user_fdn.1` to the distinguishedName attribute value for the LDAP user (See Step 13 above).

Windows Operating System only:

- Set `auth.type.ldap.user_tag.1` to 'sAMAccountName' if the client wants users to login to Blackboard using a Windows username. 'sAMAccountName' is the name of the Active Directory user record field that stores the Windows username.

Troubleshooting LDAP Authentication Properties for UNIX

For Administrators using a UNIX workstation, the LDAP Browser may be used to troubleshoot LDAP authentication properties. This tool may be found at http://www.iit.edu/~gawojar/ldap/. The following steps explain how to use the tool:

Step 1 Open the LDAP browser.

Step 2 Click **File Menu** and select **Connect.**

Step 3 Enter the LDAP server hostname in the Host: field.

Step 4 Enter the port number that the LDAP server is listening on in the Port: field.

Step 5 Enter the base search DN in the Base DN: field. If a privileged bind is required, uncheck **Anonymous bind.**

Step 6 Enter the privileged user DN in the User DN: field. If **Append base DN** is checked, the Administrator only needs to add the relative DN (for example, if the base DN is "OU=test users,dc=blackboard,dc=com" and the privileged user's full DN is "CN=privldap,OU=ldap testers,OU=test users,dc=blackboard,dc=com", then only enter "CN=privldap,OU=ldap testers").

Step 7 Click **Connect.**

Step 8 Click **Search** to search for a given user DN, or scroll through the list.

Revert to default authentication

To revert to the default authentication from LDAP, change `bbconfig.auth.type` to "rdbms", and restart the *Blackboard Learning System* application server. For more information, see the Return to Default authentication topic.

Blackboard application log

The *Blackboard Learning System* log records all application events handled by the Java API. Within the log the Blackboard LDAP module writes error, warning, informational, and debug messages to the `bb-services-log.txt` file.

Common problems

The table below outlines some of the common problems that may occur when authenticating *Blackboard Learning System* users against LDAP servers.

Problem	Action
The LDAP module loads but users cannot log in using their LDAP passwords.	Ensure that all of the users logging in have a *Blackboard Learning System* User Name. The *Blackboard Learning System* needs a user record to associate course and other information to the user.
An error is posted to the `bb-services-log.txt` whenever a user tries to log into the system. The module is configured to use SSL.	Ensure that the server certificate for your LDAP directory has been imported into the keystore of the JVM on the *Blackboard Learning System* application server. The JVM needs this certificate to allow SSL connections to the LDAP directory.
The LDAP module loads, but users cannot log in. Nothing is displayed in the logs, or the messages that are displayed are insufficient to diagnose the problem.	Re-run the `auth-type.properties` file and specify a log_level of “debug”. Log messages will generate with more detail.

LDAP scenarios

The table below details the systems response to a number of potential LDAP situations. The default configuration of LDAP will support the set of behaviors described here.

If . . .	then . . .
the LDAP server is down	authentication should fail with an appropriate message.
the user exists in Blackboard but not in LDAP	authentication should fail with an appropriate message.
the user exists in LDAP but not in Blackboard	authentication should fail with an appropriate message.
the privileged user doesn't exist or has expired	authentication should fail with an appropriate message. The *Blackboard Learning System* configuration file must be updated to proceed.
the privileged user password has changed	authentication should fail with an appropriate message. The *Blackboard Learning System* configuration file must be updated to proceed.
there are multiple LDAP accounts for a specific user	the search domain would be restricted to a specific context within the directory tree. The first account returned will be the one used. It is the institution's responsibility to set the `base_search_fdn` property correctly to avoid this situation.

the LDAP SSL certificate expires	authentication should fail with an appropriate message. The LDAP SSL certificate must be updated to proceed.

Troubleshooting LDAP with SSL

This section explains how to troubleshoot the SSL connection between the Blackboard server and the LDAP server for clients who are using an SSL connection to secure their LDAP server.

Instructions for debugging and clean up on UNIX:

Step 1 Save /usr/local/blackboard/apps/tomcat/bin/tomcat.sh as tomcat.sh.prod

Step 2 Enter the following command: `cp tomcat.sh.prod tomcat.sh.debug`

Step 3 Insert `-Djavax.net.debug=all,record,plaintext` into tomcat.sh.debug

Go to line 207 of tomcat.sh.debug

Edit this line to read:

```
$JAVACMD -Djavax.net.debug=all,record,plaintext $TOMCAT_OPTS
$JAVA_OPTS $MAIN start $@ \
```

Step 4 Enter the following command: `cp tomcat.sh.debug tomcat.sh`

Step 5 Restart services

Step 6 Login with the LDAP username and password

Step 7 Copy the SSL-connection trace information from /usr/local/blackboard/logs/tomcat-jvm-stdout.txt. See the log file example below.

Step 8 Repeat Steps 6 and 7 until debugging is complete

Step 9 Enter the following command: `cp tomcat.sh.prod tomcat.sh`

Step 10 Restart services

Instructions for debugging and clean up on Windows:

Step 1 Save D:\blackboard\apps\tomcat\conf\jk\wrapper.properties as "wrapper.properties.prod"

Step 2 Copy wrapper.properties.prod and name the copy "wrapper.properties.debug"

Step 3 Insert `-Djavax.net.debug=all,record,plaintext` into wrapper.properties.debug

Go to line 163 of wrapper.properties.debug

Edit that line to read:

```
"wrapper.cmd_line=$(wrapper.javabin) $(wrapper.java_opts) -
Djavax.net.debug=all,record,plaintext -
```

```
Djava.security.policy=="$(wrapper.tomcat_policy)" -
Djava.security.manager -Dtomcat.home="$(wrapper.tomcat_home)"
-Dblackboard.home="$(bbapp.root)" -
Dbbservices_config="$(bbapp.root)\config\service-
config.properties" -
Dorg.apache.tomcat.apps.classpath="$(wrapper.class_path.apps)"
-classpath $(wrapper.class_path) $(wrapper.startup_class) -
config $(wrapper.server_xml)"
```

Step 4 Delete wrapper.properties then copy wrapper.properties.debug and name the copy "wrapper.properties"

Step 5 Restart services

Step 6 Login with the LDAP username and password

Step 7 Copy the SSL-connection trace information from D:\blackboard\logs\tomcat-jvm-stdout.txt. See the log file example below.

Step 8 Repeat Steps 6 and 7 until debugging is complete.

Step 9 Delete wrapper.properties then copy wrapper.properties.prod and name the copy "wrapper.properties"

Step 10 Restart services

Log File Example

If the SSL -connection-setup process cannot continue, the reason for the SSL connection setup failure is printed to the tomcat-jvm-stdout.txt log. After this failure appear in the log the SSL-debug output stops. There are a number of reasons why the application server may have trouble connecting to the LDAP server over SSL. The problem can be found in the SSL-debug output. Open the tomcat-jvm-stdout.txt log; go to the end of the debug output (where it gives the reason for quitting) and then scroll backwards through the output, looking for the detailed error message.

For example, in the debug output below, the end of the output shows the message "Thread-31, SEND SSL v3.0 ALERT: fatal, description = certificate_unknown". Scrolling backwards through the log, the message "out of date cert" appears before the last certificate was processed; the certificate's information shows that the certificate had expired in 2002.

The example below includes the beginning of the debug output and the last section with the error:

```
keyStore is :
keyStore type is : jks
init keystore
init keymanager of type SunX509
trustStore is: /usr/java1.3/jre/lib/security/cacerts
trustStore type is : jks
init truststore
adding as trusted cert: [

……

out of date cert: [
[
  Version: V3
  Subject: O=HC, CN=204.165.200.98
  Signature Algorithm: SHA1withRSA, OID = 1.2.840.113549.1.1.5


  Key:  com.sun.net.ssl.internal.ssl.JSA_RSAPublicKey@187197
```

```
  Validity: [From: Sun Jun 18 07:16:00 EDT 2000,
             To: Tue Jun 18 07:16:00 EDT 2002]
  Issuer: O=HC, OU=Organizational CA
  SerialNumber: [    021411e9 6f9a05e1 28e9293c c80ae5b5 1166338c 1cbc0201 0c]

Certificate Extensions: 3
[1]: ObjectId: 2.16.840.1.113719.1.9.4.1 Criticality=false
Extension unknown: DER encoded OCTET string =
0000: 04 82 01 BD 30 82 01 B9   04 02 01 00 01 01 FF 13  ....0...........
0010: 1D 4E 6F 76 65 6C 6C 20   53 65 63 75 72 69 74 79  .Novell Security
0020: 20 41 74 74 72 69 62 75   74 65 28 74 6D 29 16 43   Attribute(tm).C
0030: 68 74 74 70 3A 2F 2F 64   65 76 65 6C 6F 70 65 72  http://developer
0040: 2E 6E 6F 76 65 6C 6C 2E   63 6F 6D 2F 72 65 70 6F  .novell.com/repo
0050: 73 69 74 6F 72 79 2F 61   74 74 72 69 62 75 74 65  sitory/attribute
0060: 73 2F 63 65 72 74 61 74   74 72 73 5F 76 31 30 2E  s/certattrs v10.
0070: 68 74 6D 30 82 01 4A A0   1A 01 01 00 30 08 30 06  htm0..J.....0.0.
0080: 02 01 01 02 01 46 30 08   30 06 02 01 01 02 01 0A  .....F0.0.......
0090: 02 01 69 A1 1A 01 01 00   30 08 30 06 02 01 01 02  ..i.....0.0.....
00A0: 01 46 30 08 30 06 02 01   01 02 01 0A 02 01 69 A2  .F0.0.........i.
00B0: 06 02 01 17 01 01 FF A3   82 01 06 A0 5A 02 01 02  ............Z...
00C0: 02 02 00 FF 02 01 00 03   0D 00 80 00 00 00 00 00  ................
00D0: 00 00 00 00 00 00 03 09   00 80 00 00 00 00 00 00  ................
00E0: 00 30 18 30 10 02 01 00   02 08 7F FF FF FF FF FF  .0.0............
00F0: FF FF 01 01 00 02 04 06   F0 DF 48 30 18 30 10 02  ..........H0.0..
0100: 01 00 02 08 7F FF FF FF   FF FF FF FF 01 01 00 02  ................
0110: 04 06 F0 DF 48 30 58 A1   58 02 01 02 02 02 00 FF  ....H0X.X.......
0120: 02 01 00 03 0D 00 40 00   00 00 00 00 00 00 00 00  ......@.........
0130: 00 00 03 09 00 40 00 00   00 00 00 00 00 30 18 30  .....@.......0.0
0140: 10 02 01 00 02 08 7F FF   FF FF FF FF FF 01 01 02  ................
0150: 00 02 04 11 E9 6F 9A 30   18 30 10 02 01 00 02 08  .....o.0.0......
0160: 7F FF FF FF FF FF FF FF   01 01 00 02 04 11 E9 6F  ...............o
0170: 9A A2 4E 30 4C 02 01 02   02 01 00 02 02 00 FF 03  ..N0L...........
0180: 0D 00 80 00 00 00 00 00   00 00 00 00 00 00 03 09  ................
0190: 00 80 00 00 00 00 00 00   00 30 12 30 10 02 01 00  .........0.0....
01A0: 02 08 7F FF FF FF FF FF   FF FF 01 01 00 30 12 30  .............0.0
01B0: 10 02 01 00 02 08 7F FF   FF FF FF FF FF 01 01 02  ................
01C0: 00                                                 .


[2]: ObjectId: 2.5.29.35 Criticality=false
AuthorityKeyIdentifier [
KeyIdentifier [
0000: 01                                                 .
]

]

[3]: ObjectId: 2.5.29.15 Criticality=false
KeyUsage [
  DigitalSignature
  Key Encipherment
]

]
  Algorithm: [SHA1withRSA]
  Signature:
0000: 50 75 22 E0 14 FE E7 50   FE 44 3B 36 D2 C8 EC 10  Pu"....P.D;6....
0010: 49 8D 48 1D 6F E6 91 1A   05 1E 8E FD 69 D3 4D 70  I.H.o.......i.Mp
0020: C3 3C FE 14 D0 D4 99 DE   CA BF 23 57 80 A0 04 F2  .<........#W....
0030: 45 33 BD B0 53 2D 72 A1   43 DD 7C 80 DD 6B 3E EC  E3..S-r.C....k>.
0040: 94 73 F9 83 21 2C 80 17   B1 CE 6E 19 FD 14 FF A8  .s..!,....n.....
0050: C0 CB 51 C7 1A C1 C0 E4   71 2F 46 9D 50 91 52 E8  ..Q.....q/F.P.R.
0060: 5B CA 24 84 FF 7F 3E 84   32 09 AA 43 66 E8 CD AB  [.$...>.2..Cf...
0070: 65 EC 5C 89 88 43 3C 15   07 3C 9D 52 AA CF 31 A1  e.\..C<..<.R..1.
0080: C9 B6 3A 7A CC 35 1B 66   CB 3C 80 00 32 15 76 2F  ..:z.5.f.<..2.v/
0090: 86 82 26 31 2F C3 EC 58   CE DD E8 E6 A4 58 6E F0  ..&1/..X.....Xn.
00A0: 70 14 36 DF CB 29 E0 E7   D4 1A 33 62 4E B7 62 3C  p.6..)....3bN.b<
00B0: 77 54 9E AA BE 57 0E 7C   F2 E1 92 D5 B0 AF E9 BB  wT...W..........
00C0: 20 CA A7 AA 4F D4 37 02   DE B2 16 9D FC 7E 90 63   ...O.7........c
00D0: 10 22 49 20 76 97 83 8A   83 0E BB A6 7B B0 E4 DE  ."I v...........
00E0: FB 62 51 FD 92 EB 9F C7   B6 91 F2 94 5C 93 29 11  .bQ.........\.).
00F0: B9 A2 AE 28 46 00 BE 14   EC 1C F8 6C 63 A3 10 BA  ...(F......lc...

]
Thread-31, SEND SSL v3.0 ALERT:  fatal, description = certificate unknown
Thread-31, WRITE:  SSL v3.0 Alert, length = 2
```

LDAP Fail Over Considerations

Overview

Administrators must determine how the *Blackboard Learning System* should function if the directory servers are not functioning correctly at the time of an authentication request or if a user who does not exist in the LDAP database attempts to login to the *Blackboard Learning System*.

Automatic authentication fail over may be set for one or both of the following properties:

```
auth.type.ldap.error_fallback_to_bb
auth.type.ldap.user_not_found_fallback
```

Automatic fail-over functionality poses certain security risks that are discussed below in the **Security Risks** section.

Note: The behaviors listed in the LDAP Scenario topic on the Troubleshooting page do not apply if the default configuration is changed.

Automatic fail over for LDAP Server error

LDAP authentication is intended as an enterprise-level integration; therefore, the expectation is that the LDAP server will be managed administratively as a mission-critical system. The LDAP interface was developed to depend upon the constant availability of the directory servers.

Automatic fail over in the case of LDAP server error enables institutions that are not supporting LDAP as a mission-critical system to allow users access to the system if the LDAP server fails. Automatic authentication fail over will allow the *Blackboard Learning System* to continue to run, in the event that the LDAP server(s) does not function correctly. In this instance, automatic fail over is set for the `auth.type.ldap.error_fallback_to_bb` property.

Automatic fail over for users who do not exist in LDAP database

This fail over option allows users who do not exist in the LDAP database to log into *Blackboard Learning System*. Examples are an “Administrator” user or students who are auditing, but are not enrolled, in a class. In this instance, automatic fail over is set for the `auth.type.ldap.user_not_found_fallback` property.

Security Risks

Automatic authentication fail over has some additional security risks:

- Passwords are not synchronized: *Blackboard Learning System* will not know the passwords in LDAP, so Administrators have to keep track of separate passwords.
- Security back doors: Automatic fail over for authentication may introduce serious security problems not related to the *Blackboard Learning System*. For example, if a user attempted a denial of service (DOS) they could shut down the directory server and attempt to log in with default passwords, which is the user name.

To address this issue:
Synchronizing user data between the LDAP servers and the *Blackboard Learning System* (via Snapshot or the Event Driven APIs) can prevent failover from using default passwords and also enable failover to require the same password as the normal LDAP authentication.

Changing the default configuration for LDAP Server Error

Follow the steps below to enable authentication fail over in case of an LDAP server error:

Step 1 Set the authentication property `auth.type.ldap.error_fallback_to_bb` to "true".

Step 2 Populate the *Blackboard Learning System* database with correct username and password information.

Step 3 Restart the application server.

For more information see the topic on LDAP Properties.

Changing the default configuration for users that do not exist in LDAP database

Follow the steps below to enable authentication fail over if a user does not exist in the LDAP database:

Step 1 Set the authentication property `auth.type.ldap.user_not_found_fallback` to "true".

Step 2 Populate the *Blackboard Learning System* database with correct username and password information.

Step 3 Restart the application server.

For more information see the topic on LDAP Properties.

LDAP with Active Directory

Overview

Administrators may decide to use Active Directory via LDAP. This may be done by connecting to Active Directory via an anonymous bind or by using a privileged user. The following topic explains how to set up an anonymous connection or a privileged connection, and some accompanying security risks.

Connecting via an anonymous bind

Active Directory does not allow anonymous access by default, but Administrators may enable anonymous searches if they choose.

Note: There are security risks with allowing anonymous LDAP binds with Active Directory; in this case, any users who have network access to the Active Directory server can search Active Directory.

To enable anonymous searches on the Active Directory server, follow the steps below:

Step 1 On the Windows 2000 Active Directory server, run the Active Directory Users and Groups administration tool.

Step 2 Select the top level of the directory from the tree view in the left hand panel, and right click. Select the first item on the menu, which begins with **Delegate Control**. Click **Next**.

Step 3 In the next window, titled "Users or Groups" click **Add**.

Step 4 On the next list, select **ANONYMOUS LOGON** and click **Add**. Administrators may also need to select **Everyone** and the **Guests** group, depending on how Active Directory is configured. Click **OK** when this is done. Click **Next**.

Step 5 Select **Create a custom task to delegate** and click **Next**.

Step 6 In the next list, select **Read**. **Read All Properties** will be selected at the same time. Click **Next**.

Step 7 Click **Finish**.

Connecting via a privileged bind

By default, Active Directory can only be searched via LDAP if a privileged user is used to connect to the LDAP server. A privileged bind requires the distinguished name (DN) and password for the user. There are two options for connecting via a privileged bind:

- Create a new Windows user within Active Directory. Assign this user only the right to read access to the directory. Use this user as the privileged user.
- Use an existing Windows user as the privileged user.

Note: There are security risks with connecting via a privileged bind to Active Directory. Any user who can navigate to the file system and locate the `authentication.properties` file may find the user ID and password of the privileged user.

Troubleshooting LDAP with Active Directory

For Administrators using a Windows workstation, the LDP executable may be used to troubleshoot LDAP authentication properties. The LDP executable, found on the Windows 2000 Server CD in the \SUPPORT\TOOLS folder, allows LDAP operations to be performed against Active Directory and includes a graphical user interface. More information about this may be found in the topic Troubleshooting LDAP Authentication Properties for Windows.

The only change for this procedure is in Steps 2, 10 and 12. Follow the steps below when using the LDP executable against Active Directory:

Step 1 Login as the Windows user (username, password, domain) whose username and password are being used for the privileged bind.

Step 2 Add 'sAMAccountName' to the Attributes: field and click OK.

Step 3 Enter the (sAMAccountName=someWindowsUserName) in the Filter: field, where 'someWindowsUserName' is the Windows username that will be used as the privileged user for binding to LDAP.

For Administrators using a UNIX workstation, the LDAP Browser may be used to troubleshoot LDAP authentication properties. More information about this may be found in the topic Troubleshooting LDAP Authentication Properties for UNIX.

Introduction to Web Server Delegation Authentication

Overview

Web servers support a range of technologies for identifying and authenticating users. By default, the application server (Tomcat) handles authentication. During Web Server Delegation authentication, the application server delegates some aspects of authentication to the Web server (Apache® or IIS).

Management

When using Web server delegation much of the work related to the authentication is handled on the Web server.

The process for Web server authentication is:

Step 1 Obtain the credentials of the user.

Step 2 Verify the user's credentials.

Step 3 Set a header in the request that is populated with the value for the CGI variable REMOTE_USER.

Step 4 If a custom module has been created, it will look for the header name that was set in Step 3. If it finds this header, the value will be used as the external user ID.

Step 1 through Step 3 must be implemented by a Web server module/filter. For example, Blackboard provides an Active Directory filter for IIS. Clients using UNIX can set up Kerberos with Apache. The System Administrator must ensure that the authentication filter has been installed on the Web server. If using Windows®, please see the Active Directory section for more information.

Step 4 takes place within the application server and requires that the `authtype.webserver.impl` is specified in the `authentication.properties` file.

Note: Kerberos authentication may be incompatible with direct access to WebDAV. Institutions using this type of authentication may be able to take advantage of WebDAV by first authenticating with the *Blackboard Learning System*, and then launching the Web Folder from within the user interface.

Implementation

The following are steps for implementing Web server delegation:

Step 5 Install the appropriate filter (for example, Kerberos on Apache).

Step 6 Update the `authentication.properties` file if needed. See the Property Configuration topic for more details.

Step 7 Restart the application server and the Web server.

Web Server Delegation with Windows 2003

Overview

It is necessary to complete some additional steps to use Web Server Delegation with Windows 2003. Note that Active Directory authentication is a form of Web Server Delegation.

Configure Web Server Delegation with Windows 2003

Follow these steps to configure Web Server Delegation with Windows 2003.

Step 1 Edit the `authentication.properties` file
- see the Property Configuration section

Step 2 Edit the `bb-config.properties` file
- change the file to `bbconfig.auth.type=webserver`

Step 3 Open the following file for editing:

`\Blackboard\apps\tomcat\config\jk2.properties file`

Step 4 Go to the end of the file and paste the following line:

`request.tomcatAuthentication=false`

Step 5 Deploy configuration updates

```
cd BB6_DEPLOY_DIR\tools\admin
PushConfigUpdates.bat
```

Step 6 Run `PushConfigUpdates.bat`. PushConfigUpdates.bat will overwrite any customizations to the configuration.

Step 7 Run the `websitereinstall.bat` which will delete and recreate the Bb website. All custom changes to IIS will be removed, including configurations to support SSL Choice.

Step 8 Open ISS 6.0.

Step 9 Right click on the Blackboard website.

Step 10 Select **Properties**.

Step 11 In the IIS Management Console for the *Blackboard Learning System*, select the authentication method or methods. The options are Basic, Digest, and Integrated Windows authentication. One or more options may be selected.

Step 12 Select the **ISAPI Filters** tab. Verify that the `Windows.dll` has been added. Also, verify the order of the filters. They should be in the following order: Sessiontracker, Jakarta, Windows. If they are not in order, move them into the correct order.

Step 13 Click **Apply**.

Step 14 Select the **Directory Security** tab.

Step 15 Click **Edit** under Authenication and Access Control.

Step 16 Disable the **Enable Anonymous Access option**.

Step 17 Enable the **Intergrated Windows Authenication**.

Step 18 Click **OK**.

Step 19 Click **Apply**.

Step 20 Click **OK**.

Introduction to Active Directory Authentication

Overview

Blackboard Academic Suite supports Web server delegated authentication. It includes a Web server filter for Active Directory that will authenticate users against an institution's Active Directory server. All the files necessary to support Active Directory authentication are included with the *Blackboard Learning System*.

Active Directory Authentication

Microsoft® Active Directory™ is a part of the Windows 2000 network architecture. It allows organizations to share and manage information about network resources and users. In addition, Active Directory acts as the central authority for network security, letting the operating system verify a user's identity and control his or her access to network resources, such as data, applications, or printers. There are a number of benefits to using Active Directory. Administrators have a single point of management for Windows-based user accounts, clients, servers, and applications. Active Directory authentication also allows for standardized business rules for applications and network resources.

Limitations

The user's *Blackboard Learning System* User Name is associated with their Microsoft Windows' login. This means that the user's database record is linked to their Windows User Name and password allowing them to only have one *Blackboard Learning System* User Name. For example, Windows user "jdoe" can only have one *Blackboard Learning System* login; he or she cannot be "Instructor1" and "Student2" in the *Blackboard Learning System*.

To ensure the safety of user accounts, the browser must be closed and the user must log off of Windows when a session on a shared computer is ended.

Please contact Blackboard Technical Support with any questions regarding these limitations.

Active Directory Authentication and Portal Direct Entry

Blackboard does not currently support using Web Server Delegation with Portal Direct Entry. Clients who would like to set up a customized authentication with Web Server Delegation and Portal Direct Entry should contact Blackboard Global Services.

Active Directory Topics

This section includes the following topics:

Active Directory Configuration

Overview

The properties set for Active Directory authentication are found in the `auth.type.webserver` section of the `authentication.properties` file.

Note: There is no special configuration needed for IIS.

File format

The `authentication.properties` file contains a series of properties that must be set before authentication against the institution's Active Directory server or servers can occur. Each property is listed with an equal sign followed by the corresponding value.

Set authentication type

Prior to editing the `authentication.properties` file, the authentication type (`bbconfig.auth.type`) must be edited in the `bb-config.properties` file. This allows the *Blackboard Learning System* to select the appropriate set of `auth.type*.*` entries at start up. The following steps are instructions for setting the authentication to Active Directory. These steps are also included in the instructions for setting up Web Server Delegation with Windows 2003.

Step 1 Edit the `authentication.properties` file

- see the Property Configuration section

Step 2 Edit the `bb-config.properties` file

- change the file to `bbconfig.auth.type=webserver`

Step 3 Deploy configuration updates

```
cd BB6_DEPLOY_DIR\tools\admin
PushConfigUpdates.bat
```

Note: `PushConfigUpdate` will over write any customizations to the configuration of the application.

Step 4 ***IIS Clients only:*** Run the Website Reinstaller tool. This tool removes and reinstalls IIS.

```
cd blackboard\tools\admin
WebsiteReinstall.bat
```

Note: All custom changes to IIS will be removed after running this tool. For example, configurations to IIS to support SSL Choice will be erased.

Step 5 ***IIS Clients only:*** In the IIS Management Console for the *Blackboard Learning System*, select the authentication method or methods. The options are Basic, Digest, and Integrated Windows authentication. One or more options may be selected.

Property Configuration

The table below details the properties configured through the `authentication.properties` file.

Property	Description
`auth.type.webserver.impl=blackboard.platform.security.authentication.WindowsAuthModule`	Defines the class which must conform to the HttpAuthModule interface. The default value, `blackboard.platform.security.authentication.WindowsAuthModule`, should not be changed unless the institution builds and implements its own class for Web Server Delegation authorization. If using Kerberos or another type of Web Server Delegation and it is configured to set the standard REMOTE_USER header then use `blackboard.platform.security.authentication.ExternalAuthModule`.
`auth.type.webserver.user_account=reconcile, create or deny`	Describes how external users are handled by the *Blackboard Learning System*. It is set to "reconcile", "create" or "deny". The first thing the system does for any of these settings is check for an existing user account that is associated with the external User ID. If one is not found, the following will occur: • If set to "reconcile" the system will display a page that allows the user to login as an existing Blackboard user once, associate that Blackboard user account with the external user ID, and then will log them in via Web server delegated authentication in the future. • If set to "create", the system will try and create a new user account with minimal information that has to be updated by the user or Administrator. The user name is automatically "webserver-user-user number" (for example, webserver-user-100). • If set to "deny", a message will be displayed for the user to contact the Administrator.
`auth.type.webserver.def_key`	Set to "BatchUid" or "UserRegistry". UserRegistry is the default setting. BatchUid is used when the Web server REMOTE_USER value should match a Blackboard user account's Batch UID. Customers who use the Snapshot Tool to set the Batch UID equal to the

	REMOTE_USER value should set the `auth.type.webserver.user_account` to deny.

Example

The example below details the properties configured for Web server delegation through the `authentication.properties` file.

```
auth.type.webserver.impl=blackboard.platform.security.authentication.WindowsAuthModule
auth.type.webserver.user_account=reconcile
```

* Acceptable entries for user_account are: reconcile, create, deny

Active Directory Security Considerations

Overview

Active Directory authentication is intended as an enterprise-level integration. Active Directory works with Microsoft® Windows® and creates a centralized place to share and manage information about users and the network. Active Directory also acts as the system's center for network security. There are a few security considerations to keep in mind when using *Blackboard Learning System* and Active Directory.

Security Considerations

When the Active Directory authentication is implemented a user has a single Microsoft Windows User ID and a single *Blackboard Learning System* User Name. The database has a single entry for this user. The level of security for the Windows User ID and password is the same as that for the *Blackboard Learning System* User Name and password.

When users are working on a shared computer they must close the browser and log out of Windows at the end of the session to ensure their security in maintained.

Passport Authentication

Overview

Blackboard Learning System includes an IIS filter and an implementation on the HttpAuthModule that can be installed on a Windows server to authenticate against Microsoft's .NET Passport® Web Service.

Note: *Blackboard Learning System* does not currently include an Apache module to allow Passport authentication on UNIX installations.

Passport Authentication

The Microsoft®.NET Passport service allows users to use a single name and password for all participating Web sites. This single user name and password reduces the amount of information that users must remember and enables institutions to streamline their processes.

Limitations

The limitations of the Passport authentication module are summarized below:

- Passport may be incompatible with direct access to WebDAV. Institutions using this type of authentication may be able to take advantage of WebDAV by first authenticating with the *Blackboard Learning System*, and then launching the Web Folder from within the user interface.
- Passport SDK 1.4.2 requires Windows NT 4.0 or Windows 2000.
- The Passport profile cannot be retrieved, except for the Passport User ID (PUID), because Passport Service cannot guarantee that any profile data (except PUID) will be available.
- The Passport profile cannot be updated with data from the *Blackboard Learning System* User record.
- Passport profile data cannot be imported into Snapshot or the Event Driven API.

Please contact Blackboard Technical Support with any questions regarding these limitations.

Overview of Blackboard and Passport Interaction

The following steps provide an overview of the interaction between the *Blackboard Learning System* and Passport:

Step 1 Register for a test Passport account that enables the Administrator to login to any Passport-affiliated site.

Step 2 Register an instance of the *Blackboard Learning System* with the Microsoft Passport server.

Step 3 Open the *Blackboard Learning System* Login page and select the **Login** button for Passport. The user is re-directed to the Microsoft Passport server and logs into Passport.

Step 4 Passport re-directs the browser to the *Blackboard Learning System*.

Step 5 The `PassportAuth` filter on the *Blackboard Learning System* Web server extracts the Passport User ID and sends it to the *Blackboard Learning System* application server.

Passport Topics

This section includes the following topics:

Passport Application Registration

Overview

This section describes how to obtain the Site ID and encryption key to enable the system to interact with Passport. This allows the PassportAuth filter to work correctly with Passport.

URL Functions

Passport registration sets URLs for the following:

- Login and logout – upon logout the user will be logged out of all Passport sites and the *Blackboard Learning System*, and all cookies will be cleared.
- Images used for co-branding - for example, an institution could add their logo to the Passport Login page.
- Institution policies – such as security and privacy.

For more information please see the Microsoft Passport Web site http://www.passport.com

Passport Installation and Configuration

Overview

This topic provides an overview of the Passport installation process. This process consists of four phases that will enable the System Administrator to use Passport authentication.

The four parts of the Passport Installation and Configuration include:

Step 1 Installation and Configuration
Step 2 Configure the *Blackboard Learning System* for Passport
Step 3 Configure the *Blackboard Learning System* Passport authentication
Step 4 Test the *Blackboard Learning System* Passport Login

Note: Microsoft requires that the Passport checklist be submitted before a Passport site is approved. Administrators install Passport on a test server at the institution. To do this, the Administrator must obtain a PREP (pre-production) site ID from Microsoft. This test is performed against Microsoft's PREP Passport server. After completing the Installation and Configuration on a test server the checklist for the test server is submitted to Microsoft. If this is passed, then the Administrator can download a production Site ID that will be used to authenticate against the Passport production server.

Part I. Passport Installation and Configuration

The following steps outline the Passport installation process.

Step 5 Go to the.NET My Services Manager, located at www.netmyservicesmanager.com and obtain an account.

Step 6 Select **Create an application in our DEV/TEST environment**. .NET My Services Manager will walk through the registration of the Passport PREP Site ID.

Step 7 Download the Passport encryption key for Site ID from.NET My Services Manager to "C:\Passport".

Step 8 Go to http://current-register.passporttest.com/and obtain a test Passport account.

Step 9 Download Passport SDK 1.4.2 and run it. This is a self-executing.exe file. When asked which IIS Web site object to install enter the default.

Step 10 The install process triggers the server to reboot.

Step 11 Close the browser.

Step 12 Go to the IIS Management console and stop the Blackboard IIS object. Start the default IIS object.

Step 13 Open the browser to "http://localhost/PassportTest/pleasewait.asp?Refresh=True&NewId=True&Env=Prep"

Step 14 Go to Passport Admin utility and create a new Web Site Name "blackboard_bb_bb60" with hostname = YOUR-MACHINE-NAME and IP=YOUR-MACHINE-IP-ADDRESS, set Site ID = YOUR-SITE-ID, commit changes. The Site ID is obtained from Passport during Step 2. DO NOT refresh the network map.

Step 15 Install the encryption key to *Blackboard Learning System*:

```
C:\Passport\ > partnerYOUR_SITE_ID_1.exe /addkey /s
blackboard_bb_bb60

C:\Passport\ > partnerYOUR_SITE_ID_1.exe /makecurrent /t 0 /s
blackboard_bb_bb60
```

Step 16 Turn off the default Web site.

Step 17 Set the Web server to use the *Blackboard Learning System* – Passport application. For more information see the Set Authentication Type section.

Part II. Configure the *Blackboard Learning System* for Passport

The `authentication.properties` file contains a series of properties that must be set before authentication against Passport can occur. Each property is listed with an equal sign followed by the corresponding value.

Prior to editing the `authentication.properties` file, the authentication type (`bbconfig.auth.type`) must be edited in the `bb-config.properties` file. This allows the *Blackboard Learning System* at start up to select the appropriate set of `auth.type*.*` entries. The following are instructions for setting the authentication to Passport:

Step 1 Edit the `authentication.properties` file
- edit as needed (see the Property Configuration section)

Step 2 Edit the `bb-config.properties` file
- change the file to `bbconfig.auth.type=passport`

Step 3 Deploy configuration updates

```
cd BB6_DEPLOY_DIR\tools\admin
PushConfigUpdates.bat
```

Note: `PushConfigUpdate` will over write any customizations to the configuration of the application.

Step 4 Run the Website Reinstaller tool. This tool removes and reinstalls IIS.

```
cd blackboard\tools\admin
WebsiteReinstaller.bat
```

Note: All custom changes to IIS will be removed after running this tool. For example, configurations to IIS to support SSL Choice will be erased.

Part III. Configure the Blackboard Passport Authentication

Once the `bbconfig.auth.type` is set correctly, open the `authentication.properties` file in an editor and edit the properties to match your institution. Descriptions of the properties appear in the following section.

The following `authentication.properties` files must be configured for Passport authentication.

Property	Description
`auth.type.passport.impl=blackboard.platform.security.authentication.PassportAuthModule`	Defines the class which must conform to the `HttpAuthModule` interface. The default value, `blackboard.platform.security.authentication.PassportAuthModule`, should not be changed unless the institution builds and implements its own class for Passport authorization.
`auth.type.passport.user_account`	Describes how external users are handled by the *Blackboard Learning System*. It is set to "reconcile", "create" or "deny". The first thing the system does for any of these settings is check for an existing user account that is associated with the external User ID. If one is not found, the following will occur: • If set to "reconcile" the system will display a page that allows the user to login as a Blackboard user once, associate that Blackboard user account with the external user account, and then log them in via Web server delegated authentication in the future. • If set to "create", the system will try and create a new user account with minimal information that has to be updated by the user or Administrator. The user name is automatically "webserver-user-user number" (for example, webserver-user-100). • If set to "deny", a message will be displayed for the user to contact the Administrator.
`auth.type.passport.def_key`	Set to "BatchUid" or "UserRegistry". UserRegistry is the default setting. BatchUid is used when the Passport UID value should match a Blackboard user account's Batch UID. Customers who use the Snapshot Tool to set the Batch UID equal to the Passport UID value should set the `auth.type.passport.user_account` to deny.

Note: If the properties are changed, the *Blackboard Learning System* must be restarted for the changes to take effect.

Example

The following is an example of the `authentication.properties` files configured for Passport authentication.

```
auth.type.passport.impl=blackboard.platform.security.authentication.PassportAuthModule
auth.type.passport.user_account=reconcile
* Acceptable entries for user_account are: reconcile, create, deny
```

Part IV. Test the Blackboard Passport Login

The following steps outline how to test the Blackboard Passport Login.

Step 1 After completing all of the steps in the installation restart the application server.

Step 2 Login to test the Blackboard Passport login.

Step 3 Log out to test the Blackboard Passport logout function.

To certify the Blackboard Passport for production, complete Microsoft's production checklist for Passport applications.

Passport Troubleshooting

Overview

Upon completing the installation and configuration of the Passport authentication module, ensure the process was successful by reviewing the procedures below.

Passport Manager Web server filter

The Passport Manager Web-server filter is installed as part of the Passport SDK 1.4.2 installation process. To verify that it has been installed, follow the steps below:

Step 1 Install Passport SDK 1.4.2, per the instructions described in the Installation and Configuration section.

Step 2 After the SDK installation is complete, open the Internet Information Services console and right-click on the icon representing the web server.

Step 3 Select Properties.

Step 4 An Internet Information Services Properties window will open. Click the **Internet Information Services** tab.

Step 5 In the Master Properties section of that tab, select **WWW Service** and click **Edit**.

Step 6 A WWW Service Master Properties window will open. Click the **ISAPI Filters** tab.

Step 7 The Passport Manager filter should be listed with a green status arrow and a High priority.

Step 8 Cancel out of the Master Properties window. Cancel out of the Internet Information Services Properties window.

Passport authentication Web server filter

The Passport authentication Web server filter populates incoming login requests with Passport authentication information for the *Blackboard Learning System*. It is installed by using the command-line process in the Set Authentication Type section. To verify that it has been installed, follow the steps below:

Step 1 Install Passport SDK 1.4.2, per the instructions described in the Installation and Configuration section.

Step 2 After the SDK installation is complete, open the Internet Information Services console and right-click on the icon representing the Web site (for example, blackboard_bb_bb60 -- substituted for *"bb60"* will be the ID for this *Blackboard Learning System* installation).

Step 3 Select **Properties**.

Step 4 A Properties window will open. Click the **ISAPI Filters** tab.

Step 5 The Passport authentication filter should be listed with a green status arrow and a High priority.

Step 6 Cancel out of the Master Properties window. Cancel out of the Internet Information Services Properties window.

Registry data installation

The Passport 1.4.2 SDK installation process configures a number of registry settings. In Windows, the SDK installation process also installs a GUI that exhibits the current registry settings. To use this GUI and verify that the registry data has been installed correctly, follow the steps below:

Step 1 Install Passport SDK 1.4.2, per the instructions described in the Installation and Configuration section.

Step 2 After the SDK installation is complete, click **Start** > **Programs** > **Microsoft Passport** > **Passport Administration Utility**.

Step 3 Select the Web site name for this installation (i.e., blackboard_bb_YOUR-SITE-NAME).

Step 4 The correct Passport Site ID, host name, and IP address for the Web server should be displayed.

Encryption key installation

Passport authentication requires that a Web site specific encryption key be installed on the Web server. This encryption key installation process must be done to correctly configure Passport authentication. The encryption key utility can be used via the command line to install, uninstall, or verify an encryption key for a given Web site. To verify that the encryption key has been installed, follow the steps below:

Step 1 Install Passport SDK 1.4.2, per the instructions described in the Installation and Configuration section.

Step 2 After the SDK installation is complete, open a Windows command line and go to the directory that stores the encryption key utility.

Step 3 Execute the following command (substitute the correct values for directory name, site id, machine name and Blackboard installation id):

```
C:\Passport\ > partnerYOUR_SITE_ID_1.exe /showcurrent
/m MACHINE_NAME /s blackboard_bb_INSTALLATION-UID
```

Step 4 The message "Current key version is 1" should be displayed.

Site ID association

If the User is not directed to the Login page after selecting **Login**, open the Passport Administrator Utility and verify that the correct site ID is associated with the correct Web site name. The site ID is obtained at registration through the My Services Web site. The site ID is included in the name of the downloaded executable that installs the encryption key. For example, C:\Passport\partnerYOUR_SITE_ID_1.exe

---

Passport Administration Utility and Maintenance

Overview

The Passport module should function with minimal maintenance if configured following the instructions in this document. The following are some tips for how to proceed if any problems are encountered.

Common problem

The table below outlines a common issue that arises when authenticating *Blackboard Learning System* users against Passport.

Issue	Action
The Sign In logo does not appear if a user is logged out ***OR*** The Sign Out logo does not appear if a user is logged in	• Check the IIS/Properties/MasterProperties/Edit/ISAPI Filters. The Passport Manager should be installed. If it is not, re-install Passport SDK 1.4.2. • Check Internet Services Manager/localhost/blackboard/blackboard_bb_bb60/Properties/ISAPI Filters. The Passport filter should be installed between the session tracker and "Jakarta". If there is not a filter change the `authentication.type` property again. • Check the Passport Administration Utility. Select the Website, blackboard_bb_bb60. When it is selected the server name, IP address and site ID should appear.

Passport events log

To see the Passport events being logged by Windows follow the steps below:

Step 1 Right-click on the Desktop/My Computer.
Step 2 Select **Manage**.
Step 3 Go to **System** > **Event Viewer** > **Application**.
Step 4 Check the log for Passport events.

Passport Administration Utility options

By default, the *Blackboard Learning System* does not set any administration options, except for those mentioned in this document. To change options for cookie expiration, forced login, or other preferences please see Microsoft's documentation at http://www.passport.com.

Passport Security Considerations

Overview

Passport authentication is intended as an enterprise-level application that allows users to use a single name and password for multiple Web sites, including the *Blackboard Learning System*. There are certain security considerations to remember when using Passport authentication.

Security Considerations

The following are a list of security considerations related to Passport authentication:

- The level of security for a user's *Blackboard Learning System* login information is the same as that of their Passport identity.
- Blackboard user information is hidden from the Passport user database, and vice versa. It is possible that malicious users may exploit Passport vulnerabilities via the browser and retrieve the Passport Login information. Please monitor Microsoft's Passport Web site for security notices.

Custom Authentication

Overview

Custom authentication enables developers to create a customized authentication module that may be plugged into the *Blackboard Learning System's* authentication framework. The *Blackboard Learning System* ships to clients with several authentication techniques, each with its own associated module.

Developing customized authentication allows clients to do either of the following:

- replace the pre-built modules with a module provided by the client or by a third party
- specify a different authentication technique with its own properties

Any module that conforms to the Blackboard End-User Authentication API may be substituted.

The Blackboard Challenge-Response authentication type is installed by default, and will be referred to as "the default authentication type" throughout this section. Module refers to a Java class that implements the interface `blackboard.platform.security.authentication.HttpAuthModule`. Please see the Authentication API for more details on this interface.

Note: For questions on creating highly customized authentication implementations, please contact Blackboard Technical Solutions.

Audience

Developers who wish to create a customized authentication module should have the following:

- a good understanding of general security principles, especially regarding authentication and Web based security
- experience with Java servlet programming

In this section

The following topics are discussed in this section:

Data model

There is no dedicated data model for authentication, however the default authentication process relies on the password and user name attributes of the user entity. See the *Blackboard Building Blocks Extension Developer's* guide for details on the data model.

Object Model

Overview

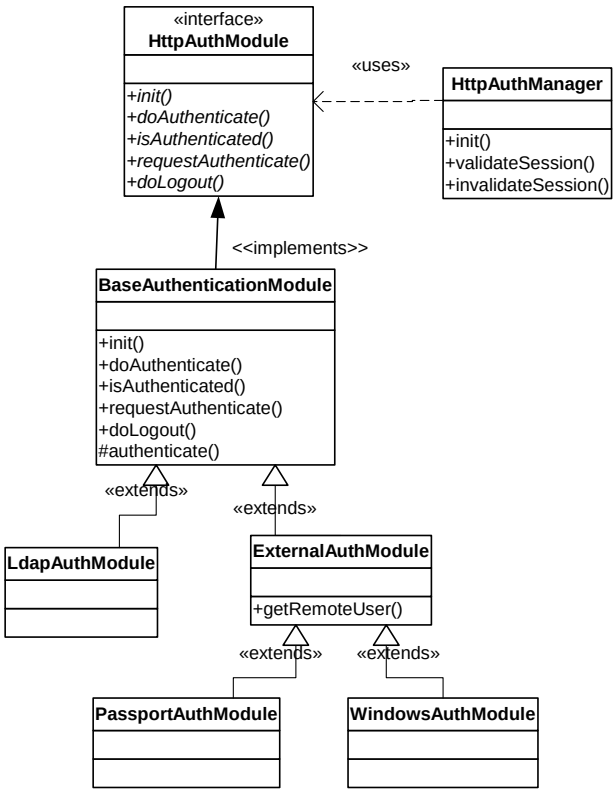
There are three objects used in Blackboard's default authentication type:

- `HttpAuthManager`
- `HttpAuthModule`
- `BaseAuthenticationModule`

The `HttpAuthManager` class and the `HttpAuthModule` interface cannot be modified. Institutions can plug-in their own module for a supported authentication type.

Authentication object model

The `BaseAuthenticationModule` class implements the `HttpAuthModule` interface and supports the default authentication type. The following object model depicts the `HttpAuthModule` implementations for LDAP, Passport, and Web server delegation. IIS Web server delegation support is implemented in the class `WindowsAuthModule`. Apache Web server delegation support is implemented in the class `ExternalAuthModule`.



Note: Although `PassportAuthModule` and `WindowsAuthModule` are public Java classes, it is not permitted to extend these classes. Most authentication modules will extend `BaseAuthenticationModule`.

Authentication Process

Overview

The servlet that initializes the *Blackboard Learning System* calls `HttpAuthManager.init()`which does the following:

- loads the authentication configuration settings
- creates and initializes an instance of an `HttpAuthModule` implementation class
- installs the instance into the `HttpAuthManager` class

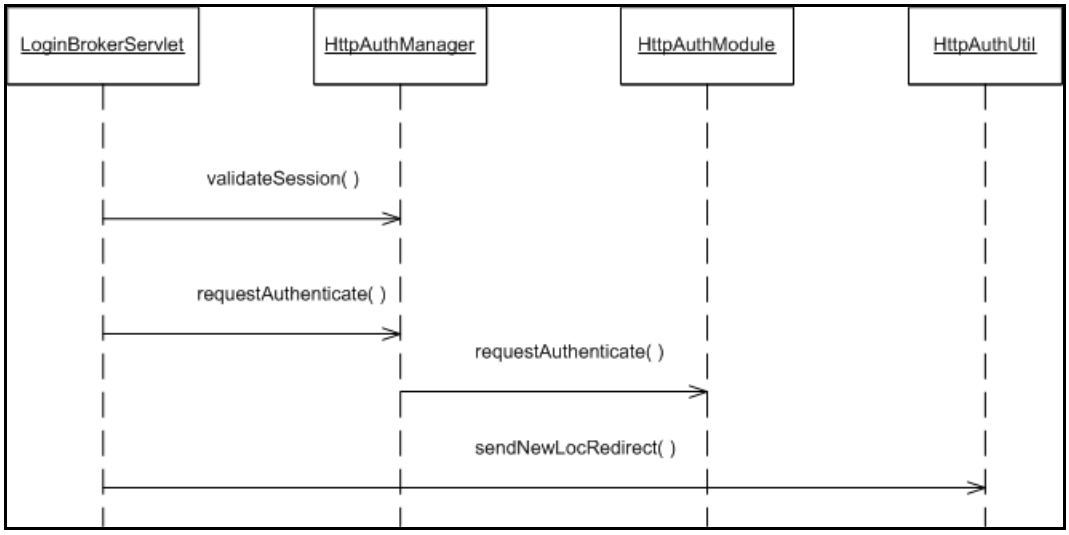
Login

Authentication processing in the *Blackboard Learning System* is centralized in a login-broker servlet, installed at `/webapps/login`. The login-broker servlet processes all login requests for the *Blackboard Learning System*. Login request universal resource identifiers (URIs) take the form of `/webapps/login`.

Every JSP page in *Blackboard Learning System* that requires an authenticated user session does one of the following:

- redirects to the login broker
- displays a link to the login broker if the current user is not logged in or is not logged in as a user with sufficient authorization for the page

The image below demonstrates how the login-broker servlet processes login requests.



The login-broker servlet invokes the method `validateSession()` on a `HttpAuthManager` instance. The `HttpAuthManager` method `validateSession()` calls the `doAuthenticate` method on the module installed on the `HttpAuthManager` class. The servlet uses the boolean result of this method to determine whether to invoke the method `sendNewLocRedirect()` on the `HttpAuthUtil` class, or to trigger the `requestAuthenticate()` method on `HttpAuthManager` object.

The `sendNewLocRedirect()` method on the `HttpAuthUtil` class retrieves the `new_loc` parameter in the request, translates it to a URL and forwards the request to that URL. Please see Customizing Authentication Page Flow for more information on the `new_loc` parameter.

The `requestAuthenticate()` method on the `HttpAuthManager` calls the `requestAuthenticate()`method on the module that was installed during system startup. The module's `requestAuthenticate()` method is called so the implementation may prompt the user for credentials. These credentials must then be submitted to the login broker via HTTP.

Logout

The login-broker servlet also processes all logout requests for the *Blackboard Learning System*. Logout request URIs take the form of `/webapps/login?action=logout`.

The servlet invokes the method `invalidateSession()` on a `HttpAuthManager` instance. After the session is invalidated, the servlet forwards the request to the *Blackboard Learning System's* index page.

When a user's session has timed out, and the user tries to access a page in the *Blackboard Learning System* that requires an authenticated user session, the system redirects the user to the login broker.

Authentication API

Overview

This section explains the different methods that require implementation for the authentication process. The Authentication API is defined by the Java interface `blackboard.platform.security.authentication.HttpAuthModule`. All custom authentication modules must implement this interface.

Please see the *Authentication API Specification* for more information on the arguments and returns for each method.

Authentication Processing methods

The following authentication processing methods are implemented in the authentication module. See the sample authentication module below for an example of the usage of these methods.

`init()` is called by the authentication framework when it creates an instance of `HttpAuthManager`, which in turn creates an instance of the appropriate authentication module and allows the implementation class to perform any required initialization. The sole argument passed in is the `ConfigurationService` object created during system startup. This method is intended to cache properties relevant to custom authentication that are defined at system startup. Installation specific properties can be obtained from the `ConfigurationService` and is different from the properties passed in `setConfig()`. See the example method below.

An authentication module that needs to insert a completely new set of entries in the `authentication.properties` must implement an `init()` method that calls the `configure()` method on the class `HttpAuthConfig`. Please see Configuration File Processing for more details.

```
/*
 * Module initalization. This method gets called when Tomcat starts up.
 */
  public void init(ConfigurationService arg0) throws
    IllegalStateException {
    //Although this is not necessary for subclasses of
    //BaseAuthenticationModule, it is a good practice to do this
    //unless there is a reason not to.
    super.init(arg0);
    try {
    // Set up logging
      _logger = BbServiceManager.getLogService();
    } catch (RuntimeBbServiceException e) {
      e.printStackTrace();
    }
    if (_logger == null) {
      // This println statement will output to the
      //stdout-stderr.log file.
      System.out.println("logger is null");
    } else {
      _logger.logWarning("Custom Auth Module: init()");
    }
}
```

requestAuthenticate() is called if the user is not logged into the *Blackboard Learning System*. The implementation may prompt the user for their credentials. This method may be used for the following:

- to generate a HTTP-302 status in the response, or Javascript, to redirect to a login URL
- to generate a login form programmatically via the servlet API
- to generate an HTTP-401 response
- to forward the user to the appropriate point in the customized *Blackboard Learning System* (for example, by using a `javax.servlet.RequestDispatcher.forward()`)

```
        /*
         * Gets called when the user needs authenticating.
         */
      public void requestAuthenticate(
        HttpServletRequest request,
        HttpServletResponse response) throws BbSecurityException {
        //Do the superclass implementation, redirect to the Default Login
        //Page
        _logger.logWarning("Custom Auth Module: requestAuthenticate()");
super.requestAuthenticate(request, response);
      }
```

Please see Customizing Authentication Page Flow in this document for related information.

doAuthenticate() is called for the implementation to:

- parse the request
- extract any credentials
- perform the authentication work

```
    /*
     * Does the work of authenticating the user.
     */
      public String doAuthenticate(
        HttpServletRequest request,
        HttpServletResponse response)
        throws BbSecurityException,
          BbAuthenticationFailedException,
          BbCredentialsNotFoundException {
        //Do custom processing here, just make sure
        //the user name is returned or the appropriate exception
        //is thrown.
        _logger.logWarning("Custom Auth Module: doAuthenticate()");
        return super.doAuthenticate(request, response);
      }
```

doLogout() is called when the user explicitly logs out. The implementation may perform any tasks required of its authentication authority and remove any session variables. Custom authentication modules that subclass `BaseAuthenticationModule`

may use its implementation to clean up Blackboard Learning System specific session information.

```
/*
 * Gets called when the user explicitly logs out.
 */
public void doLogout(
HttpServletRequest request,
HttpServletResponse response)
throws BbSecurityException {
  _logger.logWarning("Custom Auth Module logging out");
  //If subclassing BaseAuthenticationModule, the superclass
  //implementation can be called. It basically performs the
  //work of invalidating a user's session.
  SessionStub sessionStub = new SessionStub( request );
  sessionStub.disassociateCurrentSessionAndUser();
}
```

Note: This method may not be called since users may close their browser and not explicitly log out.

isAuthenticated() has been deprecated and is only included to maintain backward compatibility. If BaseAuthenticationModule is being extended, no implementation is necessary.

Note: Implementing this method will not over-ride the login broker's internal checks that determine whether the current *Blackboard Learning System* session is authenticated.

Of the above interface methods, the key methods for cooperating with the login broker are doAuthenticate(), requestAuthenticate() and doLogout().

Configuration File Processing methods

The authentication framework expects to find an authentication.properties file in the *Blackboard Learning System's* /blackboard/config directory. The class blackboard.platform.security.authentication.HttpAuthConfig loads the authentication.properties settings into memory and manages the process of configuring an HttpAuthModule object with the appropriate settings.

If an institution wishes to create a custom authentication module that requires a completely new set of entries in the authentication properties file, then all three HttpAuthModule methods described below must be implemented by the custom authentication module.

getAuthType() returns a String identifier for the authentication technique. The four authentication techniques supported by the *Blackboard Learning System* (Blackboard default authentication, LDAP, Passport, and Web server delegation) are represented as "rdbms", "ldap", "passport" and "webserver". The authentication framework loads this String identifier from the bb-config.properties setting for bbconfig.auth.type. The authentication framework then requests that the HttpAuthConfig class, which has loaded all settings found in the authentication.properties setting, creates an HttpAuthConfig instance which stores all authentication.properties settings with the given auth.type identifier. For example, if bbconfig.auth.type is set to ldap, then all property settings that match auth.type.ldap are loaded into a new HttpAuthConfig instance. This method should be implemented for every

authentication module class. For example, with the Sample Custom Autenticationh Module, the following entry needs to be changed in the bb-config.properties file:

```
auth.type.custom.impl=blackboard.authentication.test.CustomAuthModule
```

getPropKeys() returns a String array of the keys to an authentication module's configuration properties. For example, the `BaseAuthenticationModule` has the keys "impl" and "use_challenge". This method should be implemented for every authentication module class. The array must contain all of the property keys for the custom implementation. At a minimum, the method should return the "impl" property which specifies the fully qualified class name of the `HttpAuthModule` implementation. In order to obtain the value of the properties, the authentication module can use the `HttpAuthConfig.getProperty()` method with the property name. Only the property name need be specified, not the entire entry in the authentication.properties file. For example, with the Sample Custom Auth Module, the following entries need to be added to the `authentication.properties` file:

```
auth.type.custom.impl=blackboard.authentication.test.CustomAuthModule
auth.type.custom.prop1=test
```

To get the property of prop1, the following method can be used:

```
// _config has already been defined as a member variable of type HttpAuthConfig
String prop1 = _config.getProperty("prop1");
```

setConfig(HttpAuthConfig config) implies a contract between `HttpAuthModule` and `HttpAuthConfig`, such that `HttpAuthConfig` is expected to supply the correct object for a given property key. Subclasses of `BaseAuthenticationModule` do not need to implement this method, and can use the _config member variable to obtain configuration properties.

Implementation Details

Overview

This topic includes information on how the different authentication types (Blackboard default authentication, LDAP, and Web server delegation) are implemented. This information may be helpful to developers creating a custom authentication for their *Blackboard Learning System*.

Blackboard default implementation

The default implementation (realized in the class `BaseAuthenticationModule`) is implemented as a challenge response protocol, using a form submitted by HTTP-POST. This is a mechanism that avoids sending the actual password over the network in an unprotected fashion. In a naïve authentication implementation, username/password combinations would simply be transmitted across the network in clear text. The problem with this method is that malicious users would be able to see the username and password.

To facilitate greater security the authentication framework generates a pseudo-random number for each authentication attempt that is MD5-hashed against the servlet engine's session ID. This "challenge" string is sent to the client. The challenge string increases system security by improving the chances that that the transmitted code has not been tampered with. When the user enters their password, it is MD5-hashed, then that hash string is combined with the challenge string sent by the server, and the resulting string is MD5-hashed. The resulting string is cryptographically secure in that the hash is one-way (MD5). This means that it is not possible to reconstruct its inputs, or to find inputs that result in the same hashed value. This compound hash is called the "response."

The server, when it receives the client response, performs the same calculations the client performed (except on the server-side, the password is already hashed). Additionally, the challenge string is also re-calculated from the stored pseudo-random number (to help prevent session hijacking). If the results match the client response, the authentication is successful.

LDAP implementation

Blackboard has implemented a simple LDAP authentication module that uses data from the `authentication.properties` configuration file to bind to an LDAP server, or series of LDAP servers, and perform a lookup of a given user. For more information on the configuration settings for LDAP authentication, please see the LDAP Authentication section of this manual.

Web server delegation implementation

Blackboard has implemented a Windows-specific Web server delegation authentication module that assumes the installation of a Blackboard-created ISAPI filter on the Web server; the ISAPI filter populates a request header containing the Active Directory username for the Windows user. The `WindowsAuthModule` authentication module parses the request, extracts the unique identifier and attempts to match that identifier to a user in the *Blackboard Learning System* database.

`WindowsAuthModule's` parent class, `ExternalAuthModule`, is a simple, general implementation for Web server-delegated authentication. It assumes that a module or

filter has been installed on the Web server that populates the Computer Gateway Interface (CGI) variable REMOTE_USER in the request headers.

Extending other Blackboard-created authentication modules

Extending other Blackboard-created implementations such as `WindowsAuthModule` and `PassportAuthModule` is not permitted at this time.

Sample Custom Authentication Module

The following is an example of a Custom Authentication Module:

```
package blackboard.authentication.test;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import blackboard.platform.BbServiceManager;
import blackboard.platform.RuntimeBbServiceException;
import blackboard.platform.config.ConfigurationService;
import blackboard.platform.log.LogService;
import blackboard.platform.security.authentication.BaseAuthenticationModule;
import
blackboard.platform.security.authentication.BbAuthenticationFailedException;
import
blackboard.platform.security.authentication.BbCredentialsNotFoundException;
import blackboard.platform.security.authentication.BbSecurityException;
import blackboard.platform.security.authentication.HttpAuthConfig;
import blackboard.platform.security.authentication.SessionStub;

    /**
     * @author Blackboard Development
     *
     * A Sample Custom Authentication Module.
     */
public class CustomAuthModule extends BaseAuthenticationModule {

    //Save the log service so that we can log events.
      private LogService _logger;

    /*
     * Module initalization. This method gets called when Tomcat starts up.
     */
      public void init(ConfigurationService arg0) throws
        IllegalStateException {
        //Although this is not necessary for subclasses of
        //BaseAuthenticationModule, it is a good practice to do this
        //unless there is a reason not to.
        super.init(arg0);
        try {
        // Set up logging
          _logger = BbServiceManager.getLogService();
        } catch (RuntimeBbServiceException e) {
          e.printStackTrace();
        }
        if (_logger == null) {
```

```
          //This println statement will output to the
          //stdout-stderr.log file.
          System.out.println("logger is null");
        } else {
          _logger.logWarning("Custom Auth Module: init()");
        }
    }

    /*
     * Does the work of authenticating the user.
     */
      public String doAuthenticate(
            HttpServletRequest request,
            HttpServletResponse response)
            throws BbSecurityException,
              BbAuthenticationFailedException,
              BbCredentialsNotFoundException {
        //Since this module uses the standard Learning System login form,
        //use the superclass implmentation. The authenticated() method is
then
        //overriden to customize behavior. Other authentication modules
        //could do something else here if needed, as long as the
        //userid is returned or an exception is thrown.
        _logger.logWarning("Custom Auth Module: doAuthenticate()");
            return super.doAuthenticate(request, response);
      }

        /*
         * Gets called when the user explicitly logs out.
         */
        public void doLogout(
        HttpServletRequest request,
        HttpServletResponse response)
        throws BbSecurityException {

        _logger.logWarning("Custom Auth Module logging out");
        //perform custom logout work here
      }

        /*
         * Gets called when the user needs authenticating.
         */
      public void requestAuthenticate(
            HttpServletRequest request,
            HttpServletResponse response)
        throws BbSecurityException {
        //Do the superclass implementation, redirect to the Default Login
        //Page
        _logger.logWarning("Custom Auth Module: requestAuthenticate()");
        super.requestAuthenticate(request, response);
      }

        /*
         * Returns a String containing the authentication type.
         */
      public String getAuthType() {
            return "custom";
```

```
      }



       /*
        * Returns a String array of properties used by this authentication
        *  type.
        */
      public String[] getPropKeys() {
        String[] props = {"impl", "prop1"};
        return props;
      }

        /*
         * Learning System calls this to hand the authentication module the
         * configuration properties. This gets called before init so that it
can
         * use its properties if needed.
         */
      public void setConfig(HttpAuthConfig config) {
        //Just use BaseAuthenticationModule implementation. It will set
        //the member variable _config. Most modules that extend
        //BaseAuthenticationModule need not override this method.
        super.setConfig(config);
      }

        /*
         * Overrides BaseAuthenticationModule to do something different.
         */
      protected String authenticate(
      String username,
            String password,
            SessionStub sessionStub,
            boolean useChallenge)
            throws BbAuthenticationFailedException,
              BbSecurityException {
      //Do authentication logic here; validate password against an external
      //source... Could also call the superclass implementation to "fall
      //back" to Blackboard.

      //This implmentation just tries out the exceptions that can be
      //thrown.
      if (username.equals("error1")) {
        throw
        new BbAuthenticationFailedException("Custom auth module, error1.");
    }

      if (username.equals("error2")) {
          throw new BbSecurityException("Custom auth module, error2.");
      }

      if (username.equals("error3")) {
        //null parameters, should display standard message.
        throw new BbAuthenticationFailedException();
    }

     //otherwise, return the user passed without checking. The user must
```

```
     //exist in the Learning System for this to work properly.
    return username;
      }
} //End CustomAuthModule
```

Customizing Authentication Page Flow

Overview

This section discusses how to customize the routing between pages in *Blackboard Learning System*. An institution may customize routing by uploading a custom login page to the *Blackboard Learning System* server (via the **Customize Login Page** link on the System Control Panel), or by implementing the method `requestAuthenticate()` method on the `HttpAuthModule` interface.

The **Customize Login Page** function on the System Control Panel allows the Administrator to download a template for the login page for the current Virtual Installation and then upload a modified template to the server. This allows the Administrator to add extra script functionality to the login page hosted by the *Blackboard Learning System*.

Note: Institutions interested in customizing their *Blackboard Learning System* login page must not remove the JSP tags on the page.

Implementing requestAuthenticate()

If the user wishes to redirect to a login form on a page hosted by another application, instead of using the **Customize Login Page** function, the user should implement the the `HttpAuthModule` interface method `requestAuthenticate()` to do a redirect. For authentication to function properly, any login form on a page hosted by another application must submit the form to the login broker at the institution's URL (for example, a login page hosted at "http://another.institution.com" must submit its login form to the URL "http://your.institution.edu/webapps/login").

Redirecting to the original target URL

If a user has clicked a bookmarked URL that leads into the *Blackboard Learning System*, but they are not currently authenticated, the application will route the user to the login broker URI with the originally requested URL preserved. The login broker expects that the rest of the application will preserve the originally-requested URL, in URL-encoded form, as either a hidden form variable or a query string parameter named `new_loc`. Any custom Login page uploaded to the *Blackboard Learning System* server, or any third-party script page that `requestAuthenticate()` redirects to, must keep this contract as well. If not, the *Blackboard Learning System* will route to its default entry page.

Creating and Deploying Custom Implementations

Overview

This section explains how developers may use the implementations provided by *Blackboard Learning System* to create custom authentication modules for their institution. It also reviews how to extend the *Blackboard Learning System* default implementation for a custom authentication module and how to create custom authentication modules for LDAP and Web server delegation.

Extending Blackboard-provided implementations

Rather than creating a custom implementation of the `HttpAuthModule` interface from scratch, clients may create a Java class that inherits from one of the `HttpAuthModule` implementations included with *Blackboard Learning System*. Re-using existing application components allows for custom authentication functionality with a minimum of development effort.

Extending the Blackboard default implementation

The default implementation may be re-used to simplify implementations that only require custom processing of the user credentials. This has the added benefit of being able to re-use the challenge/response features of the default implementation without duplicating code.

The simplest way to extend the default implementation is to extend the class `BaseAuthenticationModule`, over-riding only the `doAuthenticate()` method. The `doAuthenticate()` method returns a string representing the username for the user who has been authenticated. Subclasses of `BaseAuthenticationModule` may either completely over-ride the `doAuthenticate()` method, or re-use the result. The following is an outline of an implementation that re-uses the result:

```
import blackboard.platform.security.authentication.*;
public class CustomAuthModule extends BaseAuthenticationModule
  throws BbSecurityException, BbAuthenticationFailedException,
BbCredentialsNotFoundException
{
  public String doAuthenticate(HttpServletRequest request,
                HttpServletResponse response)
 {
   // Get the default authentication technique's result
   String username = super.doAuthenticate( request, response );
   /* OUTLINE OF POSSIBLE CUSTOM AUTH CODE */
   // Check some data source external to the Blackboard Learning System
   // Do some extra processing related to custom authentication check

   // Return the username if no exceptions have been thrown
   return username;
 }

}
```

Alternatively, any or all of the methods in the `HttpAuthModule` interface may be overridden in custom authentication modules.

Creating a custom LDAP implementation

The Blackboard `LDAPAuthModule` is a very straightforward implementation of LDAP authentication. Custom LDAP implementations that wish to extend the functionality of

the Blackboard `LDAPAuthModule` by binding to a given LDAP schema and performing queries specific to that schema can do so by subclassing from the Blackboard class and overriding its `authenticate()` method. The LDAP module may be extended to add additional behavior (such as additional processing after calling super.authenticate()). However, changing the behavior of the LDAP module cannot be overridden. If the Blackboard-supplied LDAP module is not sufficient, a completely new module (extending `BaseAuthenticationModule`) could be developed to access the LDAP directory in a specialized way
Please see LDAP Properties for a detailed discussion of the property settings for LDAP authentication.

Creating a custom Web server delegation implementation

For custom authentication implementations that rely on Web server modules or filters, `ExternalAuthModule` has been provided as the simplest possible implementation. The `ExernalAuthModule` method parses the request and extracts the CGI variable REMOTE_USER as the user name to authenticate against. The custom Web server delegation implementation would only need to install a module or filter (for example, Kerberos on Apache) on the Web server and configure it to populate the request with the CGI variable REMOTE_USER.

Deploying Custom Implementations

Once the .JAR file containing the .class file for the custom authentication class is built, place the .JAR file in the tomcat auto load directory.

The custom .JAR files are stored in the common classloader of Tomcat. Follow the steps below:

Step 1 Open the directory `config/tomcat/classpath` and create a new text file named `yourinstitution-common.classpath.bb`.

Step 2 Open this text file and enter the full name of the .JAR files that should be included in the classpath.

Step 3 Run `pushconfig.updates` to copy the file to the correct location.

Note: A copy of the jar file should be in `/systemlib`. Additionally, edit `/system/build/bin/launch-tool.bat` (or .sh on Unix) and append the `.jar` files to the `BB_CP` variable. Otherwise, command line tools that bootstrap the core services (for example, `LogRotation` or `PurgeAccumulator`) will not work.

Updating the Collaboration Server

Administrators who use custom authentication may experience issues with the Collaboration server. The collaboration server breaks because of the AccessManager in service-config-collab-server.properties. This issue may be resolved by adding the custom authentication module class to the collaboration server's classpath. Follow the steps below:

Windows

Step 1 Add the classpath to the location of the authentication module classes (either a path to a directory, or a path to a jar file) to the install-nt-services.bat.bb. Administrators may use the wildcard syntax for the Blackboard install directory.

Step 2 Run `PushConfigUpdates`.

Note: When updates to the system are installed, the .bb file may be overwritten. Save a copy of the .bb file before an update and use this copy to replace the .bb template.

UNIX

Step 1 Add the classpath to the location of the authentication module classes (either a path to a directory, or a path to a jar file) to the collabserverctl.sh.bb. Administrators may use the wildcard syntax for the Blackboard install directory.

Step 2 Run `PushConfigUpdates`.

Note: When updates to the system are installed, the .bb file may be overwritten. Save a copy of the .bb file before an update and use this copy to replace the .bb template.

Updating the launch tool script

Administrators should add the custom authentication jar file to the classpath of the launch-tool and launch-app scripts to prevent issues with the PurgeAccumulator tool and other administrative tools.

Windows

Add the classpath to the location of the authentication module classes (either a path to a directory, or a path to a jar file) to the launch-tool.bat. This file is located in <BLACKBOARD_DIR>\system\build\bin.

Note: Save a copy of the files before attempting to add the classpath.

UNIX

Add the classpath to the location of the authentication module classes (either a path to a directory, or a path to a jar file) to the launch-tool.sh. These files are located in <BLACKBOARD_DIR>/system/build/bin.

Note: Save a copy of the files before attempting to add the classpath.

Troubleshooting Custom Implementations

This section explains how to troubleshoot problems that may occur when installing a custom authentication module.

- Examine the tomcat logs in \blackboard\logs\tomcat and \blackboard\logs\bb-services-log.txt files for Java errors that may occur
- Ensure that the class name is correct
- Ensure that all of the properties are correctly defined in the `authentication.properties` file
- Ensure that the authentication module (and all of its dependencies) are in the .JAR file and are in the proper location
- If subclassing `BaseAuthenticationModule` or other implementations, make sure that there are no namespace conflicts that may affect processing

---

Appendix—Updates Since Publication

This document was first published on February 9, 2004.

April 6, 2004: Added Web Server Delegation with Windows 2003 topic.
December 15, 2004: Updated terminology for *Blackboard Community System*.
December 15, 2004: Added information about OpenLDAP to LDAP Configuration Overview topic.

