How to Use NDS eDirectory to Secure Apache Web Server for NetWare

How-To Article

NOVELL APPNOTES

Brad Nicholes
Software Engineer
Core Operating System
bnicholes@novell.com

The open-source Apache Web Server's standard installation includes modules that provide primarily file-based security, independent of the operating system. This AppNote discusses how two additional modules, MOD_NDS and AUTH_LDAP, provide a greater level of security on the NetWare platform. In addition, this article discusses how to use MOD_TLS to set up Secure Sockets Layer (SSL) connections between the client and the server.

Contents:

- Introduction
- Installing the Apache Web Server
- Apache Web Server Modules
- Securing the Apache Web Server
- Bringing It All Together
- Conclusion

Topics	NDS eDirectory, Apache Web Server
Products	Mod_NDS ver 1.0, Mod_TLS ver 1.0, Auth_LDAP ver 1.4.5 (optional)
Audience	network designers, administrators, consultants, integratorsegrators, developers
Level	intermediate
Prerequisite Skills	familiarity with NetWare
Operating System	Apache Web Server ver. 1.3.14a or later
Tools	none
Sample Code	no

Introduction

The Apache Web Server is the most widely used Web server on the Internet. According to the NetCraft Survey, over 60% of the Web servers driving the Internet today are Apache Web Servers. Being an open source project and freely available makes Apache an attractive resource to small business as well as large corporations. Apache is stable, scalable, reliable, and free. In the past five years, Apache has equaled—and in many cases, surpassed—the functionality and stability of commercially-available Web servers.

The standard Apache Web Server installation includes modules that provide a certain level of authentication and authorization functionality. This functionality is primarily file-based and must be applied on a directory or location basis. The problem with this level of security is that it is independent of the security provided by the operating system itself.

This AppNote discusses how two additional modules, MOD_NDS and AUTH_LDAP, provide a much greater level of security by allowing Apache to rely on the operating system rather than reinventing the security wheel. In addition to authentication and authorization, this article also discusses how to use MOD_TLS to set up Secure Sockets Layer (SSL) connections between the client and the server.

Installing the Apache Web Server

Installing the Apache Web Server on NetWare is a straightforward process. Apache for NetWare does not currently include an installation or setup program. For the time being, the installation and setup of Apache on NetWare is a manual process. The process for installing the Apache Web Server on NetWare is described below. (For a more detailed description of how to set up Apache for NetWare, refer to the installation instructions that are included with the software.)

The Apache Web Server binaries for NetWare are available at *http://www.apache.org/dist/binaries/netware/*. Here are the steps to obtain and install the files from this Web site:

1. Download the file Apache_1.3.x_Netware.zip. (A newer version that is required for use with NDS is available with this article; all versions later than 1.3.14a will contain the required patches.) This file decompresses into the standard directory structure for a default installation of Apache.

2. To install the software, simply copy the entire directory structure to the root of any NetWare volume.

The Apache Web Server is configured through the HTTPD.CONF file. The default configuration file can be found in the /Apache/Conf subdirectory. This is a simple text file that can be edited with any text editor. If the Apache software has been installed on the SYS: volume of a NetWare server, there are really no changes necessary to run Apache in a default configuration. If the Apache software was installed to a volume other than the SYS: volume, or if the files were copied to a location other than the root of a volume, some minor changes are required to the HTTPD.CONF file, as described below:

1. Open the HTTPD.CONF file with any text editor.
2. Make sure that all of the directory paths match the location where the Apache files were copied. The default configuration file assumes the primary directory structure is SYS:/APACHE.
3. Make sure that each of these entries corresponds to the location of the installation. Also, Apache on NetWare requires that all directory paths include the volume name.
4. The second change required is assigning a name to the web server itself. This is done by changing the value of the SERVERNAME directive within the HTTPD.CONF file. The value of this directive can either be the server's IP address or its DNS name.

Apache for NetWare contains one platform-specific directive called THREADSTACKSIZE. This is used to determine the stack size allocated for each request service thread started by Apache. The default value for this directive is 65536. This value can be adjusted up or down depending on the type of content being delivered and processing required by the request service threads.

Apache Web Server Modules

One of the strong points of Apache is the ability to extend the Web server through modules. In fact, most of the functionality that exists in the Apache Web server is provided by the inclusion of modules.

There are two types of Apache modules: external and internal (or built-in). An external module is a set of functionality that is wrapped up into a separate executable file. Having the module as a separate file allows the administrator to add, replace, or remove the module as needed. If a newer version of the module becomes available, the administrator simply has to copy the new executable file into the /APACHE/MODULES directory and restart the server.

The second type of module is an internal (or built-in) module. These are modules or sets of functionality that have been compiled into the Apache executable itself when it was built from source. From a source code perspective, there is really no difference between an external or built-in module. The source code for the module looks the same, and the framework for implementing it does not differ. In fact, an external module can be compiled directly into the Apache executable by simply including the source as part of the core Apache code.

When a request is received by the Apache Web Server, it must pass through a series of stages in order for it to be completely handled. The architecture of Apache allows a module to insert itself into any one or more of these stages (see Figure 1).

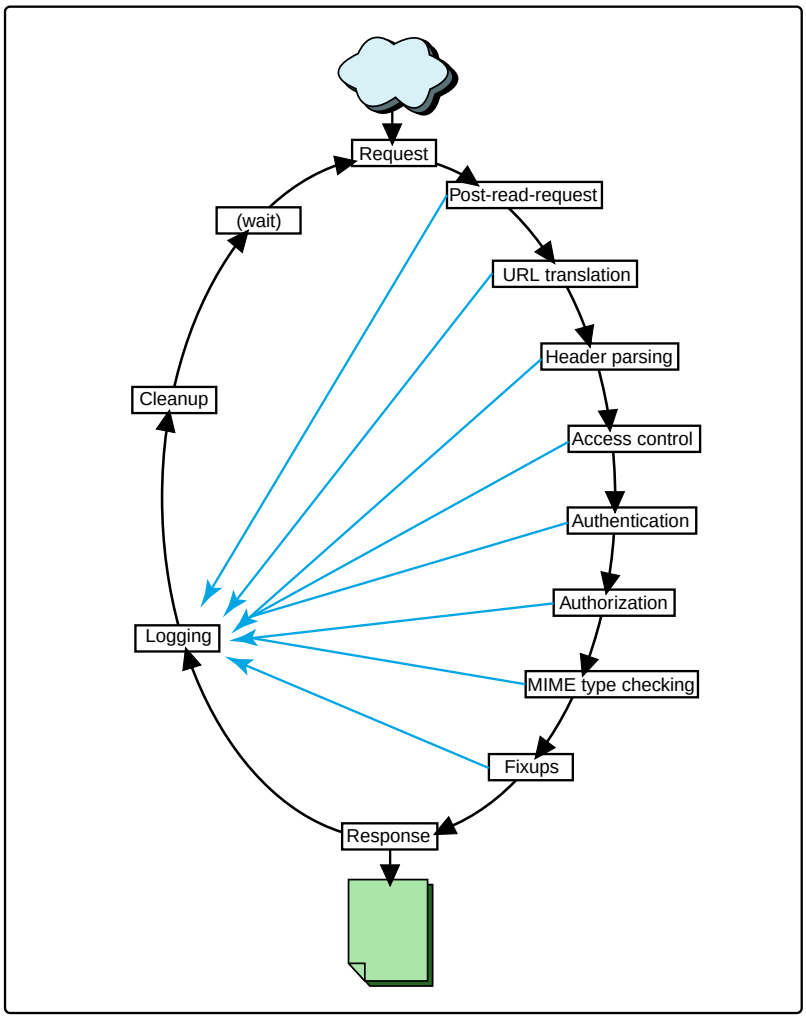

Figure 1: *Apache Web Server stages.*

Three of these stages deal with web server security: Access Control, Authentication, and Authorization. There are currently various Apache modules available that supply handlers for one or more of these stages in order to give the Apache Web Server a certain level of security.

The following sections briefly describe the default level of security that is provided by the standard modules that ship with the Apache Web Server. They will also discuss three additional modules that allow Apache to take advantage of the security provided by the operating system.

Securing the Apache Web Server

Now that Apache is installed and running, how can it be secured? The answer is through installable modules that provide either the necessary security services or access to the security features that are provided by the operating system. There are actually several areas of security that need to be addressed when deploying an Apache Web Server. These areas range from user authentication and secure connections to file access.

Apache Security Services

The Apache Web Server itself has some built in measures for allowing or denying user access as well as file access. Restricting file access is part of the built-in functionality of Apache. By default, the Apache Web Server only allows public access to the contents of the /APACHE/HTDOCS directory and its subdirectories. Allowing access to other locations or restricting access is done by defining <Directory> or <Location> blocks within the HTTPD.CONF file. For example the following <Directory> definition would restrict access to all requests except for those that match the specified domain name:

```
<Directory VOL1:/PrivateFiles>
   order deny,allow
   deny from all
   allow from MyClients.Com
</Directory>
```

While this does enforce some level of access control, it does not provide the granularity that is required to manage requests at a user or session level.

User access can be enforced through the implementation of one of the standard authentication modules that ships with the Apache Web Server. One of these authentication modules is MOD_AUTH_ ANON. This module provides a file based method of controlling user access by matching the user supplied credentials with data that is stored on the server in an encrypted password file. Apache also includes a separate utility called HTPASSWD that is used to create the password file. The following example would allow only the users listed in the file RESTRICTED.PWD, access the files in the specified directory.

```
<Directory VOL1:/PrivateFiles>
    AuthType Basic
    AuthName Restricted_Area
    AuthUserFile VOL1:/UserAccessFiles/Restricted.pwd
    Require valid-user
</Directory>
```

Apache provides the ability to restrict access to the file system, but the level of security is very basic. The functionality that is provided by the authentication and access control modules is cumbersome to use and does not respect the file rights that are built into the operating system. Also, once a user has access to a directory or location, all bets are off. In the above example, there is nothing preventing an individual user from viewing all of the files or web pages contained in that location. What is worse, the authentication mechanism has to be manually applied to each restricted area. If the set of valid users differs from one location to another, a new password file has to be defined for each.

For a detailed description of how to install and use the standard authentication modules or to define <Directory> or <Location> blocks, refer to any one of a number of Apache Web Server administration guides that are available in bookstores or visit *http://www.apache.org*.

Integrating Native Security Services with Apache

The best way to secure a web server is by using the native security inherent in the operating system. NetWare already provides the best security services on the net so why not take advantage of them. Although Apache ships with a set of modules that include a basic set of security features, all the security Apache needs already exist in NetWare. The Apache modules MOD_NDS, MOD_TLS and AUTH_LDAP allow Apache to take advantage of the NetWare security services.

Mod_TLS

One of the advantages of Apache for NetWare is the fact that the software ships with SSL support built in. All other platforms must download, build and install an additional module to obtain SSL support. The reason why Apache for NetWare is able to ship with SSL is because the support for SSL actually exists in socket layer, which is part of the NetWare OS. Since Apache simply relies on the operating system for socket support, both Apache for NetWare as well as MOD_TLS do not contain any encryption code. MOD_TLS, which is the SSL support module, simply tells the socket layer to enable SSL on a given port. It is a very small module that has only one directive.

SecureListen

Syntax	SecureListen <port number> <"Certificate_Name">
Context	server config
Description	Instructs Apache to listen to the specified port on a secure connection.

Mod_NDS

MOD_NDS is an Apache module that was derived from an implementation for Linux that is based on the NCPFS API. The module was originally developed by Philip R. Wilson and is publicly available under the GNU General Public License. The NetWare implementation of MOD_NDS simply replaces the calls to NCPFS with direct calls to the NetWare DSAPI. By doing so, MOD_NDS is also able to force Apache to respect the file rights that are manipulated through Novell NDS eDirectory and enforced by the file system. This level of file access could not be achieved by the Linux version of MOD_NDS. The following is a description of the directives that allow an administrator to integrate the NetWare Directory Services authentication and authorization with the Apache Web Server. For a complete description of the directives, refer to the documentation that accompanies MOD_NDS.

AuthNDSUserFile

Syntax	AuthNDSUserFile <File-Name>
Context	directory, .htaccess
Description	Sets the name of a text file containing a list of user-names that are allowed to authenticate (obsolete on NetWare).

AuthNDSAuthoritative

Syntax	AuthNDSAuthoritative <On (default) | Off>
Context	directory, .htaccess
Description	Determines whether the request is allowed to be passed on to lower level modules for further authentication.

AuthNDSTree

Syntax	AuthNDSAuthoritative <On (default) | Off>
Context	directory, .htaccess
Description	Sets the NDS tree that will be used for user authentication. This is a mandatory directive.

AuthNDSRequirePW

Syntax	AuthNDSRequirePW <On | Off (default)>
Context	directory, .htaccess
Description	Determines if a user name with an empty password will be allowed to access the site.

AuthNDSExpiredURI

Syntax	AuthNDSExpiredURI</Path/To/Expired-Notice.html>
Context	directory, .htaccess
Description	Provides redirection to an alternate page if an expired password is detected.

AuthNDSCacheTimeout

Syntax	AuthNDSCacheTimeout <Number>
Default	AuthNDSCacheTimeout 300
Context	server config
Description	Sets the time-to-live value for entries in the cache (in seconds), or disables the cache entirely (by setting it to zero).

AuthNDSUniqueCNs

Syntax	AuthNDSUniqueCNs <On | Off (default) >
Context	server config
Description	Enables the caching of name->FDN mappings, which prevents the module from having to search for the user's FDN on every request.

AuthNDSContext

Syntax	AuthNDSContext <.Context.To.Search.Conext.To.Search ...>
Context	directory, .htaccess
Description	Sets a search list of contexts for contextless logins.

AuthNDSContextOverride

Syntax	AuthNDSContextOverride <On | Off (Default)>
Context	directory, .htaccess
Description	This directive only applies to AuthNDSContext. If set to 'ON' for a given directory, it causes all search contexts defined in higher-level directories to be ignored.

The following 'require' directives are supported:

require user	<.user1.full.context .user2.full.context .user3.full. context ...> Defines a list of valid users.
require valid-user	Allows access for any valid user name and password.
require context	<.exact.matching.context1 .exact.matching.context2 ...> Allows access for any valid user name and password with a matching context.
require context	</.partially.matching.context1 ...> Allows access for any valid user name and password with a partial matching context.

Auth_LDAP

The AUTH_LDAP module is an authentication module that can be used to authenticate users against any LDAP compliant directory service. This is an open source module and is freely distributed under the Apache Public License. The NetWare version of this module contains no significant changes from any other platform implementation. It supports SSL connections to the LDAP server provided that both the NetWare server and the LDAP server have been configured correctly for SSL. The following is a listing of the directives that are supported by AUTH_LDAP. For a complete description, refer to the documentation that accompanies the module.

AuthLDAPBindDN

Syntax	AuthLDAPBindDN <Distinguished-Name>
Context	directory, .htaccess
Description	An optional DN used to bind to the server when searching for entries. If not provided, AUTH_LDAP will use an anonymous bind.

AuthLDAPBindPassword

Syntax	AuthLDAPBindPassword <Password>
Context	directory, .htaccess
Description	A bind password to use in conjunction with the bind DN.

AuthLDAPAuthoritative

Syntax	AuthLDAPAuthoritative <On (Default) | Off>
Context	directory, .htaccess
Description	Set to 'OFF' if this module should let other authentication modules attempt to authenticate the user, should authentication with this module fail.

AuthLDAPURL

Syntax	AuthLDAPURL <url>
Context	directory, .htaccess
Description	A URL which specifies the LDAP search parameters to use.

AuthLDAPRemoteUserIsDN

Syntax	AuthLDAPRemoteUserIsDN < Off (Default) | On>
Context	directory, .htaccess
Description	If this directive is set to 'ON', the value of the REMOTE_USER environment variable will be set to the full distinguished name of the authenticated user, rather than just the username that was passed by the client

AuthLDAPCertDBPath

Syntax	AuthLDAPCertDBPath </Path/To/Cert7.db/Directory>
Context	server config
Description	Specifies in which directory AUTH_LDAP should look for the certificate authorities database. There should be a file named cert7.db in that directory.

AuthLDAPCacheSize

Syntax	AuthLDAPCacheSize <Size>
Context	server config
Description	Specifies the maximum size of the LDAP search cache.

AuthLDAPCacheTTL

Syntax	AuthLDAPCacheTTL <Time>
Context	server config
Description	Specifies the time (in seconds) that an item in the search cache remains valid.

AuthLDAPOpCacheSize

Syntax	AuthLDAPOpCacheSize <Size>
Context	server config
Description	Specifies the size of the cache AUTH_LDAP uses to cache LDAP operations.

AuthLDAPOpCacheTTL

Syntax	AuthLDAPOpCacheTTL <Time>
Context	server config
Description	Specifies the time (in seconds) that entries in the operation cache remain valid. The default is 600 seconds.

AuthLDAPCacheCompareOps

Syntax	AuthLDAPCacheCompareOps <On (Default) | Off>
Context	server config
Description	If this directive is set to 'ON', AUTH_LDAP will cache any compare operations (these are used to satisfy require user directives).

The documentation that accompanies the AUTH_LDAP module also contains detailed examples of how to implement LDAP authentication for an Apache Web Server. Please refer to the documentation for more information.

Bringing It All Together

So far we have discussed how to install the Apache software as well as described the various modules that can be used to secure the web server. How can Apache take advantage of these modules to provide secure web services? The following example will use MOD_TLS and MOD_NDS to allow a browser to issue a request on a secure socket, and allow the web server to authenticate the user and enforce file rights before the user is able to access the requested web page.

The first thing that needs to be done is to install the modules so that they can be loaded by Apache. This is done by simply copying the file MOD_NDS.NLM to the /APACHE/MODULES directory (MOD_TLS.NLM is installed by default). Next Apache needs to be told that these modules exist and should be loaded. The following two statements added to the HTTPD.CONF will instruct Apache to load the modules:

```
LoadModule tls_module modules/mod_tls.nlm
LoadModule nds_auth_module modules/mod_nds.nlm
```

At this point the modules will be loaded and all of the directives for MOD_TLS and MOD_NDS are available and ready to go. All that is left to be done is to define the file restrictions in the HTTPD.CONF file and determining when authentication is required. One simple example of requiring authentication and enforcing file access rights could be as follows. Any request for a page that resides within the specified directory will prompt the user for authentication credentials and enforce file access rights based on those credentials.

```
SecureListen 443 "SSL CertificateIP"

<Directory sys:/apache/htdocs/private>
    AuthType Basic
    AuthName A_Protected_Place
    AuthNDSTree my_company_tree
    AuthNDSContext .my_sales.staff .my_marketing.staff
    require valid-user
</Directory>
```

When a request is issued for a page that resides in the private directory, the above example will prompt the user for a valid user name and password. If the user were a member of the one of the listed contexts, only a valid user name rather than the fully qualified distinguished name would be required. Once the user name and password are supplied to the web server, the credentials are verified against NDS and the session is logged in as the specified user. From then on all of the file rights that have been granted to the user are enforced by the operating system. If the user does not have file access to the requested page, a *403 Access Denied* response will be returned.

By using MOD_TLS and MOD_NDS, the web server is able to communicate with the browser over a secure connection and allow the user access to only those files that the user has been granted rights to view. MOD_NDS integrates Apache with the world class security services that are provided by the NetWare operating system. Therefore there is no difference between managing web users and file access as managing LAN users. In fact they are one and the same.

Conclusion

Setting up and securing an Apache Web server is literally as easy as installing the Apache software and telling it to use the security services provide by NetWare. Granting or revoking rights to web page is as simple has granting or revoking rights to a file or directory. Since Apache is simply taking advantage of the security services that are provided by the operating system, your web site has the same security that you are used to with NetWare. Managing users, groups and file access for the web is done in the same manner and using the same tools as have always been used to manage your NetWare server. There is nothing new to learn in order to secure your Apache Web Server. Best of all, the Apache Web Server, along with the security access modules, are all free and available on the Internet.

For more information about the Apache Web Server and other open source projects, visit *http://www.apache.org/*.