Apache Security and Auditing

Introduction:

The Apache Web Server is a full-featured, efficient and robust Web Server developed by the Apache Software Foundation [1]. The fact that about 66% [2] of the Websites use Apache Web Server proves that, it has rich and varied features and compares very favorably with other web servers [3]. Apache is easy to install and manage, thanks to the Apache Documentation Project. By being open source, Apache offers greater security and faster response to discovered vulnerabilities than most other web servers.

A website on the Internet, by its very purpose, is easily accessible by everyone. It is therefore of paramount importance to secure the web server. Apache can be secured if the configuration directives are used appropriately. This document focuses on the security controls provided within Apache and how to audit them.

Contents:

Secure Apache Installation

The installation directory for Apache tends to vary from one flavor of Unix to the other. For instance, in Linux the default installation directory is */var/apache*. This can be modified at install time. Throughout this document we refer to this folder as *$ServerRoot*. Also, the actual web site's files – the HTML pages, images, scripts, etc. – might be located in an entirely different folder and this is referred to as *$DocumentRoot*. The main configuration file for Apache is the *httpd.conf* file present in the *$ServerRoot/conf* directory. This file uses, what are called, 'directives' for configuration. Directives are the directions given to the Apache server to determine every type of access to its resources. For the purpose of running the web server, a new user should have been created. Under no circumstances must Apache be run as root – the super user account on Unix. Also, running Apache with an existing low-privileged user account such as *nobody* results in unnecessary access to existing files and directories. A new user *apache* can be created by root as follows:

 # useradd -c "Apache Web Server" -u 8080 -r -s /bin/false -d /home/apache apache

Check that this user has been given read-only access on the Apache server configuration files, files in *$DocumentRoot* and system files, and write permissions on log files. To check under which account apache is running, execute the following command:

 # ps –ef | grep httpd

For a minimum installation you may ensure that at least the following files have been removed:

$ServerRoot/htdocs – Apache documentation
$ServerRoot/icons – default icons used for index pages
$ServerRoot/cgi-bin/sample_scripts

Operating System Security

All critical Apache server files such as executables, configuration files, error and log files should be protected from non-root users. These directories are the primary targets for loading Trojans, corrupting files and other malicious attempts. For an explanation on Unix permissions and how to view them, go to [4]. Permissions on files/folders under *$ServerRoot* should be set as follows:

File or Folder	Owner	Group	Permissions
Conf	root	root	750
Bin	root	root	750
Logs	root	root	755
Httpd	root	root	511

Permissions on *.htaccess:*

The file *.htaccess* contains access control directives for a directory in which it is present, and can exist for any directory. Permissions set in this file override those set at an upper level in the directory hierarchy or at the OS level. A user who does not have access permissions on files, but has write permission on *.htaccess*, can change directives in *.htaccess* to gain write or execute permissions. Hence, this file should be protected from non-root users who are not a part of Apache server management. Better still, *.htaccess* must be completely avoided and all permissions should be set solely in *httpd.conf*. In case a directory needs special permissions, these should be set using a different 'Directory' directive in *httpd.conf*. We shall see more in detail in the section on Directives.

Note: A user can change the name of the *.htaccess* file using the *AccessFileName* directive in *httpd.conf*. So check to ensure that this directive has not modified the file name, but if it has, then you must apply the above section to the new file.

Permissions on *.htpasswd*

The file *.htpasswd* stores usernames and passwords for those users who may need to be authenticated before being given access to restricted areas in the website. The password can be stored as plain text, which should be strictly prohibited, or using an encryption algorithm such as MD5 (default on Windows, Netware, and TPF) or CRYPT (default on Unix), which is highly recommended. If the file is not protected, the attacker can replace a higher-privileged account's hashed password with his own, and start using the hacked account. Preferable permissions are 700.

Other Operating System hardening measures must be followed as well. See [5] and [6].

Patches:

As with any other software, Apache also has had vulnerabilities being discovered in it. So, it is very important that all necessary patches be applied, which gets reflected in the version number of Apache. Apache has two series – the 1.3 and the 2.0 with latest versions as of this writing being 1.3.20 and 2.0.44 respectively. You can determine the Apache version number by issuing the command:

$ServerRoot/bin/httpd -v

Apache HTTP Server Official patches can be downloaded from
http://www.apache.org/dist/httpd/patches/

Directives

Directory

The *Directory* directive is used to enclose a group of directives that will apply only to the named directory and all sub-directories of that directory. The default access control permissions should be set to a very restrictive set. To achieve this, ensure that the following configuration exists in the *httpd.conf* file. This is to prevent access from all (clients) to any directory or file by setting the '*deny from all'* for the '/' directory. Recommended setting –

<Directory />
Options None
AllowOverride None
Order Deny, Allow
Deny from all
</Directory>

Permissions can be set differently for specific directories by modifying the above syntax to read: *<Directory /dir_path>*. This will have to be done to allow access to the web site's content. See the section on Authorization.

Permissions can be set differently for specific directories by modifying the above syntax to read: *<Directory /dir_path>*. This will have to be done to allow access to the web site's content.

Moreover, Apache allows for host-based access control based on wildcard matching with IP addresses or with domain names. For instance, you may choose to deny access from all clients, except those that come from the domain *yourcompany.com*:
Order Deny, Allow
Deny from all
Allow from yourcompany.com
This would only allow access to users from any domain that matches *yourcompany.com* such as *hq.yourcompany.com* or *loc1.yourcompany.com*. You may also choose to allow access only from a particular range of IP addresses, by changing the above *Allow* statement as follows:
Allow from 192.168.0.0/255.255.255.0

Setting *AllowOverride* to *None* stops users from setting their own *.htaccess* files which can be used to override security configurations enabled at Server level. [See section on *.htaccess* permissions above]

Setting *Options* to *None* will disable extra features, which could lead to weaker settings. Risks associated with each of the *Options* features are explained below:

a. ExecCGI
This feature permits execution of CGI scripts at the server. If the input provided by the client is not properly evaluated before using it, the attacker can find out some way or use

known vulnerabilities found in scripting methods and inject his code to get the desired information from the server. Therefore it's recommended to avoid its use unless explicitly required. If required, use very strong parsing mechanisms to prevent any malicious inputs. For more information on CGI security see section CGI Security.

b. FollowSymLinks
This feature allows the server to follow symbolic links [8] in this directory. The permissions given on the symbolic links to the files or directories, overrides those that are present on the actual files or directories. This will allow anyone having permissions on the *symbolic link* to execute, read or write even though he has no permissions on the *actual* file or directory. Strongly recommend against enabling this feature.

c. SymLinksIfOwnerMatch
This feature allows the server to follow symbolic links for only those users who have permissions on the target file or directory. This feature overcomes the situation mentioned above. Even so, you must check the permissions on the links and the files they point to.

d. Indexes
This feature will create a directory listing if the file 'index.html' is not present in the directory that is mapped to URL requested. This will list all the files and directories including those that you do not want the world to see. ***This feature should never be used.***

e. Includes
This feature permits Server Side Includes to be used in the Apache Server. SSIs (Server Side Includes) are directives that are placed in HTML pages, and dynamically evaluated on the server while the pages are being served. For details on SSI see [9]. This feature creates both security and performance problems, since it allows execution of CGI scripts or any executables under the Apache Server permissions. This can be exploited to gain control over the server. If you still want to use SSIs, enable suexec and instead of using option *Includes* use *IncludesNOEXEC*, which would allow SSI to be used but disallows execution of any SSIs, which execute CGI scripts or programs.

f. IncludesNOEXEC
This feature permits SSIs with *#exec* and *#exec cgi* disabled. But the users may still use #include virtual="..." to execute CGI scripts if these scripts are present in directories which use the *ScriptAlias* directive. Therefore, audit all CGI scripts and HTML pages for presence of SSI directives, in case these features are required. See section on *Alias* and *ScriptAlias* below.

g. Multiviews
With this option, the server does an implicit filename pattern match and chooses the closest match that exists. It may enable the server to serve critical content accidentally, so this option should be avoided.

ServerSignature
ServerSignature will add a footer to the error pages giving out server information such as version number, name of server, etc., which must be protected from outsiders. Hence it is recommended to set it to 'off'.
Also change the banner of Apache Server, and set it to something, which does not mention about Apache and its version.. This can be done by setting the following directive in the httpd.conf
ServerTokens ProductOnly.
This will only show 'Apache' in the banner. To completely remove any trace of Apache from the banner, you will have to change the source code and recompile it.

UserDir
If it is necessary to give a user access to his home directory through Apache, then we must use the UserDir directive. With this, a Unix user called *harry* can access his home directory by browsing to: *http://the_web_server/~harry*
In that case disable access to directory in UserDir to all and enable access to only trusted users. Take special care for the permissions on this directory, if possible give read-only access. **Use of this directive is strongly discouraged**.

Alias and ScriptAlias
These directives are used to map URLs to system file paths, so that, content which is not directly under *$DocumentRoot,* can be served up as part of the website. Use of these directives must be avoided as far as possible. ScriptAlias is similar to Alias, except that it marks the target directory as containing only CGI scripts.

Directives to prevent DoS and Buffer Overflows
Apache also provides directives for limiting resource usage such as *RLimitCPU* (to limit CPU usage of child processes launched by the main Apache thread), *RlimitMem* (to limit memory usage by Apache's child processes), and *RlimitNProc* (to limit the child processes that can be spawned by Apache's child processes). The suggested values of these parameters depend upon the hardware resources available and the expected peak usage.

Buffer overflow attacks are those that try to execute code of the attacker's choice by overflowing the program stack. To mitigate the risk of these, we must apply all Apache recommended patches, and in addition may consider setting directives such as *LimitRequestBody* (to limit the total size of the HTTP request body sent by a client), *LimitRequestFields* (to limit the number of HTTP request header fields that will be accepted), *LimitRequestFieldsize* (to limit the size of the HTTP request header sent by the client), and *LimitRequestLine* (to limit the size of the HTTP request line sent by the client).

LogFormat (Auditing):
Ensure that the *LogFormat* directive includes the following critical fields: Remote IP address (%a), Remote host (%h), Remote user (%u), Time (%t), First line of request (%r), Status of the last request (%>s), Number of bytes sent (%b). Study the log files and ensure the following:

1. Keep track of all remote IPs and the first line of request where server status is 403, which imply access to forbidden web pages.
2. Archive the log files at regular intervals, so that the log files do not increase to a very large size. This is usually done by a shell script.

Authorization

Apache allows for host-based access control based on wildcard matching with IP addresses or with domain names. The module mod_access is used to restrict access and it uses the directives Order, Allow and Deny. The combination of these directives can be used to allow or deny access to users. The 'Order' directive sets the default access state and the order in which allow or deny are evaluated.

Example 1
Order allow,deny
allow yoursite.org
deny accounts.yoursite.org

This Order directive will set the default access as deny to everybody and will give access to all users from the domain yoursite.org but will deny access to accounts.yoursite.org. This setting is used when you want to keep your site very restrictive.

Example 2
Order deny,allow
deny yoursite.org
allow accounts.yoursite.org

This Order directive will set the default access to allow everybody and will deny access to all users from the domain *yoursite.org* but will allow access to *accounts.yousite.org*.

You may also choose to allow access only from a particular range of IP addresses, by changing the above *Allow* statement as follows:
Allow from 192.168.0.0/255.255.255.0

Note: Access rights are enforced in the descending order as given below, with the Location (used for access control if documents are created on the fly) directive taking the precedence. Therefore even if the Directory directive is prohibiting access to certain files, Location may give access to these files.

Directory
DirectoryMatch
Files/FilesMatch
Location/LocationMatch

Authentication

Following modules are used for user authentication: mod_auth, mod_auth_dbm, etc. The following directives are used for passing the authorization information to these modules:

1. AuthGroupFile: This directive represents the text file, which contains Apache groups and the associated users. The syntax is
group_name: user1,user2,user3

2. AuthUserFile: This directive represents the text file, which contains Apache users and their encrypted password. The syntax is
user1:password1
user2:password2

.htaccess is the default text file used for storing this information. Ensure that this file is stored outside the $DocumentRoot, i.e. the AuthUserFile directive should not contain $DocumentRoot as its parent directory.

3. AuthName: This directive sets the authorization realm (Apache refers to protected areas as realms, which contain a set of critical documents – this is usually a link or actual file location) for the client so that the client knows which user-password pair a client has to sent to the server.

4. Authtype: This directive indicates how the user-password pair is transferred over the network. Apache Server provides two types of Authentication: Basic and Digest. The Basic type sends the password from client to the browser using Base64 encryption, which is as good as being unencrypted. This authentication type will fail to protect the user password if somebody sniffs the traffic in-between. Digest type is safer to use as it provides MD5 hashing functionality. But all the browsers do not support Digest type.

5. Require: This directive lists the users or groups are allowed access to the protected documents. Check that all entries here are users who are privileged to view this information, and that there are no unauthorized users here. The possible settings can be:
Require user user1,user2
Require group group1,group2
Require valid-user

The last setting allows access to all the users from the authorization file indicated by AuthUserFile or AuthGroupFile.

For example:
AuthType Basic
AuthName "Restricted Directory"
AuthUserFile /web/users
AuthGroupFile /web/groups
Require user sam,foo

This setting of directives indicates authorization type as basic. The realm (Protected area) is known as "Restricted Directory". The Authorization files containing user-password pairs and group-users pair are stored at "/web/users" and "/web/groups" respectively. The Require directive indicates that user sam and foo are authorized to access this directory or location.

Satisfy: This directive is used if you want to mix up the access and authentication directives. This directive can be set to ‘all’, which require users to satisfy both types of directives and ‘any’, which requires the user to satisfy at least one type of directive. Say if you want to allow only two users,user1 & user2 from yoursite.org to access restricted data, then we would set the directives as follows:

Order allow,deny
allow yoursite.org
deny accounts.yoursite.org
AuthName "Restricted Access"
AuthType Basic
AuthUserFile /var/www/html/imp-docs
Require user1,user2
Satisfy all

CGI Security

The security holes generated by the CGI scripts can be neutralized to some extent by using CGI wrappers. CGI Wrappers are basically programs that get called whenever a CGI script is to be executed and they run the script under a restricted environment. Enabling CGI Wrappers requires a good deal of administration and configuration settings. Apache comes with its own wrappers script 'suEXEC', which is usually found in */usr/sbin*. It allows us to configure parameters that will control execution of scripts by Apache, such as the minimum User ID and Group ID, the user account, and the (safer) PATH with which CGI and other scripts will be executed. This mitigates the risk from scripts to a very large extent.

To check if suEXEC is used by Apache server issue the following command which will produced the list of compiled modules. If suEXEC is enabled it will display a line "suexec: enabled".
unix#>usr/sbin/httpd -l

You can view the suEXEC configuration by:
unix#>/usr/sbin/suexec –V

Following are the default settings for suEXEC.
-D AP_DOC_ROOT="/var/www"
-D AP_GID_MIN=500
-D AP_HTTPD_USER="apache"
-D AP_LOG_EXEC="/var/log/httpd/suexec.log"
-D AP_SAFE_PATH="/usr/local/bin:/usr/bin:/bin"
-D AP_UID_MIN=500
-D AP_USERDIR_SUFFIX="public_html"

AP_DOC_ROOT:
This is the same as DocumentRoot set in *httpd.conf*

AP_GID_MIN:
This is the lowest GID to which suEXEC can switch when executing CGI scripts.

AP_HTTPD_USER:
This represents the account under which Apache httpd service runs.

AP_LOG_EXEC:
This represents the file name under which suEXEC transactions and errors are logged.

AP_SAFE_PATH:
Apache CGI executables use the Unix environment variable 'PATH'. This PATH variable may contain entries like '.' which could launch a Trojan or such malicious script. SuEXEC provides a safe PATH environment using AP_SAFE_PATH variable. Review it for suspicious entries.

AP_UID_MIN:
This represents the lowest UID to which suEXEC can switch when executing CGI scripts.

Other CGI wrappers such as *sbox* and *CGI Wrap* are also available. For more information on CGI wrappers see [7].

In any case, it is strongly recommended that if CGI is enabled, you must audit the CGI scripts themselves. All default scripts must be removed. All customized scripts must check user input and allow only safe characters while eliminating all else. For instance, a CGI script might accept only alphabets and numerals while rejecting everything else. This is to protect against CGI script manipulation by the user, who may supply meta-characters such as ‘”/\?~`!@#$%^&*. These characters are interpreted by the Unix shell and might cause the CGI script to execute system commands.

Modules

Adding modules to Apache during its execution can enhance its functionality. We must ensure that only modules that are used for running and securing the Apache server should be installed and all others should be unloaded. See the list of modules in *httpd.conf* with the LoadModule directive. **For details on each module see:** **http://httpd.apache.org/docs-2.0/mod** Below is a list of modules, which should be disabled unless explicitly required:

mod_access	mod_auth_any	Libperl
mod_userdir	mod_auth_mysql	mod_php
mod_actions	mod_auth_pgsql	Libdav
mod_alias	mod_cgi	mod_roaming
mod_auth	mod_cgid	mod_put
mod_auth_anon	mod_env	mod_python
mod_auth_dbm	mod_proxy	

Other Security Issues

Chroot

The main problem with the compromise of an Apache Web Server is that it more often than not leads to a compromise of the underlying operating system, and from there, onwards to other systems in the network as well. In spite of the security measures that we put into place for Apache, we must prepare for the possibility of a successful intrusion. In such a case, we would wish to prevent the attack on Apache escalating to an attack on the OS and the rest of our network. To do this, we can run Apache in what is called a 'chroot jail'. This implies a configuration of the system wherein the Apache server believes its *$ServerRoot* to be the root directory '/' of the OS. This is done by copying necessary files from the system to respective directories under $ServerRoot. For instance, to run Apache properly we need certain system libraries, which must be copied from the original location to a */lib* directory under $ServerRoot. Similarly we will need to copy files such as */etc/passwd* and */etc/group* from their original locations to *$ServerRoot/etc*. Moreover, configuration files will need to be modified in order to use this new file system structure. In such a scenario, even if Apache does get hacked into, the hacker cannot do anything other than move around in the *chroot* jail. If the *chroot* has been configured properly, he has no means to 'break' out of the jail as he does not have access to system commands, scripts, configuration files, etc. See [9] for details on setting up a 'chroot-jail' environment for Apache.

Secure Server

A Secure Server is one that is accessed using the Secure HTTP protocol (https://domainname/) and encrypts all data between the browser and the server. Apache provides extensive support for HTTPS through the *mod_ssl* module. For this you need to create a certificate and get it signed by a root CA such as Verisign or Thawte or self-sign it. The details of this are beyond the scope of this article and the reader is referred to [11] for a more thorough discussion

Auditing tools:
The auditor may use both host-based and network-based vulnerability assessment tools to aid in his work. Host-based tools are:
Tripwire – for ensuring integrity of critical files http://www.tripwire.com/
COPS – to ensure Unix security http://dan.drydog.com/cops/
External auditing tools are:
Nmap – for checking open ports http://www.insecure.org/nmap/
Nessus– for vulnerability assessment http://www.nessus.org

References:

1. Apache Software Foundation http://www.apache.org
2. Netcraft Web Server Survey http://www.netcraft.com/survey/
3. Web Compare: http://webcompare.internet.com/
4. Unix Permissions
   http://www.acm.uiuc.edu/webmonkeys/html_workshop/unix.html
5. Practical Unix and Internet Security, Simson Garfinkel and Gene Spafford.
6. The Unix Auditor's Practical Handbook,
   http://www.nii.co.in/research/handbook.html
7. CGI Wrappers and suexec: http://httpd.apache.org/docs-2.0/suexec.html
8. Unix Symbolic Links: A link in Unix is somewhat like a shortcut in Windows.
   For a detailed explanation see: http://www.ncl.ac.uk/ucs/unix/ln.html
9. Apache and SSI http://httpd.apache.org/docs-2.0/howto/ssi.html
10. How to 'chroot' an Apache tree with Linux and Solaris:
    http://penguin.epfl.ch/chroot.html
11. Apache and SSL http://httpd.apache.org/docs-2.0/ssl/ssl_faq.html
12. Professional Apache Security, Wrox Press,
    http://www.wrox.com/books/1861007760.htm