

# Locking Down Apache

# Locking Down Apache

---

Jay Beale  
Senior Security Consultant, Intelguardians

[jay@bastille-linux.org](mailto:jay@bastille-linux.org)

(Def Con 12)

# Contents

---

- Configuration Modification
- Chroot-ing Apache
- Removing Modules
- Using Security-focused Apache Modules

# httpd.conf

---

We harden recent releases of Apache entirely through the httpd.conf file.

- /etc/apache/httpd.conf (Solaris)
- /etc/httpd/conf/httpd.conf (Linux, recent)
- /usr/local/apache/etc/httpd.conf  
(compiled w/ --prefix=/usr/local/apache)

Let's look at this file's structure.

# Apache Configuration File

---

Apache's configuration file starts with a number of generic options and then begins to set options based on parts of the web space in `<Directory>` blocks.

```
<Directory />
```

```
Order Allow, Deny
```

```
Deny from All
```

```
</Directory>
```

# Apache Config file

The Apache configuration file has three parts.  
The first part applies to the entire server as a whole, virtual servers and all.

```
### Section 1: Global Environment  
ServerRoot "/usr/local/apache"
```

```
#Listen 12.34.56.78:80
```

```
Listen 80
```

```
LoadModule access_module modules/mod_access.so
```

```
LoadModule auth_module modules/mod_auth.so
```

```
LoadModule auth_anon_module modules/mod_auth_anon.s
```

```
LoadModule alias_module modules/mod_alias.so
```

```
LoadModule rewrite_module modules/mod_rewrite.so
```

# Apache Config File Section 2 (Slide 1/3)

Section 2 applies to the main non-virtual server. It also sets defaults for the virtual servers that they can override.

```
### Section 2: 'Main' server configuration
```

```
User nobody
```

```
Group #-1
```

```
ServerAdmin you@example.com
```

```
#ServerName www.example.com:80
```

```
DocumentRoot "/usr/local/apache/htdocs"
```

```
<Directory />
```

```
    Options FollowSymLinks
```

```
    AllowOverride None
```

```
</Directory>
```

# Apache Config File Section 2 (Slide 2/3)

# This should be changed to whatever you set  
DocumentRoot to.

```
<Directory "/usr/local/apache/htdocs">  
  Options Indexes FollowSymLinks  
  AllowOverride None  
  Order allow,deny  
  Allow from all  
</Directory>
```

UserDir public\_html

```
#<Directory /home/*/public_html>  
#  AllowOverride FileInfo AuthConfig Limit Indexes  
#  Options MultiViews Indexes SymLinksIfOwnerMatch  
IncludesNoExec  
#</Directory>
```

# Apache Config File Section 2 (Slide 3/3)

```
AccessFileName .htaccess
```

```
<Files ~ "^\.ht">
```

```
    Order allow,deny
```

```
    Deny from all
```

```
</Files>
```

```
ServerTokens Full
```

```
ServerSignature On
```

```
ScriptAlias /cgi-bin/ "/usr/local/apache/cgi-bin/"
```

```
<Directory "/usr/local/apache/cgi-bin">
```

```
    AllowOverride None
```

```
    Options None
```

```
    Order allow,deny
```

```
    Allow from all
```

```
</Directory>
```

```
IndexOptions FancyIndexing VersionSort
```

# Apache Config File Section 3: Virtual Hosting

```
#NameVirtualHost *:80
```

```
#<VirtualHost *:80>
```

```
#   ServerAdmin webmaster@dummy-host.example.com
```

```
#   DocumentRoot /www/docs/dummy-host.example.com
```

```
#   ServerName dummy-host.example.com
```

```
#   ErrorLog logs/dummy-host.example.com-error_log
```

```
#   CustomLog logs/dummy-host.example.com-access_log  
common
```

```
#</VirtualHost>
```

# Web Server Tightening

Do You Need a World-Accessible Web Server?

Modify `httpd.conf` to either make the web server listen only on the loopback interface:

```
Listen 127.0.0.1:80
```

or modify it to move the port:

```
Listen 192.168.1.4:26457
```

This may make the server easier to firewall if you have less than granular control over what ports are allowed to what hosts.

# Web Server Tightening

Of course, using Listen to change the port only doesn't necessarily cut off access to clients in that granular a fashion. For internal hosts, we've only hidden the server, not cut off access.

We can restrict the server to specific hosts:

```
<Directory /usr/local/apache/htdocs>  
order allow,deny  
allow from 192.168.1.0/24  
allow from 10.0.0.0/8  
deny from all  
</Directory>
```

# Order / Allow and Deny

Order statements define whether the allow statements will be default deny or default allow.

The second of the two policies in the statement is the default policy.

# Access Control

We can even impose greater restrictions on who can access our site.

```
<Directory /foo/public_html>  
<Files foo-secret.html foo-extra-secret.html>  
  AuthName "Foo for Thought"  
  AuthType Digest  
  AuthDigestFile /foo/.htpasswd  
  Require valid-user  
</Files>  
</Directory>
```

The `<Files>` block wasn't necessary -- if we left it out, the access control applies to the entire directory.

# Access Control continued

---

Create `.htpasswd` using the `htpasswd` command to md5 passwords.

```
htpasswd -c -m /foo/.htpasswd jay jaypasswd
```

After creating the file, all future users are added without the `-c` option:

```
htpasswd -m /foo/.htpasswd seconduser secondpass
```

# Walking the Filesystem

---

Many configurations of Apache allows the webserver to serve any files readable by its user from the entire server filesystem, as soon as someone with write access to `/usr/local/apache/htdocs` does this:

```
$ ln -s /usr/local/apache/htdocs/my_link
```

A remote attacker can now see any file that the web server user can see. Not good.

Spammers think: `http://localhost/etc/passwd`

# Default Deny on Server Files

We tell the server to only serve files we intend to be served.

```
<Directory />
```

```
    Order allow,deny
```

```
    Deny from all
```

```
</Directory>
```

```
<Directory /var/www/html>
```

```
    Order Deny,Allow
```

```
    Allow from all
```

```
</Directory>
```

```
<Directory /usr/users/*/public_html>
```

```
    Order Deny,Allow
```

# Options

Options <list> lists behavior allowed in the given <Directory /foo> block.

- All - All except multiviews
- ExecCGI - Execution of CGI scripts permitted
- FollowSymLinks - The server follows symlinks.
- SymLinksIfOwnerMatch - The server only follows symbolic links if the directory owner matches the symlink target.
- Includes - Allow SSI's, including exec
- IncludesNoExec - Allow SSI's, but no exec
- Indexes - Lists files in a directory when index.foo is missing
- MultiViews - Fuzzy search for content

# Cutting Off Symlink Misconfigurations

---

Remove "FollowSymLinks" from Options statements, especially from the <Directory /> block.

If we need symlinks, use its better replacement:

SymLinksIfOwnerMatch

Alternatively, use mod\_rewrite to rewrite URLs to achieve the same effect as symlinks.

# Server Side Includes

Server side includes (SSI) can be dangerous.

They allow web developers to include other files, but also to execute arbitrary commands with the user context of the web server's user. Here are the SSI possibilities:

config    configure output formats

echo    print variables

exec    execute external programs

fsize    print size of a file

flastmod    print last modification time of a file

include    include a file

printenv    print all available variables

set    set a value of a variable

# Removing Server Side Includes

To remove the exec functionality, we can replace the "Options Includes" lines with:

```
Options IncludesNOEXEC
```

or just remove Includes altogether.

# Removing Indexing

---

The "Indexes" option tells the server to show us a list of files whenever index.html is missing from a directory. This has two weaknesses:

- 1) might allow our "walking the filesystem" attack
- 2) allows an attacker to potentially find and read files he shouldn't.

We can deactivate this by removing "Indexes" from the Option lines and by creating an index.html file for each directory.

# Protecting the .htaccess and .htpasswd files

To protect these files against dictionary attacks, make sure to not allow reads of the .ht\* files:

```
<Files ~ "\.ht">  
Order allow,deny  
Deny from all  
</Files>
```

# Directory-specific password Authentication: htaccess

There are two ways to define the authentication and other behavior for a given directory. First, and best, you can place configuration in a `<Directory>` block in the global configuration file.

Alternatively, you can apply the same directives in a `.htaccess` file in a given directory.

```
$ cat /home/student/www/.htaccess
```

```
Option Indexes
```

```
AuthName "student site"
```

```
AuthType Digest
```

```
AuthDigestFile /home/student/www/.htpasswd
```

```
Require valid-user
```

# Blocking .htaccess Overrides

An .htaccess file can override anything in the Options statements, how the server handles files, or even what hosts are allowed to connect to the server. The latter is achieved by specifying directory-specific Allow and Deny statements.

We can make overrides very specific, via the AllowOverride statement:

Set:

```
AllowOverride AuthConfig
```

in the `<Directory />` block.

# AllowOverride Options

- AuthConfig - Allow use of the authorization directives
- FileInfo - Allow use of the directives controlling document types
- Indexes - Allow use of the directives controlling directory indexing
- Limit - Allow use of the directives controlling host access (Allow, Deny and Order)
- Options - Regular Options statements

# Making Apache Offer Less Config Information

---

Hide the version number from attackers to make version scanners and potentially worms fail.

ServerSignature Off

Hide the list of modules and other status information from an attacker:

ServerTokens Prod

# Create Error Pages

Replacing the standard error messages with custom pages might help foil automated scanners, though this doesn't change the error code in the server's HTTP response, which is what most read.

ErrorDocument 500 /error-docs/error.html

ErrorDocument 404 /error-docs/error.html

# Remove Unused Methods

<Limit method1 method2 ... methodN>

Available methods:

GET POST PUT DELETE CONNECT OPTIONS PATCH  
PROPFIND

PROPPATCH MKCOL COPY MOVE LOCK UNLOCK

Methods are defined in section 9 of RFC2616  
(<http://www.ietf.org/rfc/rfc2616.txt>)

# Remove Unused Methods

- Remove WebDAV methods

```
<Limit PROPFIND PROPPATCH LOCK UNLOCK MOVE  
COPY MKCOL PUT DELETE>
```

```
Order allow,deny
```

```
Deny from all
```

```
</Limit>
```

- Removing TRACE would help, but isn't available here.  
We'll have to do this through mod\_rewrite.

More about WebDAV:

- <http://www.apacheweek.com/issues/00-12-29>)
- <http://www.ietf.org/rfc/rfc2518.txt>

# Using Limit to remove all but desired methods

```
<Directory /usr/local/apache/htdocs>  
  <Limit GET POST OPTIONS>  
    Order allow,deny  
    Allow from all  
  </Limit>  
  <LimitExcept GET POST OPTIONS>  
    Order deny,allow  
    Deny from all  
  </LimitExcept>  
</Directory>
```

# Dynamic content and CGI

---

Dynamic content authors often don't understand the HTTP protocol or have a strong grounding in security principles.

Try to read or blackbox audit your dynamic content.

# Blackbox Audit

There are some good blackbox audit programs.

- Immunity's Spike Proxy:

<http://www.immunitysec.com/resources-freesoftware.shtml>

- Paros:

<http://www.proofsecure.com/index.shtml>

- @Stake's WebProxy

<http://www.atstake.com/products/webproxy/>

In essence, allows you to modify every part of your client's interaction with the webserver.

# Coping with CGI's or other Dynamic Content

---

Either set specific directories that can run CGI scripts, as is now the default in Apache, or disable them by removing scriptalias statements and ExecCGI options in Options statements.

Forcing CGI's into a specific directory:

```
ScriptAlias /cgi-bin /var/www/cgi-bin
```

(<http://httpd.apache.org/docs/howto/cgihtml#scriptalias>)

# SuEXEC

---

Think about using suEXEC or cgiwrap!

Normally CGI scripts run as the same user as the webserver. Using suEXEC, they run as a particular user, which lets you contain damage.

Add `--enable-suexec` to your `./configure` statement, recompile and restart the server. If the server finds the suexec binary at the right place, `/usr/local/apache/sbin/suexec` in our example, it just starts using it.

<http://httpd.apache.org/docs/suexec.html>

# Remove Default Content

---

In many web server vulnerabilities, example CGI scripts caused the vulnerability.

Additionally, attackers often scan for specific Web server types by looking for its default content. In the case of Apache, that might be the manual or the icons directory.

# Removing Default Content

Remove:

<DocumentRoot>/icons

<DocumentRoot>/manual

Make sure there are no CGI scripts in the CGI directory. On Fedora, that's:

/var/www/cgi-bin

# Chrooting Apache

---

We can contain the damage that a compromised Apache server can do to a system by locking it into a jail directory.

The /jail directory becomes Apache's new root filesystem. This directory must contain every file on the system that Apache will need.

# Creating the Chroot - Compiling Apache

It's easier to re-compile Apache to put all of its files in one place than to use a distro's compile, since the distros spread the files around the filesystem.

```
$ ./configure --enable-mods-shared=most  
--prefix=/usr/local/apache --enable-suexec
```

```
$ make
```

# Chrooting Apache - Tools

When you chroot anything, your two greatest allies are strace and ldd.

- strace

runs a child process, displaying all system calls

- ldd

lists the dynamically loaded libraries a program requires

# Chrooting Apache

---

Create the primary directory structure:

```
umask 022
```

```
mkdir /jail && cd /jail
```

```
mkdir -p dev etc lib tmp usr usr/local usr/bin usr/lib
```

```
chmod 1777 /tmp
```

# Create Devices

---

```
mknod -m 666 dev/null c 1 3
```

```
mknod -m 666 dev/random c 1 8
```

# Create a passwd file

Look at what user Apache uses and create a passwd file.

```
egrep '^user:' /etc/passwd >etc/passwd
```

```
egrep '^user:' /etc/shadow >etc/shadow
```

```
egrep '^group:' /etc/group >etc/group
```

# Idd on /bin/bash

Idd tells us what dynamically loaded libraries a program uses.

```
# Idd /bin/bash
libtermcap.so.2 => /lib/libtermcap.so.2 (0xb75d2000)
libdl.so.2 => /lib/libdl.so.2 (0xb75cf000)
libc.so.6 => /lib/tls/libc.so.6 (0xb7498000)
/lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0xb75eb000)
```

# Look for necessary libraries for all Apache programs

```
# ldd /usr/local/apache/bin/* | sort | uniq | grep \=\  
libapr-0.so.0 => /usr/local/apache/lib/libapr-0.so.0  
(0xb74bb000)  
libaprutil-0.so.0 => /usr/local/apache/lib/libaprutil-0.so.0  
(0xb75d6000)  
libcrypt.so.1 => /lib/libcrypt.so.1 (0xb7457000)  
libc.so.6 => /lib/tls/libc.so.6 (0xb72f8000)  
libdb-4.1.so => /lib/libdb-4.1.so (0xb74f9000)  
libdl.so.2 => /lib/libdl.so.2 (0xb742f000)  
libexpat.so.0 => /usr/lib/libexpat.so.0 (0xb74d9000)  
libgdbm.so.2 => /usr/lib/libgdbm.so.2 (0xb75bb000)  
/lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0xb75eb000)  
libm.so.6 => /lib/tls/libm.so.6 (0xb7484000)  
libnsl.so.1 => /lib/libnsl.so.1 (0xb7442000)
```

# Copy in Libraries

```
cp -p /usr/local/apache/lib/libapr-0.so.0  
/usr/local/apache/lib/libaprutil-0.so.0 /jail/usr/local/apache/lib
```

```
cp -r /lib/libcrypt.so.1 /lib/libdb-4.1.so /lib/libdl.so.2  
/lib/ld-linux.so.2 /lib/libnsl.so.1 /jail/lib
```

```
cp -r /lib/tls/libc.so.6 /lib/tls/libm.so.6 /lib/tls/libpthread.so.0  
/lib/tls/librt.so.1 /jail/tls/lib
```

```
cp -r /usr/lib/libexpat.so.0 /usr/lib/libgdbm.so.2 /jail/usr/lib
```

# Copy in Resolution Libraries and Files

```
cp -p /lib/nss_files.so.1 /lib/libnss_files.so.2  
/lib/libnss_dns.so.1 /lib/libnss_dns.so.2 lib
```

```
cat >/jail/etc/nsswitch.conf
```

```
passwd: files
```

```
shadow: files
```

```
group: files
```

```
hosts: files dns
```

```
echo "127.0.0.1 localhost.localdomain localhost"  
>/jail/etc/hosts
```

# Time Zone Files

---

Either copy all time zone files in:

```
mkdir -p /jail/usr/share
```

```
cp -r /usr/share/zoneinfo /jail/usr/share/
```

or a single file:

```
mkdir -p /jail/usr/share/zoneinfo/America
```

```
cp -r /usr/share/zoneinfo/America/Chicago  
/jail/usr/share/zoneinfo/America
```

# Putting Apache into the Jail

```
cp -pr /usr/local/apache2 /jail/usr/local
```

Change HTTPD variable in  
/jail/usr/local/apache/bin/apachectl

from:

```
'/usr/local/apache/bin/httpd'
```

to

```
'chroot /jail /usr/local/apache/bin/httpd'
```

Now start the daemon with /usr/local/apache/bin/apachectl  
start

# mod\_rewrite

---

The apache module mod\_rewrite was created as a general swiss-army knife for rewriting incoming requests. It can be used as a security tool, though.

mod\_rewrite is very general, but its simplest use looks like this:

```
RewriteEngine on
```

```
RewriteRule ^/bad-url$ /index.html
```

# Using mod\_rewrite to protect .htaccess files

We can use modrewrite to make a particular request fail:

```
RewriteEngine on  
RewriteRule ^.htaccess - [F]
```

This rewrites the URL as a -, but also causes the request to fail.

# Removing TRACE functionality

We place the following in the general config file.

```
RewriteEngine on
```

```
RewriteCondition %{REQUEST_METHOD} ^TRACE
```

```
RewriteRule .* [F]
```

# mod\_rewrite wrapup

mod\_rewrite is extremely flexible and extremely powerful.

The great thing about mod\_rewrite is it gives you all the configurability and flexibility of Sendmail. The downside to mod\_rewrite is that it gives you all the configurability and flexibility of Sendmail.'

- Brian Behlendorf, Apache Group

- You can pass the rewrite to an external program.
- You can rewrite a given string with a randomly entry in a replacement table (originally used for load-balancing...)
- You can set a cookie on the client's browser.

# Advanced Web Server Security: Remove modules!

---

We can remove modules that we're not using. Not all vulnerabilities aren't in the core Apache code. Much is in the modules.

<http://httpd.apache.org/docs-2.0/mod/>

# Figuring Out Which Modules to Remove

---

This page allows you to look at a module and see what configuration directives it provides:

<http://httpd.apache.org/docs-2.0/mod/>

This page allows you to look at configuration directives and see what modules provide them:

<http://httpd.apache.org/docs-2.0/mod/directives.html>

Let's look at the default module list in Apache on Red Hat 9.

# Default Module List in Apache in RH9

- mod\_access.so

Provides access control based on client hostname, IP address, or other characteristics of the client request.

- mod\_auth.so

User authentication using text files

- mod\_auth\_anon.so

Allows "anonymous" user access to authenticated areas

- mod\_auth\_dbm.so

Provides for user authentication using DBM files

- mod\_auth\_digest.so

User authentication using MD5 Digest Authentication.

# Default Module List in Apache in RH9

---

- mod\_include.so

Server-parsed html documents (Server Side Includes)

- mod\_log\_config.so

Configurable-logging of the requests made to the server.

- mod\_env.so

Modifies the environment which is passed to CGI scripts and SSI pages.

- mod\_mime\_magic.so

Determines the MIME type of a file by looking at a few bytes of its contents.

- mod\_cern\_meta.so

Older "CERN" header modification method for setting expires or other custom headers.

# Default Module List in Apache in RH9

---

- mod\_expires.so

Generation of Expires HTTP headers according to config file

- mod\_headers.so

Customization of HTTP request and response headers

- mod\_usertrack.so

"Clickstream" cookie-based logging of individual user activity

- mod\_unique\_id.so

Provides an environment variable with a unique identifier for each request, potentially used in webapps.

# Default Module List in Apache in RH9

---

- mod\_setenvif.so

Allows the setting of environment variables based on characteristics of the request

- mod\_mime.so

Associates the requested filename's extensions with the file's behavior (handlers and filters) and content (mime-type, language, character set and encoding)

- mod\_dav.so

Distributed Authoring and Versioning (WebDAV) functionality ([www.webdav.org](http://www.webdav.org))

# Default Module List in Apache in RH9

---

- mod\_status.so

Provides information on server activity and performance

- mod\_autoindex.so

Generates directory indexes, automatically, similar to the Unix ls command or the Win32 dir shell command  
Requires mod\_dir.so.

- mod\_asis.so

Sends files that contain their own HTTP headers

- mod\_info.so

Provides a comprehensive overview of the server configuration

# Default Module List in Apache in RH9

---

- mod\_cgi.so

Execution of CGI scripts

- mod\_dav\_fs.so

filesystem provider for mod\_dav

- mod\_vhost\_alias.so

Provides for virtual hosting

- mod\_negotiation.so

Provides for content negotiation (best representation based on browser-supplied media type, languages, character set and encoding)

# Default Module List in Apache in RH9

---

- mod\_dir.so

Provides for "trailing slash" redirects and serving directory index files

Requires: mod\_autoindex.so

- mod\_imap.so

Server-side imagemap processing

- mod\_actions.so

This module provides for executing CGI scripts based on media type or request method.

- mod\_speling.so (not a typo!)

Attempts to correct mistaken URLs that users might have entered by ignoring

capitalization and by allowing up to one misspelling

# Default Module List in Apache in RH9

---

- mod\_userdir.so

User-specific directories

- mod\_alias.so

Provides for mapping different parts of the host filesystem in the document tree

and for URL redirection. Required for CGI ScriptAlias directive.

- mod\_rewrite.so

Provides a rule-based rewriting engine to rewrite requested URLs on the fly.

- mod\_proxy.so

HTTP/1.1 proxy/gateway server

# Default Module List in Apache in RH9

---

- mod\_proxy\_ftp.so

FTP support module for mod\_proxy

- mod\_proxy\_http.so

HTTP support module for mod\_proxy

- mod\_proxy\_connect.so

mod\_proxy extension for CONNECT request handling

# Figuring Out Which Modules to Remove

---

This page allows you to look at a module and see what configuration directives it provides:

<http://httpd.apache.org/docs-2.0/mod/>

This page allows you to look at configuration directives and see what modules provide them:

<http://httpd.apache.org/docs-2.0/mod/directives.html>

# Apache Security Modules

---

There are new modules being written specifically to increase the security of the Apache server.

mod\_security

mod\_paramguard

mod\_hackprotect

mod\_hackdetect

mod\_dosevasive

mod\_bandwidth

# mod\_security

This module looks for predefined attack signatures in the client's requests. It can block or simply alert on those requests. This is stronger than mod\_rewrite primarily because it can detect and block data in any part of the request, not simply in the GET URI.

It also performs canonicalization features.

□ [www.modsecurity.org](http://www.modsecurity.org)

# mod\_security filtering

- Prevent SQL injection in a cookie:

```
SecFilterSelective COOKIE_sessionid "!^([0-9]{1,9})$"
```

- Reject Googlebot

```
SecFilter HTTP_USER_AGENT "Google"  
nolog,redirect:http://www.google.com
```

- Reject Javascript in all variables except varfoo

```
SecFilter "ARGS|!ARG_varfoo" "<[:space:]*script"
```

- Reject Specific Command-execution

```
SecFilter /etc/password
```

```
SecFilter /bin/l
```

# mod\_security filtering (continued)

- Reject Directory Traversal

```
SecFilter "\.\./"
```

- Reject Cross-Site Scripting (XSS) Attacks

```
SecFilter "<[[:space:]]*script"
```

- Reject SQL injection attacks

```
SecFilter "delete[[:space:]]+from"
```

```
SecFilter "insert[[:space:]]+into"
```

```
SecFilter "select.+from"
```

# mod\_security Canonicalization Features

- Remove multiple forward slashes (//).
- Remove self-referenced directories (./).
- Treat \ and / equally (on Windows only).
- Perform URL decoding.
- Replace null bytes (%00) with spaces.
- URL encoding validation.
- Unicode encoding validation.
- Byte range verification.

# Miscellaneous mod\_security features

---

mod\_security can chroot Apache after its already loaded its dynamically loaded libraries.

SecChrootPath /jail/usr/local/apache

It can also change its server signature:

SecServerSignature "Microsoft-IIS/5.0"

# mod\_security as a Reverse Proxy

Ivan Ristic's SecurityFocus article gives great instructions for using mod\_security as a web security proxy (application-specific NIPS).

Set up an Apache server in front of modsecurity.org with the following virtual config:

```
<VirtualHost www.modsecurity.org>  
ServerName www.modsecurity.org  
DocumentRoot /rproxy/nowhere
```

```
ProxyRequests Off
```

```
ProxyPass / http://192.168.254.10/
```

```
ProxyPassReverse / http://192.168.254.10/
```

# mod\_security as a Reverse Proxy (continued)

SecFilterEngine On

SecFilterScanPOST On

SecFilterCheckURLEncoding On

# Scan response body

SecFilterScanOutput On

# On only if using Unicode

SecFilterCheckUnicodeEncoding Off

# Only allow certain byte values to be a part of the request.

# Most English-only applications will work with 32 - 126.

SecFilterForceByteRange 1 255

# mod\_security as a Reverse Proxy (continued)

```
# Audit log logs complete requests. Configured as below it  
# will only log invalid requests for further analysis.
```

```
SecAuditEngine RelevantOnly
```

```
SecAuditLog logs/audit_log
```

```
SecFilterDebugLevel 0
```

```
SecFilterDebugLog logs/modsec_debug_log
```

```
# By default, deny requests with status 500
```

```
SecFilterDefaultAction "deny,log,status:500"
```

```
# Put your mod_security rules here
```

```
# ...
```

```
</VirtualHost>
```

# mod\_parmguard

---

Definitely the most useful of the Apache modules, mod\_parmguard (parameter guard) inspects incoming form submittals for abnormally-set parameters.

The module includes a script that spiders your web application, building up a profile of all forms in use. You can use this profile directly or instead tune it for better detection.

For instance, the script might make sure that a parameter only got numeric values, but you could force those numeric values to be between 1 and 5.

# mod\_paramguard - Setting up Apache to Use

Make sure your Apache server has the module activated:

```
LoadModule paramguard_module  
modules/mod_paramguard.so
```

```
ParamguardConfFile /usr/local/apache/conf/mainconf.xml
```

```
<Location /usr/local/apache/htdocs/applicationdir>
```

```
    ParamguardEngine on
```

```
</Location>
```

# mod\_paramguard configuration

```
<xml version="1.0"?>>
<!DOCTYPE parmguard SYSTEM "mod_parmguard.dtd"/>
<parmguard>

  <global name="http_error_code" value="404"/>

  <url>
    <match>validate.php</match>

    <parm name="name">
      <type name="string"/>
      <attr name="maxlen" value="10"/>
      <attr name="charclass" value="^[a-zA-Z]+$"/>
    </parm>
```

# mod\_paramguard configuration (continued)

```
<parm name="age">  
  <type name="integer"/>  
  <attr name="minval" value="10"/>  
  <attr name="maxval" value="99"/>  
</parm>
```

```
</url>
```

```
</paramguard>
```

# Managing paramguard's configuration

---

htmlspider creates a config file that you can start with. Obviously, it gets radio buttons very right, but can only do length checks on strings. It's up to you to then tighten this up:

```
htmlspider.pl -v -h www.mysite.com/target_form.php
```

As you build more profiles using htmlspider.pl, or as you re-run it on changed sites, you can use confmerger to combine them.

```
confmerger.pl current-config file1 file2
```

# mod\_hackprotect

---

Commercially sold for \$50 per server, this module detects brute-forcing password guessing attempts and locks out those IPs.

This only detects HTTP auth, not custom script authentication.

[www.howlingfrog.com/products/apache](http://www.howlingfrog.com/products/apache)

# mod\_hackdetect

---

Commercially sold for \$50 per server, this one detects multiple logins for a given user from different IP addresses and alerts or deactivates the user account.

This can be strong for detecting users who are sharing their accounts or having their accounts stolen. Utility is limited by the fact that this is focused on HTTP auth.

[www.howlingfrog.com/products/apache](http://www.howlingfrog.com/products/apache)

# mod\_dosevasive

---

This module is quite simple. It keeps count of the number of concurrent connections from each IP address connecting to the server and cuts off an IP that's making too many connections too fast. The cutoff lasts for 10 seconds since the last attempted connection.

[www.freshmeat.net/projects/mod\\_dosevasive](http://www.freshmeat.net/projects/mod_dosevasive)

# mod\_bandwidth

---

This module, only for Apache 1.3, allows you to configure strong bandwidth limits into Apache. While this isn't primarily a security tool, it can be useful for blocking DoS attacks.

[www.cohprog.com/mod\\_bandwidth.html](http://www.cohprog.com/mod_bandwidth.html)

# Credits and Book Reference

The chroot process and much of the security module research is based strongly on Tony Mobily's book:

- Hardening Apache

by Tony Mobily

ISBN 1590593782

Mod\_security info comes from Ivan Ristic's OnLamp article:

- 

[http://www.onlamp.com/pub/a/apache/2003/11/26/mod\\_secu](http://www.onlamp.com/pub/a/apache/2003/11/26/mod_secu)

and Ivan Ristic's SecurityFocus article:

- <http://www.securityfocus.com/infocus/1739>

Much of the other Apache research comes from

# Speaker Bio

---

Jay Beale is the Lead Developer of the Bastille Linux project, the creator of the Center for Internet Security Unix Audit Tool and Linux Benchmark, and a member of the HoneyNet Project.

[www.bastille-linux.org/jay](http://www.bastille-linux.org/jay)

He earns his living as a security consultant and penetration tester with the firm Intelguardians.

[www.intelguardians.com](http://www.intelguardians.com)