Secure Your Apache With mod_security

Secure Your Apache With mod_security

Version 1.0
Author: Falko Timme <ft [at] falkotimme [dot] com>
Last edited 07/05/2006

This article shows how to install and configure *mod_security*. *mod_security* is an Apache module (for Apache 1 and 2) that provides intrusion detection and preven and unknown attacks, such as SQL injection attacks, cross-site scripting, path traversal attacks, etc.

In the first chapter I will show how to install *mod_security* on Debian Sarge, Ubuntu 6.06 LTS (Dapper Drake), and on Fedora Core 5, and in the second chapter I independent from the distribution you're using.

I want to say first that this is not the only way of setting up such a system. There are many ways of achieving this goal but this is the way I take. I do not issue a

1 Installation

1.1 Debian Sarge

mod_security is available as a Debian package in the default Debian repositories, therefore the installation is as simple as this:

```
apt-get install libapache2-mod-security
a2enmod mod-security
/etc/init.d/apache2 force-reload
```

1.2 Ubuntu 6.06 LTS (Dapper Drake)

The installation is exactly the same as on Debian Sarge:

```
apt-get install libapache2-mod-security
a2enmod mod-security
/etc/init.d/apache2 force-reload
```

1.3 Fedora Core 5

On Fedora, you can install and activate *mod_security* like this:

```
yum install mod_security
/etc/init.d/httpd restart
```

You should now find the file */etc/httpd/conf.d/mod_security.conf* which already contains a basic *mod_security* configuration:

```
vi /etc/httpd/conf.d/mod_security.conf
```

```
# Example configuration file for the mod_security Apache module

LoadModule security_module modules/mod_security.so

<IfModule mod_security.c>

    # Turn the filtering engine On or Off
    SecFilterEngine On

    # The audit engine works independently and
    # can be turned On of Off on the per-server or
    # on the per-directory basis
    SecAuditEngine RelevantOnly

    # Make sure that URL encoding is valid
    SecFilterCheckURLEncoding On

    # Unicode encoding check
    SecFilterCheckUnicodeEncoding On

    # Only allow bytes from this range
    SecFilterForceByteRange 1 255

    # Cookie format checks.
    SecFilterCheckCookieFormat On

    # The name of the audit log file
    SecAuditLog logs/audit_log

    # Should mod_security inspect POST payloads
    SecFilterScanPOST On

    # Default action set
    SecFilterDefaultAction "deny,log,status:406"

    # Simple example filter
    # SecFilter 111

    # Prevent path traversal (..) attacks
    # SecFilter "\.\./"

    # Weaker XSS protection but allows common HTML tags
    # SecFilter "<( |\n)*script"

    # Prevent XSS atacks (HTML/Javascript injection)
    # SecFilter "<(.|\n)+>"

    # Very crude filters to prevent SQL injection attacks
    # SecFilter "delete[[:space:]]+from"
    # SecFilter "insert[[:space:]]+into"
    # SecFilter "select.+from"

    # Require HTTP_USER_AGENT and HTTP_HOST headers
```

```
    SecFilterSelective "HTTP_USER_AGENT|HTTP_HOST" "^$"

    # Only accept request encodings we know how to handle
    # we exclude GET requests from this because some (automated)
    # clients supply "text/html" as Content-Type
    SecFilterSelective REQUEST_METHOD "!^GET$" chain
    SecFilterSelective HTTP_Content-Type "!(^$|^application/x-www-form-urlencoded$|^multipart/form-data)"

    # Require Content-Length to be provided with
    # every POST request
    SecFilterSelective REQUEST_METHOD "^POST$" chain
    SecFilterSelective HTTP_Content-Length "^$"

    # Don't accept transfer encodings we know we don't handle
    # (and you don't need it anyway)
    SecFilterSelective HTTP_Transfer-Encoding "!^$"

    # Some common application-related rules from
    # http://modsecrules.monkeydev.org/rules.php?safety=safe

    #Nuke Bookmarks XSS
    SecFilterSelective THE_REQUEST "/modules\.php\?name=Bookmarks\&file=(del_cat\&catname|del_mark\&markname|edit_cat\&catname|edit_cat\&catcomment|marks\&cat

    #Nuke Bookmarks Marks.php SQL Injection Vulnerability
    SecFilterSelective THE_REQUEST "modules\.php\?name=Bookmarks\&file=marks\&catname=.*\&category=.*/\*\*/(union|select|delete|insert)"

    #PHPNuke general XSS attempt
    #/modules.php?name=News&file=article&sid=1&optionbox=
    SecFilterSelective THE_REQUEST "/modules\.php\?*name=<[[:space:]]*script"

    # PHPNuke SQL injection attempt
    SecFilterSelective THE_REQUEST "/modules\.php\?*name=Search*instory="

    #phpnuke sql insertion
    SecFilterSelective THE_REQUEST "/modules\.php*name=Forums.*file=viewtopic*/forum=.*\'/"

    # WEB-PHP phpbb quick-reply.php arbitrary command attempt

    SecFilterSelective THE_REQUEST "/quick-reply\.php" chain
    SecFilter "phpbb_root_path="

    #Topic Calendar Mod for phpBB Cross-Site Scripting Attack
    SecFilterSelective THE_REQUEST "/calendar_scheduler\.php\?start=(<[[:space:]]*script|(http|https|ftp)\:/)"

    # phpMyAdmin: Safe

    #phpMyAdmin Export.PHP File Disclosure Vulnerability
    SecFilterSelective SCRIPT_FILENAME "export\.php$" chain
    SecFilterSelective ARG_what "\.\."

    #phpMyAdmin path vln
    SecFilterSelective REQUEST_URI "/css/phpmyadmin\.css\.php\?GLOBALS\[cfg\]\[ThemePath\]=/etc"

</IfModule>
```

You can keep this configuration, but to get a better understanding of what *mod_security* can do, you should comment out the *<IfModule mod_security.c>...</IfMod* your own *mod_security* ruleset, or just switch back to this one.

Secure Your Apache With mod_security - Page 2

2 Configuration

Now let's start with a basic *mod_security* configuration that allows us to insert rules quickly. We put all *mod_security* rules in the **global** Apache configuratic but not all).

On Debian and Ubuntu, we edit */etc/apache2/apache2.conf* and put this at the end of it:

Debian/Ubuntu:

```
vi /etc/apache2/apache2.conf
```

```
<IfModule mod_security.c>
    # Turn the filtering engine On or Off
    SecFilterEngine On

    # Make sure that URL encoding is valid
    SecFilterCheckURLEncoding On

    # Unicode encoding check
    SecFilterCheckUnicodeEncoding Off

    # Only allow bytes from this range
    SecFilterForceByteRange 0 255

    # Only log suspicious requests
    SecAuditEngine RelevantOnly

    # The name of the audit log file
    SecAuditLog /var/log/apache2/audit_log
    # Debug level set to a minimum
    SecFilterDebugLog /var/log/apache2/modsec_debug_log
    SecFilterDebugLevel 0

    # Should mod_security inspect POST payloads
    SecFilterScanPOST On

    # By default log and deny suspicious requests
    # with HTTP status 500
    SecFilterDefaultAction "deny,log,status:500"

</IfModule>
```

On Fedora, we add pretty much the same to */etc/httpd/conf.d/mod_security.conf*, but change the paths to the log files as Fedora's Apache uses */var/log/http*

Fedora:

```
vi /etc/httpd/conf.d/mod_security.conf
```

```
<IfModule mod_security.c>
    # Turn the filtering engine On or Off
    SecFilterEngine On
```

```
    # Make sure that URL encoding is valid
    SecFilterCheckURLEncoding On

    # Unicode encoding check
    SecFilterCheckUnicodeEncoding Off

    # Only allow bytes from this range
    SecFilterForceByteRange 0 255

    # Only log suspicious requests
    SecAuditEngine RelevantOnly

    # The name of the audit log file
    SecAuditLog /var/log/httpd/audit_log
    # Debug level set to a minimum
    SecFilterDebugLog /var/log/httpd/modsec_debug_log
    SecFilterDebugLevel 0

    # Should mod_security inspect POST payloads
    SecFilterScanPOST On

    # By default log and deny suspicious requests
    # with HTTP status 500
    SecFilterDefaultAction "deny,log,status:500"

</IfModule>
```

Afterwards we restart Apache:

Debian/Ubuntu:

```
/etc/init.d/apache2 restart
```

Fedora:

```
/etc/init.d/httpd restart
```

The directives are pretty self-explanatory.

- `SecFilterEngine` (On|Off): enables/disables the filtering engine.
- `SecFilterCheckURLEncoding` (On|Off): Special characters need to be encoded before they can be transmitted in the URL. With `SecFilterCheckURLEncoding`
- `SecFilterCheckUnicodeEncoding` (On|Off): enables/disables unicode encoding validation. This should be turned off unless you're sure your web applica
- `SecFilterForceByteRange`: force requests to consist only of bytes from a certain byte range. This can be useful to avoid stack overflow attacks. Default
- `SecAuditEngine` (On|Off|RelevantOnly): enables/disables mod_security logging. `RelevantOnly` means: only log relevant requests. Relevant requests are th
- `SecAuditLog`: the path to the `mod_security` log file.
- `SecFilterDebugLog`: path to `mod_security`'s debug log.
- `SecFilterDebugLevel` (0-9): controls how detailed the debug log is. 0: nothing gets logged (for production systems); 1: significant events; 2: info mess
- `SecFilterScanPOST` (On|Off): with mod_security, you cannot only scan GET variables, you can also scan POST variables (from a form on a web site, etc
- `SecFilterDefaultAction`: sets the default action for an event that is filtered by our filtering ruleset (which we still have to define). This directive is follo log it to the audit log, and return a 500 (internal server error) error to the user. I will explain the most important actions next.

Actions

These are the most important actions `mod_security` can apply to an event that is catched by the filtering ruleset:

- `pass`: Allow request to continue on filter match. This action is useful when you want to log a match but otherwise do not want to take action.
- `allow`: This is a stronger version of the previous filter. After this action is performed the request will be allowed through and no other filters will be tri
- `deny`: Interrupt request processing on a filter match. Unless the status action is used too, ModSecurity will immediately return a HTTP 500 error code.
- `status`: Use the supplied HTTP status code when the request is denied.
- `redirect`: On filter match redirect the user to the given URL.
- `exec`: Execute a binary on filter match. Full path to the binary is required.
- `log`: Log filter match to the Apache error log.
- `nolog`: Do not log the filter match. This will also prevent the audit logging from taking place.
- `chain`: Rule chaining allows you to chain several rules into a bigger test.
- `auditlog`: Log the transaction information to the audit log.
- `noauditlog`: Do not log transaction information to the audit log.

Until now not much has happened. I will now present a few filter rules that should give you an idea what you can do with `mod_security`.

Let's assume you have an application that is vulnerable to SQL injection attacks. An attacker could try to delete all records from a MySQL table like this:

```
http://www.example.com/login.php?user=tom';DELETE%20FROM%20users--
```

You can prevent this with this rule:

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

Whenever a request is caught by your filter, something like this is logged to your `audit_log`:

```
============================================
Request: 192.168.0.207 - - [04/Jul/2006:23:43:00 +1200] "GET /login.php?user=tom';DELETE%20FROM%20users-- HTTP/1.1" 500 1215
Handler: (null)
--------------------------------------------
GET /login.php?user=tom';DELETE%20FROM%20users-- HTTP/1.1
Host: 192.168.0.100
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.0.4) Gecko/20060508 Firefox/1.5.0.4
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
mod_security-message: Access denied with code 500. Pattern match "delete[[:space:]]+from" at THE_REQUEST
mod_security-action: 500

HTTP/1.1 500 Internal Server Error
Last-Modified: Fri, 21 Oct 2005 14:30:18 GMT
ETag: "8238-4bf-833a5280"
Accept-Ranges: bytes
Content-Length: 1215
Connection: close
Content-Type: text/html
```

and `SecFilterDefaultAction` is applied (i.e., the request is denied, logged, and the user gets a 500 error). If you want a different action to take place, you can

```
SecFilter "delete[[:space:]]+from" log,redirect:http://example.com/invalid_request.html
```

This would redirect the request to a HTML page that could say something about that the request was invalid.

To prevent more SQL injection attacks, we can add a few other rules:

```
SecFilter "insert[[:space:]]+into"
SecFilter "select.+from"
SecFilter "drop[[:space:]]table"
```

The following directives help to prevent cross-site scripting attacks:

```
SecFilter "<script"
SecFilter "<.+>"
```

This one is for preventing path traversal attacks:

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

Please note: sometimes you find

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

instead of

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

As of `mod_security` 1.8, there is no need to escape dots anymore. This is managed automatically by `mod_security` which means you it doesn't masster if you e

---

This one blocks all requests that do not contain the string `php` in it:

```
SecFilter !php
```

This directive blocks requests that try to execute `/bin/sh` on your server:

```
SecFilter /bin/sh
```

This one blocks all requests that contain the string `viagra`:

```
SecFilter "viagra"
```

You can also use regular expressions like here:

```
SecFilter "(viagra|mortgage|herbal)"
```

The problem with the `SecFilter` directive is that it scans the whole request instead of particular fields. If the referrer is `ihateviagra.mydomain.com`, it would be spam, and your form to submit comments uses the `POST` method, then it would be better to just scan the `POST` variables for the string `viagra`. We can do this v

```
SecFilterSelective "POST_PAYLOAD" "viagra"
```

You can also scan other fields of the request:

```
SecFilterSelective "HTTP_REFERER" "(viagra|mortgage|texasholdem)"
```

would block all requests that contain either `viagra`, `mortgage`, or `texasholdem` in the `HTTP_REFERER` field.

This rule requires `HTTP_USER_AGENT` and `HTTP_HOST` headers in every request:

```
SecFilterSelective "HTTP_USER_AGENT|HTTP_HOST" "^$"
```

You can also block IP addresses:

```
SecFilterSelective "REMOTE_ADDR" "^1.2.3.4$"
```

If you have an input field `url` in your comment form, and you want to scan the value of `url` for the string `viagra`, you do it like this:

```
SecFilterSelective "ARG_url" "viagra"
```

The following rule would redirect the Googlebot to the Google start page:

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

You can find a list of all fields you can scan in the ModSecurity documentation: http://www.modsecurity.org/documentation/modsecurity-apache/1.9.3/htm

You should also check out these pages: http://www.onlamp.com/pub/a/apache/2003/11/26/mod_security.html and http://atomicplayboy.net/blog/2005/01/ and a more detailed explanation about what `mod_security` can do.

`mod_security` also allows your Apache to pretend it's another web server, e.g. like this:

```
SecServerSignature "Microsoft-IIS/5.0"
```

If Apache shouldn't show a signature at all, use this:

```
SecServerSignature " "
```

`mod_security` also allows you to filter outgoing content. For example, if you use PHP scripts, and there's a possibility that your PHP scripts result in a fatal e (because it can contain some important details that only you should see), you can do it like this:

```
SecFilterScanOutput On
SecFilterSelective OUTPUT "Fatal error:" deny,status:500
ErrorDocument 500 /php-fatal-error.html
```

If a fatal error occurs, the user will be redirected to the file `php-fatal-error.html` (which you must create before, of course).

This should give you a basic idea what you can do with `mod_security`. For more examples and details, you should definitely visit these URLs:

- http://www.modsecurity.org
- http://www.modsecurity.org/projects/rules/index.html
- http://www.modsecurity.org/documentation/modsecurity-apache/1.9.3/html-multipage/index.html
- http://www.onlamp.com/pub/a/apache/2003/11/26/mod_security.html
- http://atomicplayboy.net/blog/2005/01/30/an-introduction-to-mod-security

There's also an online rule creator for `mod_security` here: http://leavesrustle.com/tools/modsecurity which helps you to create your own rules.

You should now be able to add your own rules to the basic configuration from above. If you're unsure, you can start with this configuration:

```
<IfModule mod_security.c>
    # Turn the filtering engine On or Off
    SecFilterEngine On

    # Change Server: string
    SecServerSignature " "

    # Make sure that URL encoding is valid
    SecFilterCheckURLEncoding On

    # This setting should be set to On only if the Web site is
    # using the Unicode encoding. Otherwise it may interfere with
    # the normal Web site operation.
    SecFilterCheckUnicodeEncoding Off

    # Only allow bytes from this range
    SecFilterForceByteRange 1 255

    # The audit engine works independently and
    # can be turned On of Off on the per-server or
    # on the per-directory basis. "On" will log everything,
    # "DynamicOrRelevant" will log dynamic requests or violations,
    # and "RelevantOnly" will only log policy violations
    SecAuditEngine RelevantOnly

    # The name of the audit log file
    SecAuditLog /var/log/apache2/audit_log

    # Should mod_security inspect POST payloads
    SecFilterScanPOST On

    # Action to take by default
    SecFilterDefaultAction "deny,log,status:500"

    # Require HTTP_USER_AGENT and HTTP_HOST in all requests
    SecFilterSelective "HTTP_USER_AGENT|HTTP_HOST" "^$"

    # Prevent path traversal (..) attacks
    SecFilter "../"

    # Weaker XSS protection but allows common HTML tags
    SecFilter "<[[:space:]]*script"

    # Prevent XSS atacks (HTML/Javascript injection)
    SecFilter "<(.|n)+>"

    # Very crude filters to prevent SQL injection attacks
    SecFilter "delete[[:space:]]+from"
    SecFilter "insert[[:space:]]+into"
    SecFilter "select.+from"
    SecFilter "drop[[:space:]]table"

    # Protecting from XSS attacks through the PHP session cookie
    SecFilterSelective ARG_PHPSESSID "!^[0-9a-z]*$"
    SecFilterSelective COOKIE_PHPSESSID "!^[0-9a-z]*$"
</IfModule>
```

Another good starting point is the configuration proposed by the *mod_security* documentation (http://www.modsecurity.org/documentation/modsecurity-apa

```
<IfModule mod_security.c>
    # Turn ModSecurity On
    SecFilterEngine On

    # Reject requests with status 403
    SecFilterDefaultAction "deny,log,status:403"

    # Some sane defaults
    SecFilterScanPOST On
    SecFilterCheckURLEncoding On
    SecFilterCheckUnicodeEncoding Off

    # Accept almost all byte values
    SecFilterForceByteRange 1 255

    # Server masking is optional
    # SecServerSignature "Microsoft-IIS/5.0"

    SecUploadDir /tmp
    SecUploadKeepFiles Off

    # Only record the interesting stuff
    SecAuditEngine RelevantOnly
    SecAuditLog /var/log/apache2/audit_log

    # You normally won't need debug logging
    SecFilterDebugLevel 0
    SecFilterDebugLog /var/log/apache2/modsec_debug_log

    # Only accept request encodings we know how to handle
    # we exclude GET requests from this because some (automated)
    # clients supply "text/html" as Content-Type
    SecFilterSelective REQUEST_METHOD "!^(GET|HEAD)$" chain
    SecFilterSelective HTTP_Content-Type \
    "!(^application/x-www-form-urlencoded$|^multipart/form-data;)"

    # Do not accept GET or HEAD requests with bodies
    SecFilterSelective REQUEST_METHOD "^(GET|HEAD)$" chain
    SecFilterSelective HTTP_Content-Length "!^$"

    # Require Content-Length to be provided with
    # every POST request
    SecFilterSelective REQUEST_METHOD "^POST$" chain
    SecFilterSelective HTTP_Content-Length "^$"

    # Don't accept transfer encodings we know we don't handle
    SecFilterSelective HTTP_Transfer-Encoding "!^$"
</IfModule>
```

Or you take the configuration that comes with Fedora's *mod_security* package. Always make sure you adjust the paths to the log files!

3 Links

- Apache Module *mod_security*: http://www.modsecurity.org
- *mod_security* Documentation: http://www.modsecurity.org/documentation/modsecurity-apache/1.9.3/html-multipage/index.html
- ModSecurity Rules: http://www.modsecurity.org/projects/rules/index.html
- Introducing mod_security (onlamp.com): http://www.onlamp.com/pub/a/apache/2003/11/26/mod_security.html
- An introduction to mod_security (atomicplayboy.net): http://atomicplayboy.net/blog/2005/01/30/an-introduction-to-mod-security
- Online Rule Creator For *mod_security*: http://leavesrustle.com/tools/modsecurity