Apache mod_security and CSF integration

A more effective way to block WordPress brute force attempts without the need for silly plugins or huge .htaccess files.

What is mod_security?
An apache plugin that catches bad HTTP requests (like the \234x\x\234\23x\ ones you see in frequently-targeted sites' error logs.) It can also be configured to watch for custom triggers, for instance WordPress brute-force attacks or even more nefarious things like uploading executable data to a comment form.

Why not use WordFence?
Plugins such as WordFence are great for two things:

- Scanning for existing exploits (ideally these would not be a problem because the server would be secure in the first place)
- Blocking brute-force attempts through the use of .htaccess (which still allows malicious users to view pages other than wp-login.php and doesn't block them from other directories on the server not covered by .htaccess)

We already use CSF on our cPanel shared servers and CSF has built-in functionality that allows it to integrate with mod_security.

What are the advantages of using CSF vs. WordFence?
A CSF block will prevent packets from ever reaching the server if they come from the specified IP address. This means that as soon as a malicious user triggers a mod_security rule, they are no longer able to communicate with the server *including possible attempts at exploiting apache vulnerabilities which may arise that would render the .htaccess file useless*.

Awesome! Less jabber, more implementation!

Stopping WordPress Brute-Force Attacks With mod_security

In the csf configuration file (usually /etc/csf/csf.conf) there are two lines which need to be changed:

```
# [*]Enable failure detection of repeated Apache mod_security rule triggers

LF_MODSEC = "1"
LF_MODSEC_PERM = "1"
```

LF_MODSEC refers to the number of times mod_security must block an IP using its filtering method before CSF will pick up and block it.

LF_MODSEC_PERM set to 1 means that the block will be permanent (as opposed to temporary.)

Additionally, the following line should be changed:

```
MODSEC_LOG = "/var/log/httpd/error_log"
```

This line needs to point to the Apache error log, not the modsec audit_log (lots of debugging taught me that for whatever reason CSF doesn't properly respond to things in the audit log but will still pick up modsec events from the error log.)

Depending on the number and paranoia of mod_security rules in place, LF_MODSEC may need to be increased to prevent false-positive blocks.

NOTE: It is very important to check /etc/csf/csf.syslogs and make sure modsec_audit.log in the Apache section is pointing to the right place as well as the other Apache logs. Otherwise LFD won't process the log and CSF won't block anything. Additionally, /etc/httpd/conf.d/modsecurity.conf needs to have SecRuleEngine changed to "On"

Rules For wp-login

```
SecAction phase:1,nolog,pass,initcol:ip=%{REMOTE_ADDR},initcol:user=%{REMOTE_ADDR},id:5000134
<FilesMatch "wp-login.php">
# Setup brute force detection.
# React if block flag has been set.
SecRule user:bf_block "@gt 0" "deny,severity:0,status:406,log,auditlog,id:5000135"
# Setup Tracking. On a successful login, a 302 redirect is performed, a 200 indicates login failed.
SecRule RESPONSE_STATUS "^302" "phase:5,t:none,nolog,pass,setvar:ip.bf_counter=0,id:5000136"
SecRule RESPONSE_STATUS "^200"
phase:5,chain,t:none,nolog,pass,setvar:ip.bf_counter=+1,deprecatevar:ip.bf_counter=1/180,id:5000137
"
SecRule ip:bf_counter "@gt 20"
t:none,setvar:user.bf_block=1,expirevar:user.bf_block=300,setvar:ip.bf_counter=0"
SecAuditLogParts ABFZ
</FilesMatch>
```

(Note that the FilesMatch directive operates on a Regular Expression and thus does not require path specification or any wildcards in order to work on this file in all directories.)

How These Rules Work

The first line sets up a new SecAction (take a wild guess at what this abbreviates) which runs in **Phase 1** of document processing. The phases of mod_security are as follows:

1. Request headers (REQUEST_HEADERS)
2. Request body (REQUEST_BODY)
3. Response headers (RESPONSE_HEADERS)
4. Response body (RESPONSE_BODY)
5. Logging (LOGGING)

For wordpress logins, the only area of interest is the request headers. Using phase:1 prevents mod_security from processing the entire HTTP response which would waste a lot of resources.

Severity should be set to 0 as this reflects an EMERGENCY condition to which CSF will respond. Other levels don't seem to be effective.

Why the request headers?

When a user puts in a **correct** password to WordPress' login prompt, they receive a 302 redirect response from the server which places them in the admin area (after setting the proper session values, etc.) When a user puts in an **invalid** password, they are redirected to wp-login.php again, with a 200 OK response code.

But then they'll be blocked for just trying to log in!

Not true. The following two rules address this issue:

```
SecRule RESPONSE_STATUS "^302" "phase:5,t:none,nolog,pass,setvar:ip.bf_counter=0,id:5000136"
SecRule RESPONSE_STATUS "^200"
"phase:5,chain,t:none,nolog,pass,setvar:ip.bf_counter=+1,deprecatevar:ip.bf_counter=1/180,id:500013
7"
```

The first rule clears the "counter" of bad logins upon a successful login.

The second increments it for a failed login.

After over 20 attempts:

```
SecRule ip:bf_counter "@gt 20"
"t:none,setvar:user.bf_block=1,expirevar:user.bf_block=300,setvar:ip.bf_counter=0"
```

The user will be blocked on the 21st attempt for a period of 300 seconds (5 minutes.)

After 1 of these 5-minute blocks which equates to 21+ failed login attempts (definitely not legit traffic) the IP will be blocked in CSF permanently. The amount of time could be reduced to a period of hours or days depending on what is judged to be the best during production.

More Security for WordPress

There are numerous rules for mod_security which are not enabled by default but which come standard with an installation of mod_security on CentOS. Some of these conflict with WordPress. I will go over how to fix those conflicts as well as the benefits of using these rules.

SQL Injection Prevention – Prevents WordPress from executing legit queries:

One of the default rules for mod_security checks for the number of special characters in a request URI. WordPress' load-styles and load-scripts files trigger this rule, so it's important to ignore this rule within those two files:

```
<LocationMatch /load-styles.php>
    SecRuleRemoveById 981173
</LocationMatch>
<LocationMatch /load-scripts.php>
    SecRuleRemoveById 981173
</LocationMatch>
```

These directives should be placed in a .conf file wherever apache's extra conf is stored on the server (usually /etc/httpd/conf.d/) . For instance, /etc/httpd/conf.d/modsec_exceptions.conf. Other exceptions should be added to this file as needed and it should be called with an Include directive inside an <IfModule> statement within the main Apache configuration file. Failure to do this will result in odd errors and a 403 being logged for a lot of normal admin-area activities.

Asynchronous Uploads – Blocked by Mod_Security:
WordPress uses a script to upload files from the admin area which triggers heuristic filters in mod_security. The following directive can be added to ensure that this script is not interrupted by mod_security and that legitimate users are not blocked by CSF:

```
<IfModule mod_security.c>
SecFilterSelective REQUEST_URI "/wp-admin/async-upload\.php.*$" "allow,pass"
</IfModule>
```

Failure to do this will result in odd errors and a 403 being logged for a lot of normal admin-area activities.

It is still a good idea to install suPHP and in fact will prevent some upload issues with plugins and updates.

Example Config Files

Httpd.conf (could also go in vhosts directive elsewhere)

```
<IfModule mod_security2.c>
SecAuditEngine RelevantOnly
SecAuditLogRelevantStatus "^(?:5|4(?!04))"
SecAuditLogType Serial
SecAuditLog /var/log/modsec_audit.log
SecAuditLogParts ABIFHZ
SecArgumentSeparator "&"
SecCookieFormat 0
SecRequestBodyInMemoryLimit 131072
#SecDataDir /var/asl/data/msa
SecTmpDir /tmp
#SecAuditLogStorageDir /var/asl/data/audit
#SecResponseBodyLimitAction ProcessPartial
#SecDataDir /var/asl/data/msa
SecDefaultAction "phase:2,deny,log,status:406"
SecRule REMOTE_ADDR "^127.0.0.1$" nolog,allow
#Include "/usr/local/apache/conf/modsec2.user.conf"
Include "/etc/httpd/conf.d/modsec_exceptions.conf"
</IfModule>
```

Modsec_exceptions.conf (Should be one Include directive at bottom of httpd.conf)

```
 <LocationMatch /load-styles.php>
             SecRuleRemoveById 981173
 </LocationMatch>
 <LocationMatch /load-scripts.php>
             SecRuleRemoveById 981173
 </LocationMatch>
 <LocationMatch /wp-admin/upload.php>
            SecRuleRemoveById 981173
 </LocationMatch>
 <LocationMatch update.php>
             SecRuleRemoveById 981173
 </LocationMatch>
SetEnvIfNoCase Content-Type "^multipart/form-data;" "MODSEC_NOPOSTBUFFERING=Do not buffer
file uploads"
```

Major Benefits

The major benefits to using this method are:

- We can standardize it to run on all WordPress installs transparently without the user needing to install or update "security plugins" which are themselves vulnerable to PHP bugs and other issues.
- It's also a "better practice" as it prevents a lot of unnecessary processing both on the malicious request side as well as the blocking side.
- Additionally, this method could be adapted to any number of other CMS's, provided adequate research into the content of HTTP responses for valid vs. invalid logins. It could also be used to block in CSF based on User Agent, which would prevent "bad bots" from crawling the site using different IP ranges.
- Additionally, using this method would ensure that a hacker attempting to brute-force one domain on a server would be prevented from trying it on any other domain on the server – thus greatly reducing the number of brute-force attempts as well as the amount of processing done on .htaccess files and WordFence scripts.
- It would also prevent oversized data from being uploaded which could cause a buffer overflow in PHP, something WordFence would not be able to accomplish as it is also processed by PHP and subject to the same vulnerabilities.

Pitfalls

- There is a limit to the number of deny rules in CSF which would need to be increased or disabled in order to ensure this setup doesn't interfere with other blocking triggers.
- A lot of initial research was done, however it's impossible to predict what kind of silly things users will do and as such this method may require some modification that isn't listed here.