How to Set Up ModSecurity with Apache on Debian/Ubuntu

Last Updated: December 8th, 2022 Xiao Guoan (Admin) 29 Comments Security

This tutorial is going to show you how to install and use **ModSecurity** with Apache on Debian/Ubuntu servers. ModSecurity is the most well-known open-source **web application firewall** (WAF), providing comprehensive protection for your web applications (like WordPress, Nextcloud, Ghost etc) against a wide range of Layer 7 (HTTP) attacks, such as SQL injection, cross-site scripting, and local file inclusion.

Web applications are inherently insecure. If you are a WordPress admin, you probably hear news of hackers exploiting vulnerabilities in WordPress plugins and themes every once in a while. It’s essential that you deploy a WAF on your web server, especially when you have old applications that don’t receive updates. ModSecurity is originally created by Ivan Ristić in 2002, currently maintained by Trustwave SpiderLabs. It’s the world’s most widely deployed WAF, used by over a million websites. cPanel, the most widely used hosting control panel, includes ModSecurity as its WAF.

You may have heard of other host-based firewalls like iptables, UFW, and Firewalld, etc. The difference is that they work on layers 3 and 4 of the OSI model and take actions based on IP address and port number. ModSecurity, or web application firewalls in general, is specialized to focus on HTTP traffic (layer 7 of the OSI model) and takes action based on the content of HTTP request and response.

ModSecurity 3

ModSecurity was originally designed for Apache web server. It could work with Nginx before version 3.0 but suffered from poor performance. ModSecurity 3.0 (aka **libmodsecurity**) was released in 2017. It’s a milestone release, particularly for Nginx users, as it’s the first version to work natively with Nginx. The caveat of ModSecurity 3 is that it doesn’t yet have all the features as in the previous version (2.9), though each new release will add some of the missing features. Nginx users should use ModSecurity 3. However, if you use Apache, it’s recommended to continue using the 2.9 branch for the time being.

Step 1: Install ModSecurity with Apache on Debian/Ubuntu

The ModSecurity module for Apache is included in the default Debian/Ubuntu repository. To install it, run

```
sudo apt install libapache2-mod-security2
```

Then enable this module.

```
sudo a2enmod security2
```

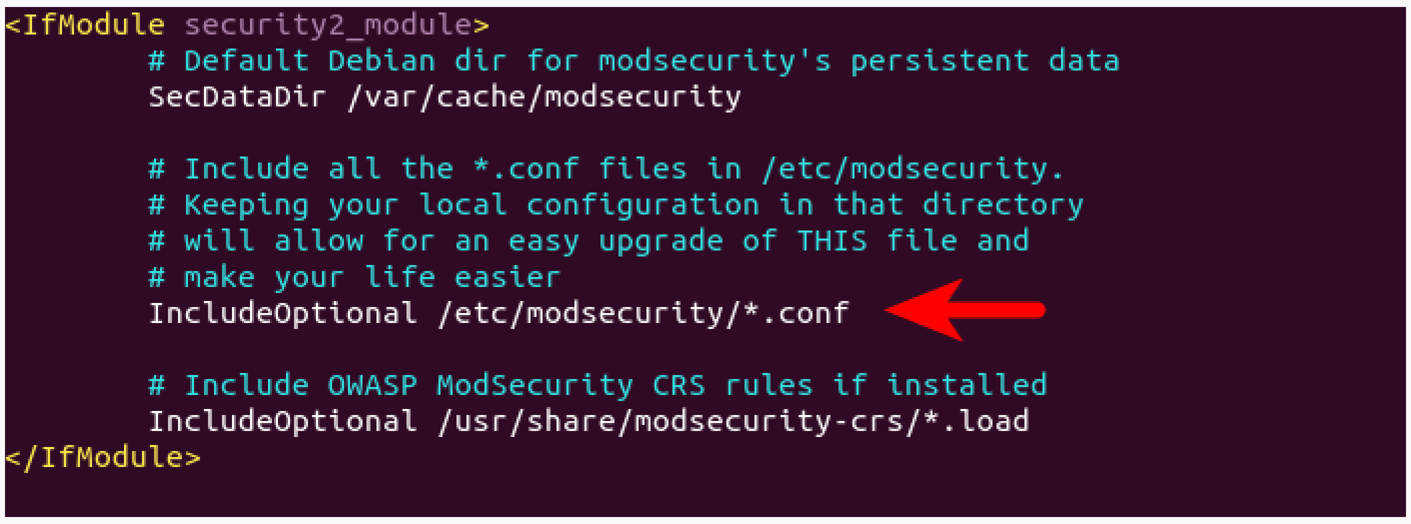
Restart Apache for the change to take effect.

```
sudo systemctl restart apache2
```

Step 2: Configure ModSecurity

In the `/etc/apache2/mods-enabled/security2.conf` configuration file, you can find the following line.

```
IncludeOptional /etc/modsecurity/*.conf
```



```
<IfModule security2_module>
        # Default Debian dir for modsecurity's persistent data
        SecDataDir /var/cache/modsecurity

        # Include all the *.conf files in /etc/modsecurity.
        # Keeping your local configuration in that directory
        # will allow for an easy upgrade of THIS file and
        # make your life easier
        IncludeOptional /etc/modsecurity/*.conf

        # Include OWASP ModSecurity CRS rules if installed
        IncludeOptional /usr/share/modsecurity-crs/*.load
</IfModule>
```

This means Apache will include all the `*.conf` files in `/etc/modsecurity/` directory. We need to rename the `modsecurity.conf-recommended` file to make it work.

```
sudo mv /etc/modsecurity/modsecurity.conf-recommended /etc/modsecurity/modsecurity.con
f
```

Then edit this file with a command-line text editor like Nano.

```
sudo nano /etc/modsecurity/modsecurity.conf
```

Find the following line.

```
SecRuleEngine DetectionOnly
```

This config tells ModSecurity to log HTTP transactions, but takes no action when an attack is detected. Change it to the following, so ModSecurity will detect and block web attacks.

```
SecRuleEngine On
```

Then find the following line (line 186), which tells ModSecurity what information should be included in the audit log.

```
SecAuditLogParts ABDEFHIJZ
```

However, the default setting is wrong. You will know why later when I explain how to understand ModSecurity logs. The setting should be changed to the following.

```
SecAuditLogParts ABCEFHJKZ
```

Save and close the file. Then restart Apache for the change to take effect. (Reloding the web server isn’t enough.)

```
sudo systemctl restart apache2
```

Step 3: Install the OWASP Core Rule Set (CRS)

To make ModSecurity protect your web applications, you need to define rules to detect malicious actors and block them. For beginners, it's a good idea to install existing rule sets, so you can get started quickly and then learn the nitty-gritty down the road. There are several free rule sets for ModSecurity. The **OWASP Core Rule Set** (CRS) is the standard rule set used with ModSecurity.

- It's free, community-maintained and the most widely used rule set that provides a sold default configuration for ModSecurity.
- It contains rules to help stop common attack vectors, including SQL injection (SQLi), cross-site scripting (XSS), and many others.
- It can integrate with Project Honeypot.
- It also contains rules to detect bots and scanners.
- It has been tuned through wide exposure to have very few false positives.

When installing ModSecurity from the default Debian/Ubuntu repository, the `modsecurity-crs` package is also installed as a dependency. This package contains the OWASP core rule set version 3.x. However, it can become out of date. If you care about security, you should use the latest version of core rule set.

Download the latest OWASP CRS from GitHub.

```
wget https://github.com/coreruleset/coreruleset/archive/v3.3.0.tar.gz
```

Extract the file.

```
tar xvf v3.3.0.tar.gz
```

Create a directory to store CRS files.

```
sudo mkdir /etc/apache2/modsecurity-crs/
```

Move the extracted directory to `/etc/apache2/modsecurity-crs/`.

```
sudo mv coreruleset-3.3.0/ /etc/apache2/modsecurity-crs/
```

Go to that directory.

```
cd /etc/apache2/modsecurity-crs/coreruleset-3.3.0/
```

Rename the `crs-setup.conf.example` file.

```
sudo mv crs-setup.conf.example crs-setup.conf
```

Edit the `/etc/apache2/mods-enabled/security2.conf` file.

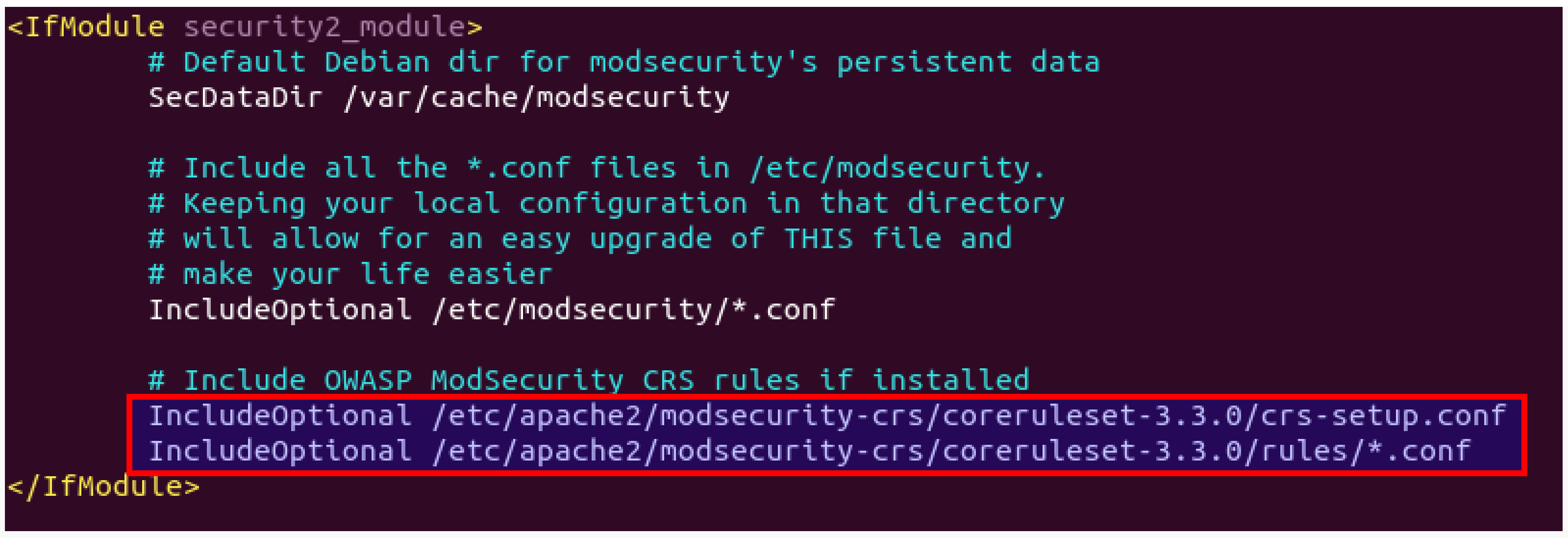
```
sudo nano /etc/apache2/mods-enabled/security2.conf
```

Find the following line, which loads the default CRS files.

```
IncludeOptional /usr/share/modsecurity-crs/*.load
```

Change it to the following, so the latest OWASP CRS will be used.

```
IncludeOptional /etc/apache2/modsecurity-crs/coreruleset-3.3.0/crs-setup.conf
IncludeOptional /etc/apache2/modsecurity-crs/coreruleset-3.3.0/rules/*.conf
```


```
<IfModule security2_module>
        # Default Debian dir for modsecurity's persistent data
        SecDataDir /var/cache/modsecurity

        # Include all the *.conf files in /etc/modsecurity.
        # Keeping your local configuration in that directory
        # will allow for an easy upgrade of THIS file and
        # make your life easier
        IncludeOptional /etc/modsecurity/*.conf

        # Include OWASP ModSecurity CRS rules if installed
        IncludeOptional /etc/apache2/modsecurity-crs/coreruleset-3.3.0/crs-setup.conf
        IncludeOptional /etc/apache2/modsecurity-crs/coreruleset-3.3.0/rules/*.conf
</IfModule>
```

Save and close the file. Then test Apache configuration.

```
sudo apache2ctl -t
```

If the syntax is OK, then restart Apache.

```
sudo systemctl restart apache2
```

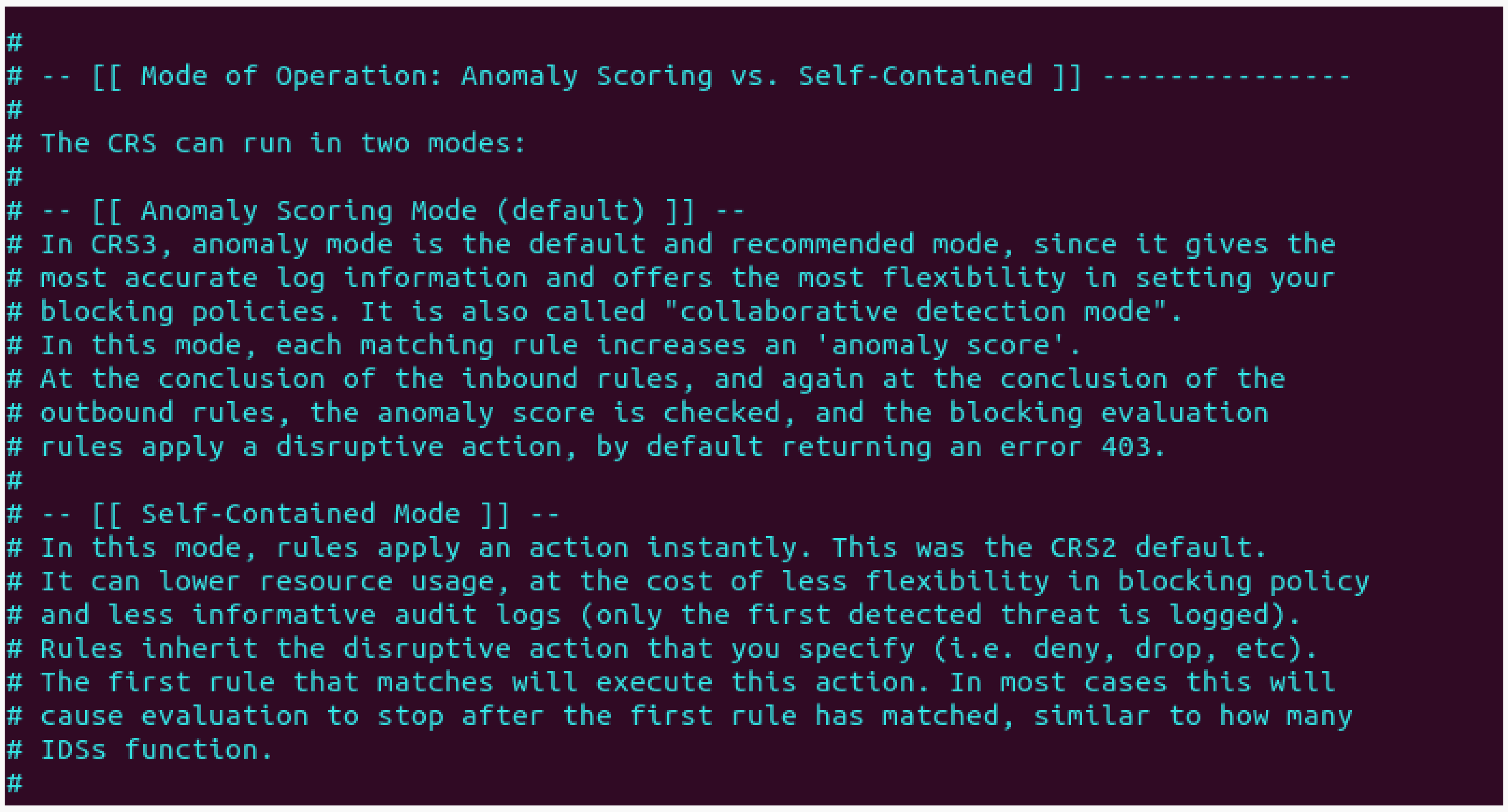
Step 4: Learn How OWASP CRS Works

Let’s take a look at the CRS config file, which provides you with good documentation on how CRS works.

```
sudo nano /etc/apache2/modsecurity-crs/coreruleset-3.3.0/crs-setup.conf
```

You can see that OWASP CRS can run in two modes:

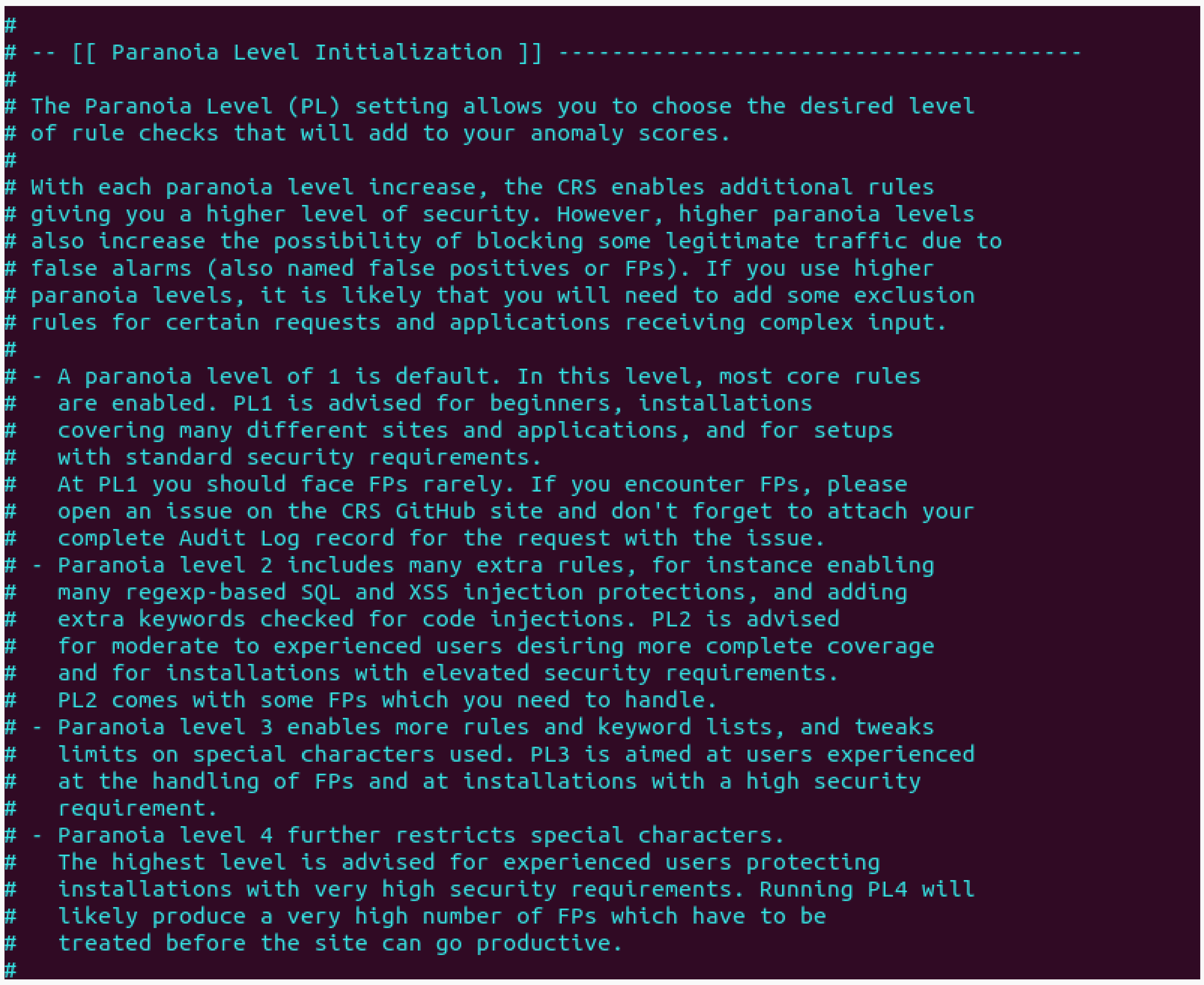
- **self-contained mode**. This is the traditional mode used in CRS v2.x. If an HTTP request matches a rule, ModSecurity will block the HTTP request immediately and stop evaluating remaining rules.
- **anomaly scoring mode**. This is the default mode used in CRS v3.x. ModSecurity will check an HTTP request against all rules, and add a score to each matching rule. If a threshold is reached, then the HTTP request is considered an attack and will be blocked. The default score for inbound requests is 5 and for outbound response is 4.


```
#
# -- [[ Mode of Operation: Anomaly Scoring vs. Self-Contained ]] ---------------
#
# The CRS can run in two modes:
#
# -- [[ Anomaly Scoring Mode (default) ]] --
# In CRS3, anomaly mode is the default and recommended mode, since it gives the
# most accurate log information and offers the most flexibility in setting your
# blocking policies. It is also called "collaborative detection mode".
# In this mode, each matching rule increases an 'anomaly score'.
# At the conclusion of the inbound rules, and again at the conclusion of the
# outbound rules, the anomaly score is checked, and the blocking evaluation
# rules apply a disruptive action, by default returning an error 403.
#
# -- [[ Self-Contained Mode ]] --
# In this mode, rules apply an action instantly. This was the CRS2 default.
# It can lower resource usage, at the cost of less flexibility in blocking policy
# and less informative audit logs (only the first detected threat is logged).
# Rules inherit the disruptive action that you specify (i.e. deny, drop, etc).
# The first rule that matches will execute this action. In most cases this will
# cause evaluation to stop after the first rule has matched, similar to how many
# IDSs function.
#
```

When running in anomaly scoring mode, there are 4 paranoia levels.

- Paranoia level 1 (default)
- Paranoia level 2
- Paranoia level 3
- Paranoia level 4

With each paranoia level increase, the CRS enables additional rules giving you a higher level of security. However, higher paranoia levels also increase the possibility of blocking some legitimate traffic due to false alarms.



```
#
# -- [[ Paranoia Level Initialization ]] ---------------------------------------
#
# The Paranoia Level (PL) setting allows you to choose the desired level
# of rule checks that will add to your anomaly scores.
#
# With each paranoia level increase, the CRS enables additional rules
# giving you a higher level of security. However, higher paranoia levels
# also increase the possibility of blocking some legitimate traffic due to
# false alarms (also named false positives or FPs). If you use higher
# paranoia levels, it is likely that you will need to add some exclusion
# rules for certain requests and applications receiving complex input.
#
# - A paranoia level of 1 is default. In this level, most core rules
#   are enabled. PL1 is advised for beginners, installations
#   covering many different sites and applications, and for setups
#   with standard security requirements.
#   At PL1 you should face FPs rarely. If you encounter FPs, please
#   open an issue on the CRS GitHub site and don't forget to attach your
#   complete Audit Log record for the request with the issue.
# - Paranoia level 2 includes many extra rules, for instance enabling
#   many regexp-based SQL and XSS injection protections, and adding
#   extra keywords checked for code injections. PL2 is advised
#   for moderate to experienced users desiring more complete coverage
#   and for installations with elevated security requirements.
#   PL2 comes with some FPs which you need to handle.
# - Paranoia level 3 enables more rules and keyword lists, and tweaks
#   limits on special characters used. PL3 is aimed at users experienced
#   at the handling of FPs and at installations with a high security
#   requirement.
# - Paranoia level 4 further restricts special characters.
#   The highest level is advised for experienced users protecting
#   installations with very high security requirements. Running PL4 will
#   likely produce a very high number of FPs which have to be
#   treated before the site can go productive.
#
```

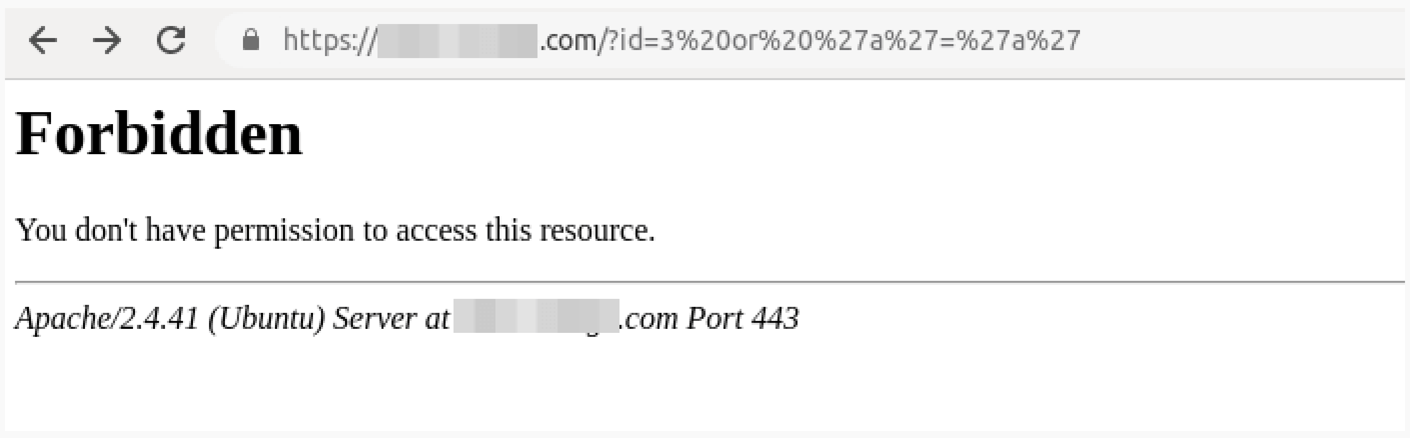
These are the two basic concepts you need to understand before using the CRS. Now we can close the file. The individual CRS rules are stored in `/etc/apache2/modsecurity-crs/coreruleset-3.3.0/rules/` directory. Each matching rule will increase the anomaly score.

Step 5: Testing

To check if ModSecurity is working, you can launch a simple SQL injection attack on your own website. (Please note that it's illegal to do security testing on other people's websites without authorization.) Enter the following URL in your web browser.

```
https://yourdomain.com/?id=3 or 'a'='a'
```

If ModSecurity is working properly, your Apache web server should return a 403 forbidden error message.



https://[redacted].com/?id=3%20or%20%27a%27=%27a%27

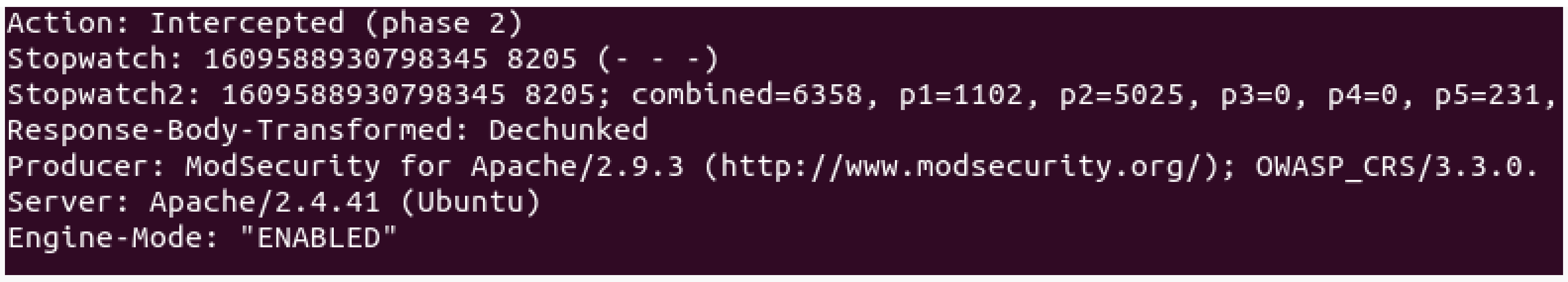
Forbidden

You don't have permission to access this resource.

Apache/2.4.41 (Ubuntu) Server at [redacted].com Port 443

And in the audit log (`/var/log/apache2/modsec_audit.log`), you can see the following line in section H, which means ModSecurity detected and blocked this SQL injection attack by using OWASP CRS v3.3.0.

```
Action: Intercepted (phase 2)
```


```
Action: Intercepted (phase 2)
Stopwatch: 1609588930798345 8205 (- - -)
Stopwatch2: 1609588930798345 8205; combined=6358, p1=1102, p2=5025, p3=0, p4=0, p5=231,
Response-Body-Transformed: Dechunked
Producer: ModSecurity for Apache/2.9.3 (http://www.modsecurity.org/); OWASP_CRS/3.3.0.
Server: Apache/2.4.41 (Ubuntu)
Engine-Mode: "ENABLED"
```

When ModSecurity runs in `DetectionOnly` mode, it won't block this SQL injection attack.

Step 6: Understanding the ModSecurity Logs

It's important to analyze the ModSecurity logs, so you will know what kind of attacks are directed to your web applications and take better actions to defend against threat actors. There are mainly two kinds of logs in ModSecurity:

- debug log: disabled by default.
- audit log: `/var/log/apache2/modsec_audit.log`

To understand ModSecurity audit logs, you need to know the 5 processing phases in ModSecurity, which are:

- Phase 1: Inspect request headers
- Phase 2: Inspect request body
- Phase 3: Inspect response headers
- Phase 4: Inspect response body
- Phase 5: Action (logging/blocking malicious requests)

They are also two types of logging file.

- **Serial**: one file for all logs. This is the default.
- **Concurrent**: multiple files for logging. This can provide better write performance. If you can notice your web pages slowing down after enabling ModSecurity, you can choose to use the concurrent logging type.

Events in the log are divided into several sections.

- section A: audit log header
- section B: request header
- section C: request body
- section D: reserved
- section E: intermediary response body
- section F: final response headers
- section G: reserved
- section H: audit log trailer
- section I: compact request body alternative, which excludes files
- section J: information on uploaded files
- section K: every rule matched by an event, in order of match
- section Z: final boundary

If you run a high traffic website, the ModSecurity audit log can get too large very quickly, so we need to configure log rotation for the ModSecurity audit log. Create a logrotate configuration file for ModSecurity.

```
sudo nano /etc/logrotate.d/modsecurity
```

Add the following lines to this file.

```
/var/log/apache2/modsec_audit.log
{
        rotate 14
        daily
        missingok
        compress
        delaycompress
        notifempty
}
```

This will rotate the log file every day (`daily`), compressing old versions (`compress`). The previous 14 log files will be kept (`rotate 14`), and no rotation will occur if the file is empty (`notifempty`). Save and close the file.

Step 7: Handling False Positives

ModSecurity is a generic web application firewall and not designed for a specific web application. The OWASP core rule set is also a generic rule set with no particular application in mind, so it's likely that you will see false positives after enabling ModSecurity and OWASP CRS. If you increase the paranoia level in the CRS, there will be more false positives.

For example, by default, the CRS forbids Unix command injection like entering `sudo` on a web page, which is rather common on my blog. To eliminate false positives, you need to add rule exclusions to the CRS.

Application-Specific Rule Exclusions

There are some prebuilt, application-specific exclusions shipped with OWASP CRS. Edit the `crs-setup.conf` file.

```
sudo nano /etc/apache2/modsecurity-crs/coreruleset-3.3.0/crs-setup.conf
```

Go to the **Application Specific Rule Exclusions** section and find the following lines.

```
#SecAction \
# "id:900130,\
#  phase:1,\
#  nolog,\
#  pass,\
#  t:none,\
#  setvar:tx.crs_exclusions_cpanel=1,\
#  setvar:tx.crs_exclusions_drupal=1,\
#  setvar:tx.crs_exclusions_dokuwiki=1,\
#  setvar:tx.crs_exclusions_nextcloud=1,\
#  setvar:tx.crs_exclusions_wordpress=1,\
#  setvar:tx.crs_exclusions_xenforo=1"
```

For instance, If I want to enable WordPress exclusions, the above lines should be changed to the following. Please be careful about the syntax. There should be no comments between `t:none,\` and `setvar:tx.crs_exclusions_wordpress=1"`. (The backslash `\` character at the end indicates the next line is a continuation of the current line.)

```
SecAction \
  "id:900130,\
   phase:1,\
   nolog,\
   pass,\
   t:none,\
   setvar:tx.crs_exclusions_wordpress=1"
#  setvar:tx.crs_exclusions_cpanel=1,\
#  setvar:tx.crs_exclusions_drupal=1,\
#  setvar:tx.crs_exclusions_dokuwiki=1,\
#  setvar:tx.crs_exclusions_nextcloud=1,\
#  setvar:tx.crs_exclusions_xenforo=1"
```

Save and close the file. Then test Apache configurations.

```
sudo apache2ctl -t
```

If the test is successful, restart Apache for the change to take effect.

```
sudo systemctl restart apache2
```

Note that if you have multiple applications such as (WordPress, Nextcloud, Drupal, etc) installed on the same server, then the above rule exclusions will be applied to all applications. To minimize the security risks, you should enable a rule exclusion for one application only. To do that, go to the `/etc/apache2/modsecurity-crs/coreruleset-3.3.0/rules/` directory.

```
cd /etc/apache2/modsecurity-crs/coreruleset-3.3.0/rules/
```

Rename the `REQUEST-900-EXCLUSION-RULES-BEFORE-CRS` file.

```
sudo mv REQUEST-900-EXCLUSION-RULES-BEFORE-CRS.conf.example REQUEST-900-EXCLUSION-RULES-BEFORE-CRS.conf
```

Then edit this file.

```
sudo nano REQUEST-900-EXCLUSION-RULES-BEFORE-CRS.conf
```

Add the following line at the bottom of this file. If your WordPress is using the `blog.yourdomain.com` sub-domain and the request header sent from visitor’s browser contains this sub-domain, then ModSecurity will apply the rule exclusions for WordPress.

```
SecRule REQUEST_HEADERS:Host "@streq blog.yourdomain.com" "id:1000,phase:1,setvar:tx.crs_exclusions_wordpress=1"
```

If you have installed Nextcloud on the same server, then you can also add the following line in this file, so if a visitor is accessing your Nextcloud sub-domain, ModSecurity will apply the Nextcloud rule exclusions.

```
SecRule REQUEST_HEADERS:Host "@streq nextcloud.yourdomain.com" "id:1001,phase:1,setvar:tx.crs_exclusions_nextcloud=1"
```

Save and close this file. Then test Apache configurations.

```
sudo apache2ctl -t
```

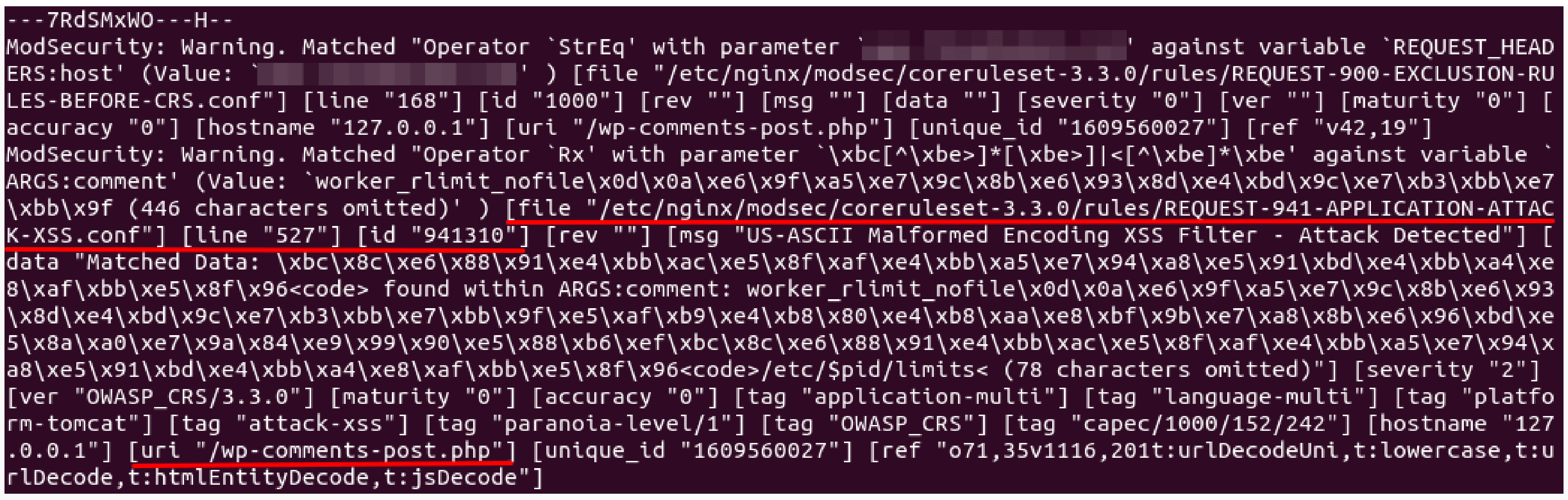
If the test is successful, rstart Apache for the change to take effect.

```
sudo systemctl restart apache2
```

Custom Rule Exclusions

Enabling the prebuilt application-specific rule exclusions might not eliminate all false positives. If so, you need to examine the ModSecurity audit log (`/var/log/apache2/modsec_audit.log`), check which rule caused the false positive and add your custom rule exclusions in the `REQUEST-900-EXCLUSION-RULES-BEFORE-CRS.conf` file.

Section H in the audit log tells you which rule is matched. For example, If I try to use the `<code>...</code>` HTML in the comment form, ModSecurity blocks my comment. The following log tells me that the HTTP request matched a rule in `REQUEST-941-APPLICATION-ATTACK-XSS.conf` (line 527). The rule ID is 941310. The request URI is `/wp-comments-post.php`.



```
---7RdSMxWO---H--
ModSecurity: Warning. Matched "Operator `StrEq' with parameter `[redacted]' against variable `REQUEST_HEAD
ERS:host' (Value: `[redacted]' ) [file "/etc/nginx/modsec/coreruleset-3.3.0/rules/REQUEST-900-EXCLUSION-RU
LES-BEFORE-CRS.conf"] [line "168"] [id "1000"] [rev ""] [msg ""] [data ""] [severity "0"] [ver ""] [maturity "0"] [
accuracy "0"] [hostname "127.0.0.1"] [uri "/wp-comments-post.php"] [unique_id "1609560027"] [ref "v42,19"]
ModSecurity: Warning. Matched "Operator `Rx' with parameter `\xbc[^\xbe>]*[\xbe>]|<[^\xbe]*\xbe' against variable `
ARGS:comment' (Value: `worker_rlimit_nofile\x0d\x0a\xe6\x9f\xa5\xe7\x9c\x8b\xe6\x93\x8d\xe4\xbd\x9c\xe7\xb3\xbb\xe7
\xbb\x9f (446 characters omitted)' ) [file "/etc/nginx/modsec/coreruleset-3.3.0/rules/REQUEST-941-APPLICATION-ATTAC
K-XSS.conf"] [line "527"] [id "941310"] [rev ""] [msg "US-ASCII Malformed Encoding XSS Filter - Attack Detected"] [
data "Matched Data: \xbc\x8c\xe6\x88\x91\xe4\xbb\xac\xe5\x8f\xaf\xe4\xbb\xa5\xe7\x94\xa8\xe5\x91\xbd\xe4\xbb\xa4\xe
8\xaf\xbb\xe5\x8f\x96<code> found within ARGS:comment: worker_rlimit_nofile\x0d\x0a\xe6\x9f\xa5\xe7\x9c\x8b\xe6\x93
\x8d\xe4\xbd\x9c\xe7\xb3\xbb\xe7\xbb\x9f\xe5\xaf\xb9\xe4\xb8\x80\xe4\xb8\xaa\xe8\xbf\x9b\xe7\xa8\x8b\xe6\x96\xbd\xe
5\x8a\xa0\xe7\x9a\x84\xe9\x99\x90\xe5\x88\xb6\xef\xbc\x8c\xe6\x88\x91\xe4\xbb\xac\xe5\x8f\xaf\xe4\xbb\xa5\xe7\x94\x
a8\xe5\x91\xbd\xe4\xbb\xa4\xe8\xaf\xbb\xe5\x8f\x96<code>/etc/$pid/limits< (78 characters omitted)"] [severity "2"]
[ver "OWASP_CRS/3.3.0"] [maturity "0"] [accuracy "0"] [tag "application-multi"] [tag "language-multi"] [tag "platfo
rm-tomcat"] [tag "attack-xss"] [tag "paranoia-level/1"] [tag "OWASP_CRS"] [tag "capec/1000/152/242"] [hostname "127
.0.0.1"] [uri "/wp-comments-post.php"] [unique_id "1609560027"] [ref "o71,35v1116,201t:urlDecodeUni,t:lowercase,t:u
rlDecode,t:htmlEntityDecode,t:jsDecode"]
```

It's detected as malformed encoding XSS filter attack. However, I want users to be able to use the `<code>...</code>` and <`pre>...</pre>` HTML tag in the comment form, so I created a rule exclusion. Add the following line at the bottom of the `REQUEST-900-EXCLUSION-RULES-BEFORE-CRS.conf` file.

```
SecRule REQUEST_URI "@streq /wp-comments-post.php" "id:1002,phase:1,ctl:ruleRemoveById
=941310"
```

This line tells ModSecurity that if the request URI is `/wp-comments-post.php`, then don't apply rule ID 941310. Save and close the file. Then test Apache configurations.

```
sudo apache2ctl -t
```

If the test is successful, restart Apache for the change to take effect.

```
sudo systemctl restart apache2
```

If a false positive matches multiple rule IDs, you can add rule exclusions in one line like so:

```
SecRule REQUEST_URI "@streq /wp-admin/post.php" "id:1003,phase:1,ctl:ruleRemoveById=94
1160,ctl:ruleRemoveById=941310,ctl:ruleRemoveById=942100"
```

Note: It's not recommended to disable too many rules of level 1 in the OWASP CRS, as it will make your website be hacked much more easily. Only disable rules if you know what you are doing.

IP Whitelisting

If you want to disable ModSecurity for your own IP address, but leave it enabled for all other IP addresses, then add the following custom rule in the `REQUEST-900-EXCLUSION-RULES-BEFORE-CRS.conf` file. Replace `12.34.56.78` with your real IP address.

```
SecRule REMOTE_ADDR "@ipMatch 12.34.56.78" "id:1004,phase:1,allow,ctl:ruleEngine=off"
```

To whitelist a subnet, use the following syntax, which will whitelist the `10.10.10.0/24` network.

```
SecRule REMOTE_ADDR "@ipMatch 10.10.10.10/24" "id:1005,phase:1,allow,ctl:ruleEngine=of
f"
```

Save and close the file. Then test Apache configurations.

```
sudo apache2ctl -t
```

If the test is successful, restart Apache for the change to take effect.

```
sudo systemctl restart apache2
```

Chaining Rules

If your Apache has multiple virtual hosts, you may want to whitelist your IP address for a specific virtual host. You need to chain two rules like so:

```
SecRule REMOTE_ADDR "@ipMatch 12.34.56.78" "id:1004,phase:1,allow,ctl:ruleEngine=off,c
hain"
SecRule REQUEST_HEADERS:Host "@streq nextcloud.yourdomain.com" "t:none"
```

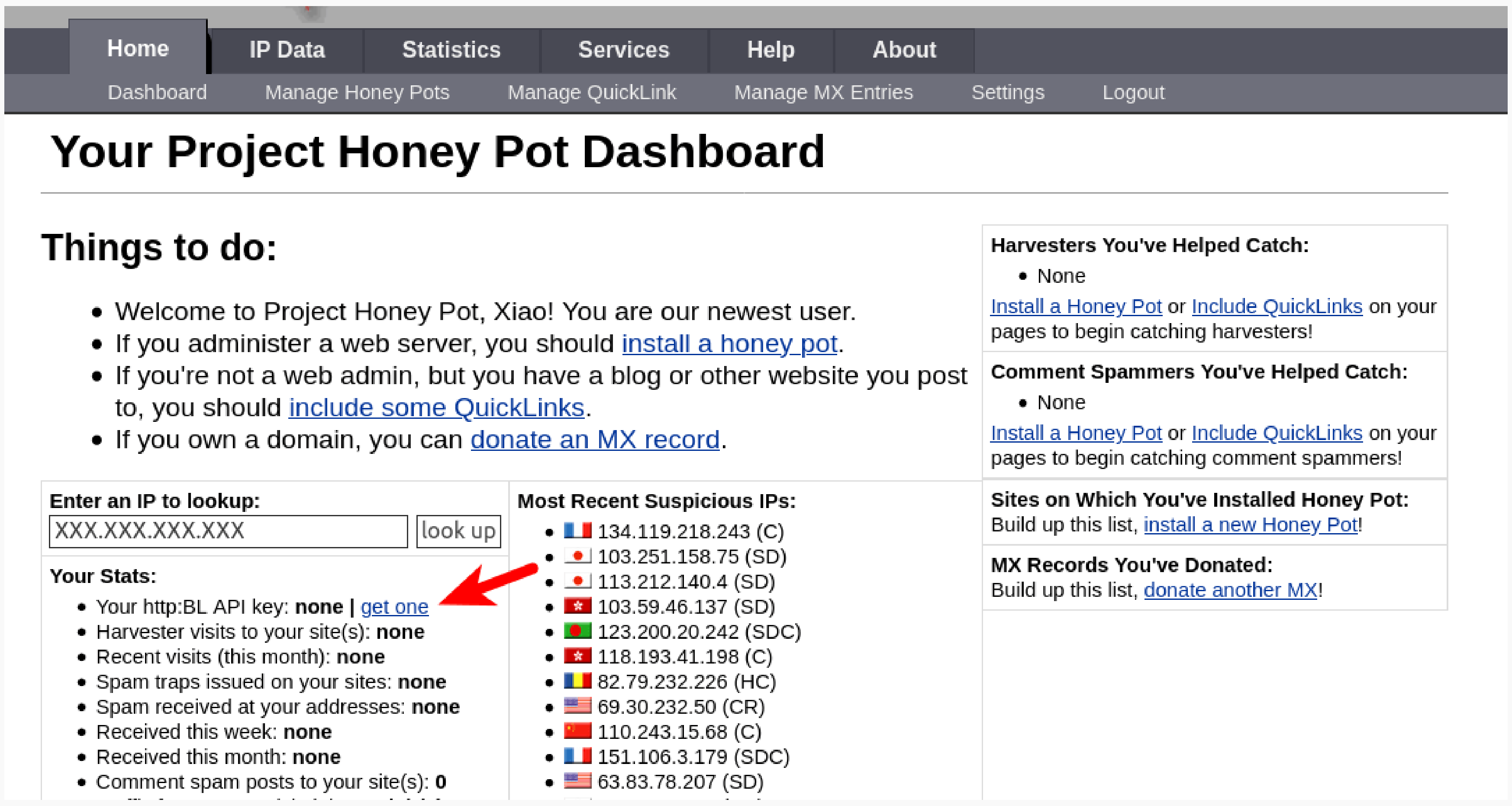
The `chain` keyword at the end of the first rule indicates that the `ruleEngine=off` action will only be taken if the condition in the next rule is also true.

(Optional) Integrate ModSecurity with Project Honeypot

Project Honeypot maintains a list of known malicious IP addresses, available free to the public. ModSecurity can integrates with Project Honeypot and block IP addresses on the Project Honeypot list.

Note that using Project Honeypot will make your website slower for new visitors, because your web server will need to send a query to Project Honeypot before it can send a response to the new visitor. However, once the IP reputation data is cached on your web server, the performance impact will be very minimal.

To use Project Honeypot, first create a free account on its website. Then go to your account dashboard and click the `get one` link to request an access key for the HTTP blacklist.



Your Project Honey Pot Dashboard

Things to do:

- Welcome to Project Honey Pot, Xiao! You are our newest user.
- If you administer a web server, you should install a honey pot.
- If you're not a web admin, but you have a blog or other website you post to, you should include some QuickLinks.
- If you own a domain, you can donate an MX record.

Enter an IP to lookup:
XXX.XXX.XXX.XXX

Your Stats:
- Your http:BL API key: **none** | get one
- Harvester visits to your site(s): **none**
- Recent visits (this month): **none**
- Spam traps issued on your sites: **none**
- Spam received at your addresses: **none**
- Received this week: **none**
- Received this month: **none**
- Comment spam posts to your site(s): **0**

Most Recent Suspicious IPs:
- 134.119.218.243 (C)
- 103.251.158.75 (SD)
- 113.212.140.4 (SD)
- 103.59.46.137 (SD)
- 123.200.20.242 (SDC)
- 118.193.41.198 (C)
- 82.79.232.226 (HC)
- 69.30.232.50 (CR)
- 110.243.15.68 (C)
- 151.106.3.179 (SDC)
- 63.83.78.207 (SD)

Harvesters You've Helped Catch:
- None

Install a Honey Pot or Include QuickLinks on your pages to begin catching harvesters!

Comment Spammers You've Helped Catch:
- None

Install a Honey Pot or Include QuickLinks on your pages to begin catching comment spammers!

Sites on Which You've Installed Honey Pot:
Build up this list, install a new Honey Pot!

MX Records You've Donated:
Build up this list, donate another MX!

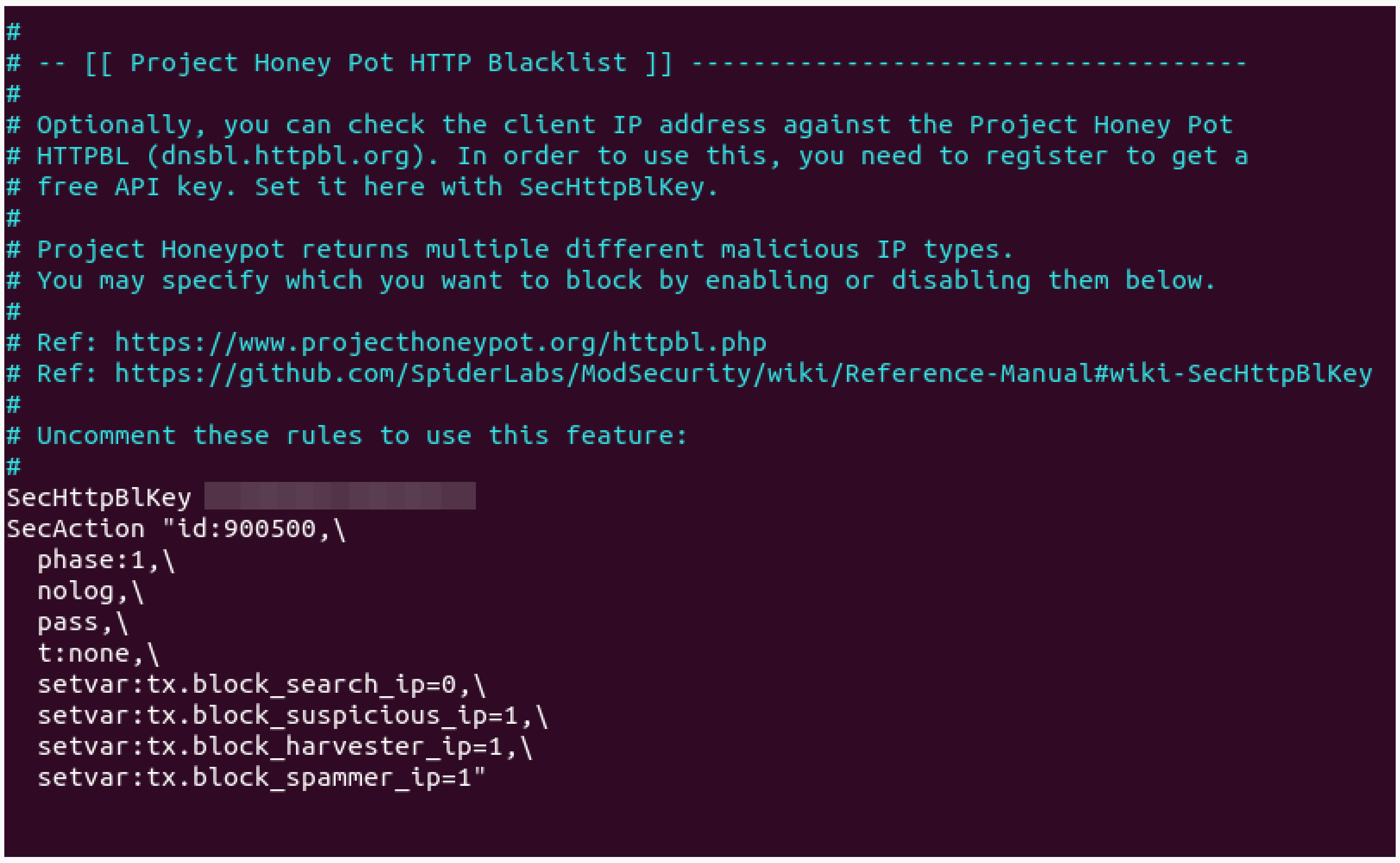
Next, edit the `crs-setup.conf` file.

```
sudo nano /etc/apache2/modsecurity-crs/coreruleset-3.3.0/crs-setup.conf
```

Find the following lines.

```
#SecHttpBlKey XXXXXXXXXXXXXXXXX
#SecAction "id:900500,\
#  phase:1,\
#  nolog,\
#  pass,\
#   t:none,\
#   setvar:tx.block_search_ip=1,\
#   setvar:tx.block_suspicious_ip=1,\
#   setvar:tx.block_harvester_ip=1,\
#   setvar:tx.block_spammer_ip=1"
```

Remove the beginning `#` characters to uncomment them, and add your HTTPBL API key obtained from Project Honeypot.



```
#
# -- [[ Project Honey Pot HTTP Blacklist ]] ------------------------------------
#
# Optionally, you can check the client IP address against the Project Honey Pot
# HTTPBL (dnsbl.httpbl.org). In order to use this, you need to register to get a
# free API key. Set it here with SecHttpBlKey.
#
# Project Honeypot returns multiple different malicious IP types.
# You may specify which you want to block by enabling or disabling them below.
#
# Ref: https://www.projecthoneypot.org/httpbl.php
# Ref: https://github.com/SpiderLabs/ModSecurity/wiki/Reference-Manual#wiki-SecHttpBlKey
#
# Uncomment these rules to use this feature:
#
SecHttpBlKey [redacted]
SecAction "id:900500,\
  phase:1,\
  nolog,\
  pass,\
  t:none,\
  setvar:tx.block_search_ip=0,\
  setvar:tx.block_suspicious_ip=1,\
  setvar:tx.block_harvester_ip=1,\
  setvar:tx.block_spammer_ip=1"
```

Note that `block_search_ip` should be set to `0` (disabled), as we don't want to block search engine crawlers. Save and close the file. Then restart Apache.

```
sudo systemctl restart apache2
```

Now ModSecurity will query Project Honeypot on all HTTP requests. To test if this would work, edit the crs-setup.conf file.

```
sudo nano /etc/apache2/modsecurity-crs/coreruleset-3.3.0/crs-setup.conf
```

In Nano text editor, you can quickly jump to the end of the file by pressing `Ctrl+W`, then `Ctrl+V`. Add the following line at the end of this file. This allows us to pass an IP address in an URL. (Once the test is successful, you can remove this line from the file.)

```
SecRule ARGS:IP "@rbl dnsbl.httpbl.org" "phase:1,id:171,t:none,deny,nolog,auditlog,ms
g:'RBL Match for SPAM Source'
```

Save and close the file. Test Apache configurations.

```
sudo apache2ctl -t
```

Then restart Apache.

```
sudo systemctl restart apache2
```

Go to Project Honeypot website and find a malicious IP address, for example, 134.119.218.243. Run the following command to test the HTTP blacklist.

```
curl -i -s -k -X $'GET' 'https://yourdomain.com/?IP=134.119.218.243'
```

Your Apache web server should return a 403 forbidden response because the IP address is on Project Honeypot.

How to Disable ModSecurity for a Virtual Host

By default, ModSecurity is enabled for all Apache virtual hosts. If you want to disable ModSecurity for a specific virtual host, then edit the virtual host file (`/etc/apache2/sites-enabled/example.com.conf`) and add the following line to the `<VirtualHost>...</VirtualHost>` context.

```
SecRuleEngine DetectionOnly
```

Reload Apache for the change to take effect.

```
sudo systemctl reload apache2
```

How to Upgrade OWASP CRS

You need to upgrade the core rule set when a new version comes out. The process is straightforward.

- Go through step 3 again to install the new version of core rule set.
- Then go to step 7. Copy of your custom rules in the `crs-setup.conf` and `REQUEST-900-EXCLUSION-RULES-BEFORE-CRS` file.

Next, test Apache configurations.

```
sudo apache2ctl -t
```

If the test is successful, restart Apache for the change to take effect.

```
sudo systemctl restart apache2
```

How do you know if the new version is working? Launch a simple SQL injection attack like in step 5 and check your server logs. It will show you the CRS version that’s preventing this attack.

Next Step

I hope this tutorial helped you set up ModSecurity web application firewall with Apache on Debian/Ubuntu. You may also want to check out other security tutorials.

- How to Use UFW Firewall on Debian, Ubuntu, Linux Mint
- Set Up Automatic Security Update (Unattended-Upgrade) on Ubuntu
- Canonical Livepatch Service: Patch Linux Kernel on Ubuntu without Reboot
- 2 Simple Steps to Set up Passwordless SSH Login on Ubuntu
- 5 Effective Tips to Harden SSH Server on Ubuntu
- How to Secure Email Server Against Hacking with VPN (Debian/Ubuntu)

As always, if you found this post useful, then subscribe to our free newsletter to get new tutorials ●

Rate this tutorial

★★★★★ [Total: 16 Average: 4.9]

Apache   Linux Server   ModSecurity   Web Application Firewall