



# **Deploying and Configuring a Web Application Firewall**

**CS 3061 -1 Computer Networks Security**

**Term Project Report - Fall 2023**

**Students: Leen Sharab (S21107195), Sarah Alshumayri (S20106125), Reema  
Abdallah (S20106463)**

**Instructor: Dr. Sohail Khan**

**Date Last Edited: December 24, 2023**

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Understanding Modsecurity</b>	<b>2</b>
2.1	What is ModSecurity? . . . . .	2
2.2	Core Features of ModSecurity: . . . . .	2
<b>3</b>	<b>Setting up the Environment</b>	<b>3</b>
3.1	Install and Configure Apache . . . . .	3
<b>4</b>	<b>Installing ModSecurity</b>	<b>4</b>
<b>5</b>	<b>Configure ModSecurity</b>	<b>5</b>
<b>6</b>	<b>Install the OWASP Core Rule Set (CRS)</b>	<b>6</b>
<b>7</b>	<b>Learn How OWASP CRS Works</b>	<b>7</b>
<b>8</b>	<b>Testing</b>	<b>9</b>
<b>9</b>	<b>Conclusion</b>	<b>10</b>

# 1 Introduction

In an era where digital security is crucial, web applications are continuously under the threat of sophisticated cyber-attacks. This is where a "firewall" becomes a critical tool in the cybersecurity arsenal. A firewall acts as a security guard, standing between the vast ocean of internet traffic and your digital assets. It monitors network traffic, both incoming and outgoing, and decides whether to allow or block specific traffic based on a defined set of security rules.

This is where ModSecurity, a robust web application firewall (WAF), comes into play. **ModSecurity** is an open-source firewall application for Apache web servers. It provides a protective layer that monitors, logs, and filters incoming traffic to a web server, blocking potentially harmful requests.

This comprehensive guide is crafted to assist you in integrating ModSecurity with the standalone Apache web server.

To set the stage for a successful integration, ensure you have the following:

- An operational Ubuntu machine with administrative rights.
- A website to test the firewall on.
- Apache2 server installed and actively running without issues.

## 2 Understanding Modsecurity

### 2.1 What is ModSecurity?

ModSecurity is an open-source Web Application Firewall (WAF) that provides a layer of security for web servers. It is designed to protect web applications from various security threats by monitoring HTTP traffic and executing predefined rules to block potentially harmful requests. It's most commonly used with Apache, but it can work with other web servers too.

### 2.2 Core Features of ModSecurity:

1. **Real-Time Monitoring and Logging:** Tracks HTTP traffic to and from a web server in real-time, providing detailed logs that can be crucial for identifying and understanding threats.
2. **Customizable Rule Engine:** Allows administrators to define rules for detecting and blocking malicious traffic, offering flexibility to tailor the security measures to specific needs.
3. **Data Leakage Prevention:** Helps in preventing sensitive information, like credit card details or personal data, from being leaked or exposed.
4. **Attack Mitigation:** Protects against various web application attacks such as SQL injection, Cross-Site Scripting (XSS), and others.
5. **Request and Response Filtering:** Inspects and modifies both incoming requests and outgoing responses, ensuring only legitimate traffic is processed.

Let's proceed to establish a secure fortress for your web applications with the help of ModSecurity.

## 3 Setting up the Environment

### 3.1 Install and Configure Apache

Before you begin, it's a good practice to update your package repository to ensure you get the latest available version of Apache2. Open your terminal and run:

```
1 sudo apt update
```

Then, install the apache2 package by running:

```
1 sudo apt install apache2
```

After the installation is complete, you can check if Apache2 is running with:

```
1 sudo systemctl status apache2
```

To access your web page, you should navigate to the `/var/www/html/` directory. This is the default directory where Apache searches for files to display on your website. By placing your web files in this directory, they become accessible through 'localhost' in your web browser. Begin by navigating to the `/var` Directory:

```
1 cd /var/
```

Move to the `www` Directory:

```
1 cd www
```

Enter the `html` Directory:

```
1 cd html/
```

- This places you in the **html** directory, which is typically the default document root for the Apache web server.
- The path `/var/www/html/` is where you'll place the web files (like HTML, PHP, CSS files) that you want Apache to serve.

To execute your website on localhost, begin by navigating to the `/var/www/html` directory:

```
1 cd /var/www/html
```

Create a PHP script. We'll name it "test.php" in this example:

```
1 sudo nano test.php
```

Inside the "test.php" file, add the following PHP code that will display "Hello, World!" when executed then save and exit:

```
1 <?php
2 echo "Hello, World!";
3 ?>
```

To make the "test.php" file executable, use the `chmod` command with the `+x` option:

```
1 sudo chmod +x test.php
```

Now you can access your webpage by entering this link in the browser:

```
1 HTTP://localhost/test.php
```

## 4 Installing ModSecurity

The ModSecurity module is available in the default Ubuntu repository. To install it, open your terminal and enter the following command:

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

This command installs the ModSecurity module on your Apache server.

After installing the module, you need to enable it to activate its features. Run the following command to enable the ModSecurity module:

```
1 sudo a2enmod security2
```

For the changes to take effect, you must restart the Apache server. Restart Apache by executing:

```
1 sudo systemctl restart apache2
```

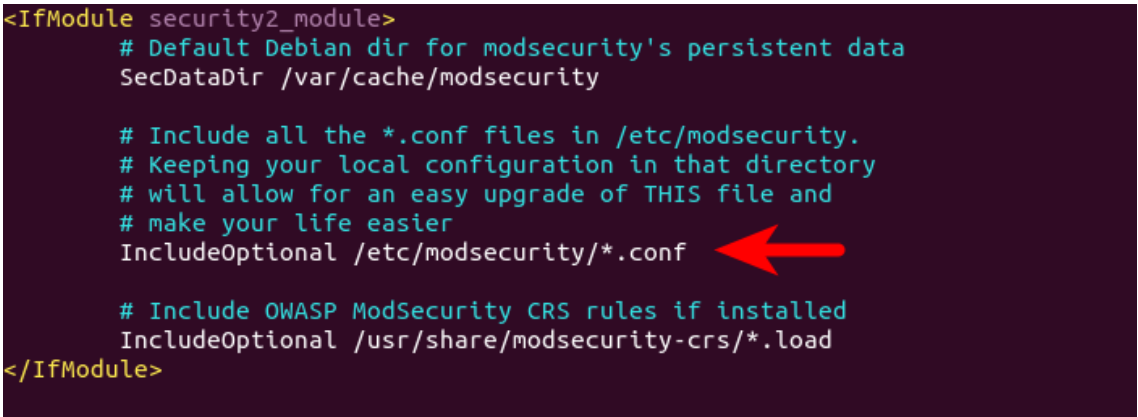
## 5 Configure ModSecurity

You need to access the `/etc/apache2/mods-enabled/security2.conf` by executing:

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

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

```
1 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:

```
1 sudo mv /etc/modsecurity/modsecurity.conf-recommended /etc/modsecurity/modsecurity.conf
```

Then open the file and modify it with a command-line text editor like Nano:

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

Find the following line:

```
1 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:

```
1 SecRuleEngine On
```

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

```
1 SecAuditLogParts ABDEFHIJZ
```

However, the default setting is wrong. The setting should be changed to the following:

```
1 SecAuditLogParts ABCEFHKZ
```

Save and close the file. Then restart Apache for the change to take effect:

```
1 sudo systemctl restart apache2
```

## 6 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. 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 solid default configuration for ModSecurity.
- It contains rules to help stop common attack, including SQL injection (SQLi), cross-site scripting (XSS), and many others.
- It also contains rules to detect bots and scanners.
- It has been tuned through wide exposure to have very few false positives.

Download the latest OWASP CRS from GitHub:

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

Extract the file:

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

Create a directory to store CRS files:

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

Move the extracted directory to /etc/apache2/modsecurity-crs/:

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

Go to that directory:

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

Rename the crs-setup.conf.example file:

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

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

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

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

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

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

```
1 IncludeOptional /etc/apache2/modsecurity-crs/coreruleset-3.3.0/crs-setup.conf
2 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 /usr/share/modsecurity-crs/*.load
</IfModule>
```

Save and close the file. Then test Apache configuration:

```
1 sudo apache2ctl -t
```

If the syntax is OK, then restart Apache:

```
1 sudo systemctl restart apache2
```

## 7 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:

```
1 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.
#
```

Now you can close the file.

## 8 Testing

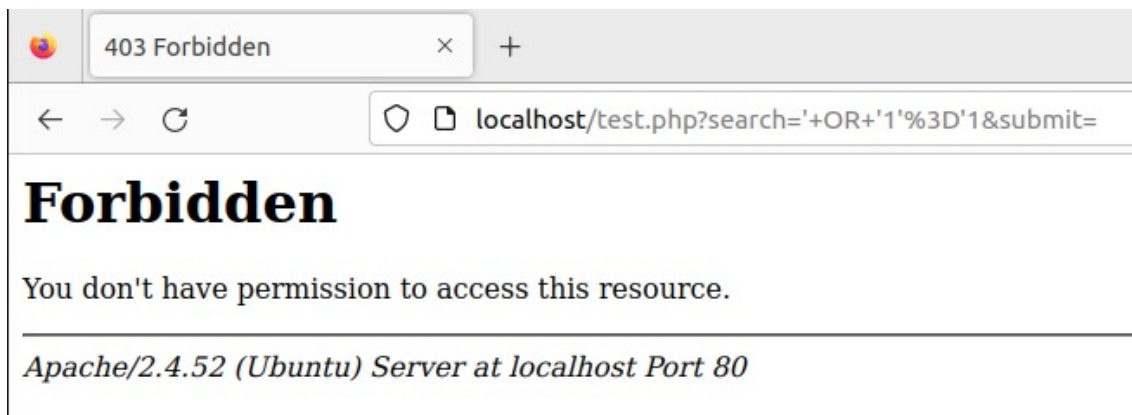
To check if ModSecurity is working, you can launch a simple SQL injection attack on your own website. Enter the following URL in your web browser:

```
1 https://yourdomain.com/?id=3 OR '1'='1'
```

In our case, we inserted the statement in the search box:



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

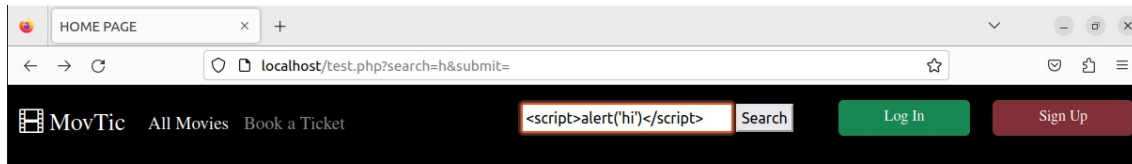


And in the audit log `/var/log/apache2/modsec_audit.log`, you can see the following line in section H, which means ModSecurity detected and blocked this SQL injection attack by using OWASP CRS v3.3.0.:

```
1 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"
```

For extra testing, you can also launch a simple Cross-site scripting(xss) attack on your own website:



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



## 9 Conclusion

As we conclude, it is important to remember that web security is a continuous process, not a one-time setup. Regular updates, vigilant monitoring, and an ongoing commitment to security best practices are the keystones to ensuring the safety and integrity of your web applications. The steps outlined in this guide are just the beginning of a comprehensive security strategy that should include regular audits, updates, and an awareness of the latest security trends and threats.