ModSecurity and nginx | Linux Journal

by Elliot Cooper

13-17 minuti

---

nginx is the web server that's replacing Apache in more and more of the world's websites. Until now, nginx has not been able to benefit from the security ModSecurity provides. Here's how to install ModSecurity and get it working with nginx.

Earlier this year the popular open-source web application firewall, ModSecurity, released version 3 of its software. Version 3 is a significant departure from the earlier versions, because it's now

modularized. Before version 3, ModSecurity worked only with the Apache web server as a dependent module, so there was no way for other HTTP applications to utilize ModSecurity. Now the core functionality of ModSecurity, the HTTP filtering engine, exists as a standalone library, libModSecurity, and it can be integrated into any other application via a "connector". A connector is a small piece of code that allows any application to access libModSecurity.

A Web Application Firewall (WAF) is a type of firewall for HTTP requests. A standard firewall inspects data packets as they arrive and leave a network interface and compares the properties of the packets against a list of rules. The rules dictate whether the firewall will allow the packet to pass or get blocked.

ModSecurity performs the same task as a standard firewall, but instead of looking at data packets, it inspects HTTP traffic as it

arrives at the server. When an HTTP request arrives at the server, it's first routed through ModSecurity before it's routed on to the destination application, such as Apache2 or nginx. ModSecurity compares the inbound HTTP request against a list of rules. These rules define the form of a malicious or harmful request, so if the incoming request matches a rule, ModSecurity blocks the request from reaching the destination application where it may cause harm.

The following example demonstrates how ModSecurity protects a WordPress site. The following HTTP request is a non-malicious request for the index.php file as it appears in Apache2's log files:

```
GET /index.php HTTP/1.1
```

This request does not match any rules, so ModSecurity allows it

onto the web server.

WordPress keeps much of its secret information, such as the database password, in a file called wp-config.php, which is located in the same directory as the index.php file. A careless system administrator may leave this important file unprotected, which means a web server like Apache or nginx happily will serve it. This is because they will serve any file that is not protected by specific configuration. This means that the following malicious request:

```
GET /wp-config.php HTTP/1.1
```

will be served by Apache to whomever requests it.

This is where ModSecurity offers protection to an application accepting HTTP data. In this case, the free, core ModSecurity

ruleset contains rules to deny any HTTP request that attempts to access any sensitive file in a WordPress installation. The core ruleset also contains rules for another popular CMS, Drupal.

The core ruleset also contains rules covering the many other ways that HTTP requests can be constructed maliciously to gain access or confidential information from a website. These methods include SQL injection, vulnerability scanning, Java and PHP exploits and many more. ModSecurity also supports custom rules, so you can protect your HTTP application against specifically targeted attacks by writing your own rules.

First let's install the core ModSecurity library, libModSecurity, and then let's install the nginx connector that enables nginx to use ModSecurity. Before version 3, it wasn't possible to use ModSecurity with nginx. If you are using Apache2, you should

continue to use ModSecurity version 2, as the Apache2 connector is still quite buggy and not recommended for production use.

Compiling and Installing libModSecurity

ModSecurity3 isn't available via the package manager for any of the major Linux distributions. Instead, you'll need to clone the ModSecurity GitHub repository and build the library from its source code. Before you can do that though, you must install all of the required build tools and dependencies. The following list of packages provides all of the required and most of the optional discrepancies on Debian and Ubuntu distributions: bison, flex, make, automake, gcc, pkg-config, libtool, doxygen, git, curl, zlib1g-dev, libxml2-dev, libpcre3-dev, build-essential, libyajl-dev, yajl-tools, liblmdb-dev, rdmacm-utils, libgeoip-dev, libcurl4-openssl-dev,

liblua5.2-dev, libfuzzy-dev, openssl and libssl-dev.

Note that some of those packages have different names on Red Hat-based distributions. This page will help you figure out what the specific package names are.

After installing those packages, you can move on to compiling the library. These instructions are distribution-agnostic.

First, clone the libModSecurity git repository, which will download all the source code you need to build the libModSecurity. Use the /opt/ directory as the destination for all source code. Move to the /opt/ directory, and clone the libModSecurity git repository with the following commands:

```
cd /opt/
git clone https://github.com/SpiderLabs
```

```
/ModSecurity
```

Next, move into the new directory that you created when you cloned the ModSecurity repository, and switch to the v3 branch. You'll also need to pull in a couple necessary sub-modules:

```
cd ModSecurity
git checkout v3/master
git submodule init
git submodule update
```

You're now ready to build libModSecurity. This should be a familiar process to anyone who has compiled a program from source code. You need only the following three commands to compile and install

the library:

```
sh build.sh
./configure
make
make install
```

The `make` command takes a few minutes if you are running this on a modest virtual server. The libModSecurity library now is installed at /usr/local/modsecurity/lib/libmodsecurity.so. However, it can't do anything until you install an application and connector that will redirect the HTTP data to the libModSecurity library along with some rules. The next section looks at installing the nginx connector and the core ruleset provided by the ModSecurity developers.

Compiling the nginx Connector

Let's compile the nginx connector by utilizing nginx's capability of dynamic loading of third-party modules. nginx has been able to do this since version 1.11.5. This version or one higher is not available from the standard repositories of most of the major distributions. nginx provides repositories for the current stable releases of Red Hat/CentOS, Debian and Ubuntu that contain a version that supports dynamic module loading. This page lists these repositories along with information for adding the nginx repository to your distribution. After you have added the nginx repository to your repository configuration, you need to install nginx using your package manager. When you have installed nginx, find the version you installed with this command:

```
nginx -v
```

When you have the version number, change to the /opt/ directory and download the source code that matched your nginx version from this page, and unpack the archive that you downloaded.

Next, you need to clone the git repository for the ModSecurity nginx connector. From the /opt/ directory, run the following command to clone this repository:

```
git clone https://github.com/SpiderLabs
/ModSecurity-nginx
```

Now change into the new directory that you created when you unpacked the nginx source archive. In that directory, run the

following commands to compile the connector:

```
./configure --with-compat
 ↪--add-dynamic-module=/opt/ModSecurity-nginx
make modules
```

Now you need to copy the connector module into the nginx modules directory with this command:

```
cp objs/ngx_http_modsecurity_module.so /etc/nginx
/modules/
```

Now that you've compiled the nginx connector and copied it to the right location, you need to configure nginx to use it. In addition, you

also need to download the rules that libModSecurity will use to filter the HTTP data.

First, move to the nginx configuration directory:

```
cd /etc/nginx/
```

and add the following line to the nginx's main configuration file at /etc/nginx/nginx.conf:

```
load_module
modules/ngx_http_modsecurity_module.so;
```

You need to put this line in the first section under the line that

begins `pid` and not in either the `events` or `http` sections.

Next, create a new directory and load the ModSecurity rules and configuration into it:

```
mkdir /etc/nginx/modsec
cd /etc/nginx/modsec
git clone https://github.com/SpiderLabs/
↪owasp-modsecurity-crs.git
```

Use the ModSecurity rules configuration file that was downloaded from the git repository by renaming it with the following command:

```
mv /etc/nginx/modsec/owasp-modsecurity-crs/
```

```
↪crs-setup.conf.example /etc/nginx/modsec/
↪owasp-modsecurity-crs/crs-setup.conf
```

Now you need to copy the ModSecurity configuration file from the directory where you built libModSecurity to /etc/nginx/modsec/:

```
cp /opt/ModSecurity/modsecurity.conf-recommended
 ↪/etc/nginx/modsec/modsecurity.conf
```

Finally, create a new configuration file that loads these two configuration files and all the rules files. This file will be invoked by a couple lines in an nginx server configuration block, which will invoke the use of ModSecurity. Create and start editing this file with a text editor:

```
nano /etc/nginx/modsec/main.conf
```

Add the following three lines to this file:

```
Include /etc/nginx/modsec/modsecurity.conf
Include /etc/nginx/modsec/owasp-modsecurity-
crs/crs-setup.conf
Include /etc/nginx/modsec/owasp-modsecurity-
crs/rules/*.conf
```

You've now completed building and installing nginx, libModSecurity, the nginx connector and ModSecurity rules. Now you can start or re-start nginx to load the new configuration. If everything is working,

you won't see any errors printed when you restart nginx.

Testing ModSecurity

Let's test ModSecurity by adding a couple lines to the "default" server and making a request that will be blocked by ModSecurity. The default server configuration is the configuration that nginx uses on installation and is listening only on localhost and not on the internet-facing network interface. This makes it secure to start nginx before any custom server configuration has been created, because the default configuration is inaccessible from the internet.

The default server configuration is located at /etc/nginx/conf.d /default.conf. Open this file with a text editor, and add the following two lines under the `server_name` line:

```
modsecurity on;
modsecurity_rules_file /etc/nginx/modsec
/main.conf;
```

Restart nginx again to load this new configuration. Now, all you need to do to test that ModSecurity is working is make an HTTP request that matches a banned rule.

ModSecurity has two modes of operation. The default is that it will only log any queries that match banned rules but allow them to pass to the application. This mode allows system administrators to run ModSecurity for a period and ensure that no false positive requests are getting blocked that would interfere with the normal operation of the website. ModSecurity records these requests that match banned rules to /var/log/modsec_audit.log.

You can create an HTTP request that will be recorded to that log file by using `curl` to make a request that contains a banned user agent header. The following command makes an HTTP request that contains the header "User-Agent: masscan". This is a banned user agent, so ModSecurity records that it witnessed a banned HTTP request. This command looks like:

```
curl -H "User-Agent: masscan" http://localhost/
```

nginx returns the default welcome page as raw HTML, but ModSecurity creates a lengthy log entry in /var/log /modsec_audit.log that begins:

```
ModSecurity: Warning. Matched "Operator
```

```
`PmFromFile'
 ↪with parameter `scanners-user-agents.data'
against
 &rarrhkk;variable `REQUEST_HEADERS:User-Agent'
(Value: `masscan' )
```

This indicates that ModSecurity successfully intercepted and matched the malicious HTTP request.

When you want to toggle ModSecurity from logging malicious HTTP requests to blocking them actively, edit the line in /etc/nginx/modsec /modsecurity.conf from:

```
SecRuleEngine DetectionOnly
```

to:

```
SecRuleEngine On
```

and restart nginx. Now the same `curl` request will result in a 403 error:

```
curl -H "User-Agent: masscan" http://localhost/
<html>
<head><title>403 Forbidden</title></head>
<body bgcolor="white">
<center><h1>403 Forbidden</h1></center>
<hr><center>nginx/1.12.2</center>
</body>
```

```
</html>
```

The blocked request also will be logged to /var/log /modsec_audit.log.

Additional ModSecurity Connectors

The new modular nature of ModSecurity means that any application that accepts or processes HTTP data can use ModSecurity and its rules to analyze HTTP data. At the time of this writing, ModSecurity v3 has been of release quality only for a few months, so there aren't many additional connectors that enable applications to hook into libModSecurity.

The Google Summer of Code has produced a couple interesting new connectors. The first extends the ability of Snort v3 to use the

ModSecurity rules. Snort is an intrusion-detection and real-time packet-sniffing and logging application. This connector allows Snort to send captured HTTP data to libModSecurity and get it checked against the ModSecurity ruleset. The home page for this project is here.

A second Google-sponsored connector targets the node.js server. Node.js is a JavaScript runtime that enables the creation of scalable network applications. This connector routes all inbound HTTP requests via ModSecurity, thereby adding a security layer to the Node application. You can read more about this project at its home page.

The release of ModSecurity v3 has transformed ModSecurity from an Apache module to a flexible application that is easily leveraged by any application that accepts HTTP data. Given that more and

more of the applications that people depend on are moving from their local computers into data centers, the need to ensure the security of those applications and that data is becoming increasingly important.