MODSECURITY 3.0 & NGINX: Quick Start Guide

NGINX

MODSECURITY 3.0 & NGINX: Quick Start Guide

by Faisal Memon, Owen Garrett, and Michael Pleshakov

NGINX

Table of Contents

Preface

For a very long time, the NGINX community waited for ModSecurity. The popular plug-in for the Apache HTTP server seemed like a natural fit to be ported over to NGINX, but it's never as easy as it sounds to port something over to a different platform. In 2017, the wait finally ended, and the long-awaited ModSecurity 3.0 was finally released. ModSecurity 3.0 is the first version of ModSecurity to work natively with NGINX.

ModSecurity is an open source web application firewall (WAF). ModSecurity protects applications against a broad range of Layer 7 attacks, such as SQL injection (SQLi), local file inclusion (LFI), and cross-site scripting (XSS). These three attack vectors together accounted for 95% of known Layer 7 attacks in Q1 2017, according to the report, Akamai State of the Internet – Security. The report also states that the purpose of attacks varies, but the majority are aimed at stealing data.

ModSecurity is used by over a million websites today, making it the world's most widely deployed WAF. ModSecurity is open source and community-driven, which has led to its broad adoption by organizations ranging from small businesses to governments and the enterprise. It uses a rules language based on the Perl Compatible Regular Expression (PCRE) standard, which makes it much more accessible than many other WAFs.

This ebook was created to help you get up and running with ModSecurity 3.0 and NGINX as easily as possible. It covers how to install, enable the major features of, and troubleshoot ModSecurity 3.0. Please use other resources available online, including the brilliant ModSecurity Handbook, if you need more information about ModSecurity before you put it into production.

I hope this guide will be helpful to you in your journey with ModSecurity 3.0 and NGINX.

–Faisal Memon
Product Marketer, NGINX, Inc.

1 Introduction

". . . even when you understand web security it is difficult to produce secure code, especially when working under the pressure so common in today's software development projects."

–Ivan Ristić, ModSecurity creator

A Brief History of ModSecurity

ModSecurity was originally created by Ivan Ristić to help protect several web applications he was responsible for at the time. He envisioned creating a software solution that would sit in front of these applications, analyzing the data flow in and out. A solution like this could both provide visibility and block potential attacks.

Ristić then set out to create this solution as a hobbyist side project. He created it as a plugin for the Apache HTTP Server. By November 2002 he put out the first open source release, calling his work ModSecurity. ModSecurity quickly gained in popularity, as it provided instant value as a tool for protecting web-based applications.

From there, events followed:

- 2004: ModSecurity commercialized as Thinking Stone; Ristić quits day job to focus fulltime on ModSecurity
- 2006: Thinking Stone acquired by Breach Security
- 2006: ModSecurity 2.0 released
- 2009: Ivan Ristić leaves Breach Security and the ModSecurity project

- 2010: Breach Security acquired by TrustWave; additional releases follow
- 2017: ModSecurity 3.0 released with support for NGINX and NGINX Plus

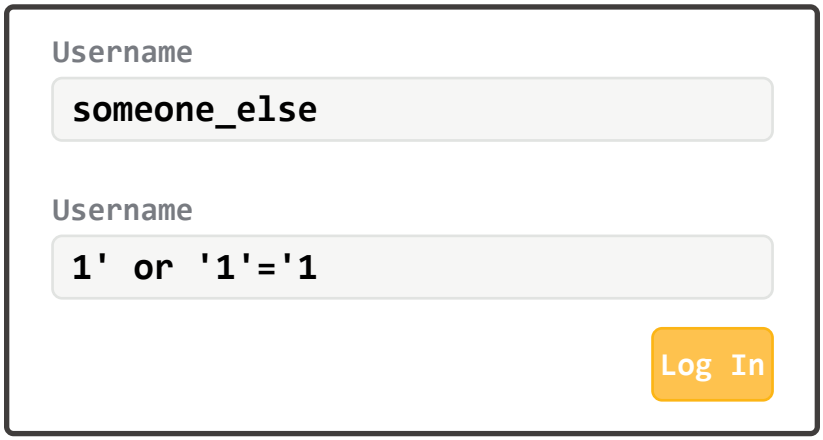
Today, Trustwave continues to be the corporate sponsor of ModSecurity, funding development of the open source project while also offering a commercial rule set.

How ModSecurity Works

As a web application firewall (WAF), ModSecurity is specialized to focus on HTTP traffic. When an HTTP request is made, ModSecurity inspects all parts of the request for any malicious content or anomalies in the traffic. If the packet is deemed malicious it can be blocked, logged, or both, depending on configuration.

ModSecurity uses a database of "rules" that define malicious behaviors. ModSecurity 3.0 supports both the OWASP ModSecurity Core Rule Set (CRS), the most widely used, open source rule set for ModSecurity, and the TrustWave commercial rule set, for which users pay a fee. The OWASP CRS is community-maintained and has been been tuned through wide exposure to have very few false positives.

SQL injection is one of the most common exploits, accounting for over 51% of known web application attacks, according to the report, Akamai State of the Internet – Security. This type of attack exploits improper validation of user input fields, such as forms. As a simple example, consider an attacker entering the following into a login form (the password field is unmasked for clarity):



If a web application uses the information entered on the form directly to form a SQL query for retrieving user information from a database, without first validating the password, it generates an SQL statement like this:

```
SELECT * FROM Users WHERE Username='someone_else' AND
Password='1' OR '1' = '1';
```

Because the statement `'1' = '1'` is always true, this query bypasses the password check and allows the attacker to log in as `someone_else` (or any other account).

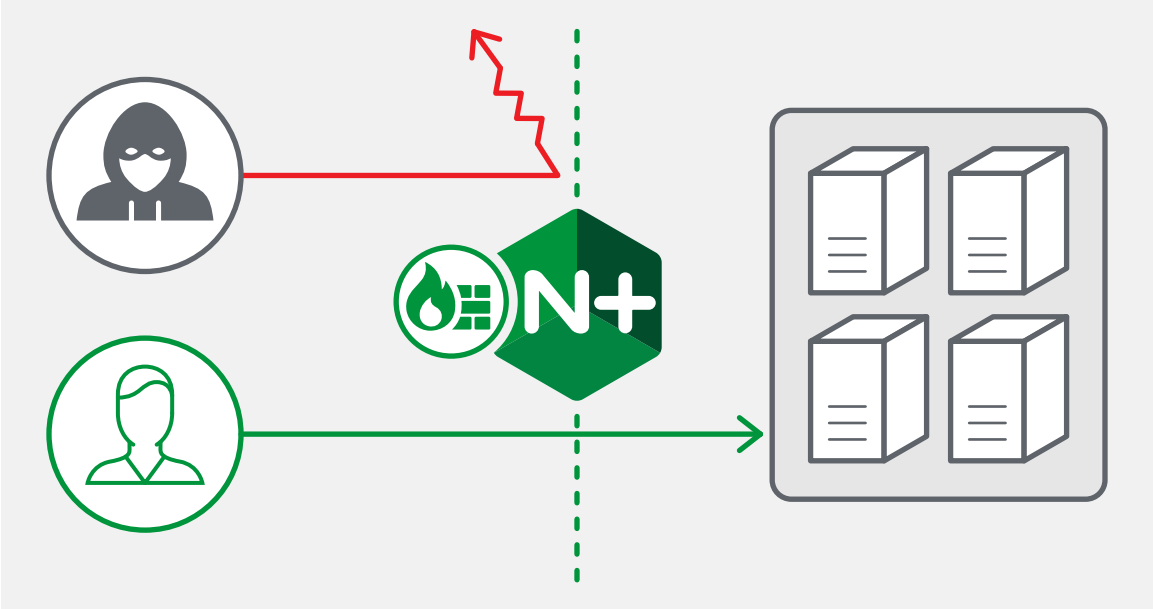
The CRS SQL injection rules in the OWASP CRS test input parameters and cookies for all requests against this and other patterns that attackers use to insert malicious SQL queries into forms. If a request matches any of the SQL injection rules, ModSecurity can drop the packet and/or log it, as configured.

There are similar rules in the CRS to detect and stop other common attacks, such as XSS and LFI.

What's Changed with ModSecurity 3.0

Previous versions of ModSecurity, up through version 2.9, did technically work with NGINX, but suffered from poor performance. This is because ModSecurity was wrapped inside a full version of the Apache HTTP Server, which provided a compatibility layer. ModSecurity was very heavily tied to Apache. ModSecurity 3.0, however, is a complete rewrite of ModSecurity that works natively with NGINX without requiring Apache.

This new architecture in version 3.0 moves the core functionality into a stand-alone engine called `libmodsecurity`. The `libmodsecurity` component interfaces with NGINX through an optimized connector called the NGINX connector. There is also a separate connector for Apache.



ModSecurity 3.0 is a dynamic module for NGINX and NGINX Plus

The combined package of **`libmodsecurity`** along with the NGINX connector create the ModSecurity dynamic module for NGINX. For NGINX Plus, the dynamic module is provided for you, already compiled. For NGINX open source, you have to compile the ModSecurity source code.

The new module is considerably faster and more stable than the legacy module has been with open source NGINX, and is actively supported by the NGINX, Inc. and Trustwave teams.

Caveats

ModSecurity 3.0 does not yet have feature parity with the previous version, ModSecurity 2.9. Please be aware of the following limitations:

- Rules that inspect the response body are not supported and are ignored if included in the configuration. The NGINX **`sub_filter`** directive can be used to inspect and rewrite response data.
- Inclusion of the request and response body in the audit log is not supported
- Some directives are not implemented; you may get an error if you try to use them. The ModSecurity Reference Manual lists all the available directives in ModSecurity and whether or not they are supported in **`libModSecurity`**.

There is no set date for ModSecurity 3.0 to reach feature parity with ModSecurity 2.9, but it is under active development, and each new release reduces the gap.

2 Installing ModSecurity

"Web applications – yours, mine, everyone's – are terribly insecure on average. We struggle to keep up with the security issues and need any help we can get to secure them."

–Ivan Ristić, ModSecurity creator

This chapter will cover how to install ModSecurity 3.0 for both open source NGINX and NGINX Plus. Open source NGINX users will have to compile ModSecurity from source. NGINX Plus users benefit from using the precompiled ModSecurity 3.0 dynamic module provided by NGINX Inc.

Benefits of using ModSecurity 3.0 with NGINX Plus

ModSecurity 3.0 for NGINX Plus is known as the NGINX WAF. There are a number of benefits to a commercial subscription:

- You don't need to compile the ModSecurity dynamic module yourself, NGINX, Inc. provides a precompiledmodule for you, saving time and effort
- NGINX, Inc. has extensively tested the dynamic module, so you know it's suitable for production usage
- NGINX, Inc. continually tracks changes and updates the module for every important change and security vulnerability, so you don't have to do this yourself
- Each new release of NGINX Plus includes a new version of the dynamic module, so you can upgrade without having to re-compile ModSecurity
- 24x7 support to help you with both installation of the ModSecurity and the OWASP Core Rule Set; as well as troubleshooting and debugging assistance

Open Source NGINX Installation Instructions

Installation Overview

In NGINX 1.11.5 and later, you can compile individual dynamic modules without compiling the complete NGINX binary. After covering the compilation process step by step, we'll explain how to load the ModSecurity dynamic module into NGINX and run a basic test to make sure it's working.

1 – Install NGINX from Our Official Repository

If you haven't already, the first step is to install NGINX. There are multiple ways to install NGINX, as is the case with most open source software. We generally recommend you install NGINX from the mainline branch in our official repository. For instructions, please see: nginx.org/en/linux_packages.html#mainline.

The instructions in this chapter assume that you have installed NGINX from our official repository. They might work with NGINX as obtained from other sources, but that has not been tested.

Note: NGINX 1.11.5 or later is required.

2 – Install Prerequisite Packages

The first step is to install the packages required to complete the remaining steps in this tutorial. Run the following command, which is appropriate for a freshly installed Ubuntu/Debian system. The required packages might be different for RedHat Enterprise/CentOS/Oracle Linux:

```
$ apt-get install -y apt-utils autoconf automake build-essential
git libcurl4-openssl-dev libgeoip-dev liblmdb-dev libpcre++-dev
libtool libxml2-dev libyajl-dev pkgconf wget zlib1g-dev
```

3 – Download and Compile `libmodsecurity`

With the required prerequisite packages installed, the next step is to compile ModSecurity as an NGINX dynamic module. In ModSecurity 3.0's new modular architecture, **`libmodsecurity`** is the core component which includes all rules and functionality. The second main component in the architecture is a connector that links **`libmodsecurity`** to the web server it is running with. There are separate connectors for NGINX and Apache HTTP Server. We cover the NGINX connector in the next section.

To compile `libmodsecurity`:

A. Clone the GitHub repository.

B. Change to the ModSecurity directory and compile the source code:

```
$ git clone --depth 1 -b v3/master --single-branch
https://github.com/SpiderLabs/ModSecurity

$ cd ModSecurity
$ git submodule init
$ git submodule update
$ ./build.sh
$ ./configure
$ make
$ make install
```

The compilation takes about 15 minutes, depending on the processing power of your system.

Note: It's safe to ignore messages like the following during the build process. Even when they appear, the compilation completes and creates a working object:

```
fatal: No names found, cannot describe anything.
```

4 – Download the NGINX Connector for ModSecurity and Compile it as a Dynamic Module

Compile the ModSecurity connector for NGINX as a dynamic module for NGINX.

A. Clone the GitHub repository:

```
$ git clone --depth 1 https://github.com/SpiderLabs/ModSecurity-
nginx.git
```

B. Determine which version of NGINX is running on the host where the ModSecurity module will be loaded:

```
$ nginx -v
nginx version: nginx/1.13.7
```

C. Download the source code corresponding to the installed version of NGINX (the complete sources are required even though only the dynamic module is being compiled):

```
$ wget http://nginx.org/download/nginx-1.13.7.tar.gz
$ tar zxvf nginx-1.13.7.tar.gz
```

D. Compile the dynamic module and copy it to the standard directory for modules:

```
$ cd nginx-1.13.7
$ ./configure --with-compat --add-dynamic-module=../ModSecurity-nginx
$ make modules
$ cp objs/ngx_http_modsecurity_module.so /etc/nginx/modules
```

5 – Load the NGINX ModSecurity Connector Dynamic Module

Add the following `load_module` directive to the main (top-level) context in **/etc/nginx/nginx.conf**. It instructs NGINX to load the ModSecurity dynamic module when it processes the configuration:

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

NGINX Plus Installation Instructions

This section explains how to install the NGINX web application firewall (WAF). The NGINX WAF is built on ModSecurity 3.0.

The NGINX WAF is available to NGINX Plus customers as a downloaded dynamic module at an additional cost. To purchase or start a free trial of NGINX WAF, or add the NGINX WAF to an existing NGINX Plus subscription, contact the NGINX sales team at nginx-inquires@nginx.com.

Prerequisites

Install NGINX Plus R12 or later following these instructions: nginx.com/resources/admin-guide/installing-nginx-plus/.

Installing the NGINX WAF

To install the dynamic module for the NGINX WAF, perform the following steps.

1. Use the OS package-management utility to install the dynamic module from the NGINX Plus module repository. The following command is appropriate for Debian and Ubuntu systems. For systems that use RPM packages, substitute the `yum install` command:

```
$ sudo apt-get install nginx-plus-module-modsecurity
```

2. Add the following line in the top-level ("main") context of **/etc/nginx/nginx.conf**:

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

3. Run the following command to verify that the module loads successfully, as confirmed by the indicated output:

```
$ sudo nginx -t
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
```

Example: Configuring ModSecurity 3.0 with a Simple Rule

In this example we configure a simple ModSecurity rule to block certain requests to a demo application. NGINX acts as the reverse proxy in the example, but the same configuration applies to load balancing. The demo application is simply an NGINX virtual server that returns status code `200` and a text message. It serves as the demo application in the chapters about using rule sets with the ModSecurity 3.0 as well.

NOTE: This section applies to both NGINX and NGINX Plus.

Creating the Demo Web Application

Create the demo web application by configuring a virtual server in NGINX.

1. Create the file **/etc/nginx/conf.d/echo.conf** with the following content. It configures a webserver that listens on localhost port 8085 and returns status code `200` and a message containing the requested URI:

```
server {
    listen localhost:8085;

    location / {
        default_type text/plain;
        return 200 "Thank you for requesting ${request_uri}\n";
    }
}
```

2. Test the application by reloading the NGINX configuration and making a request:

```
$ sudo nginx -s reload
$ curl -D - http://localhost:8085 HTTP/1.1 200 OK
Server: nginx/1.11.10
Date: Wed, 3 May 2017 08:55:29 GMT Content-Type: text/plain
Content-Length: 27
Connection: keep-alive

Thank you for requesting /
```

Configuring NGINX as a Reverse Proxy

Configure NGINX as a reverse proxy for the demo application.

1. Create the file **/etc/nginx/conf.d/proxy.conf** with the following content. It configures a virtual server that listens on port 80 and proxies all requests to the demo application:

```
server {
    listen 80;

    location / {
        proxy_pass http://localhost:8085;
        proxy_set_header Host $host;
    }
}
```

Note: If any other virtual servers (`server` blocks) in your NGINX configuration listen on port 80, you need to disable them for the reverse proxy to work correctly. For example, the **/etc/nginx/conf.d/default.conf** file provided in the **nginx** package includes such a server block. Comment out or remove the **`server`** block, but do not remove or rename the **default.conf** file itself – if the file is missing during an upgrade, it is automatically restored, which can break the reverse-proxy configuration.

2. Reload the NGINX configuration:

```
$ sudo nginx -s reload
```

3. Verify that a request succeeds, which confirms that the proxy is working correctly:

```
$ curl -D - http://localhost
HTTP/1.1 200 OK
Server: nginx/1.11.10
Date: Wed, 3 May 2017 08:58:02 GMT
Content-Type: text/plain
Content-Length: 27
Connection: keep-alive

Thank you for requesting /
```

Protecting the Demo Web Application

Configure ModSecurity 3.0 to protect the demo web application by blocking certain requests.

1. Create the folder **/etc/nginx/modsec** for storing ModSecurity configuration:

```
$ sudo mkdir /etc/nginx/modsec
```

2. Download the file of recommended ModSecurity configuration from the **v3/master** branch of the ModSecurity GitHub repo and name it **modsecurity.conf**:

```
$ cd /etc/nginx/modsec
$ sudo wget https://raw.githubusercontent.com/SpiderLabs/
ModSecurity/v3/master/modsecurity.conf-recommended
$ sudo mv modsecurity.conf-recommended modsecurity.conf
```

3. Enable execution of rules by commenting out the existing `SecRuleEngine` directive in **modsecurity.conf** and adding the indicated directive. We will define the sample rule in the next step:

```
# SecRuleEngine DetectionOnly
SecRuleEngine On
```

For more information about the SecRuleEngine directive, see: github.com/SpiderLabs/ModSecurity/wiki/Reference-Manual#SecRuleEngine.

4. Create the main ModSecurity 3.0 configuration file, **/etc/nginx/modsec/main.conf,** and define a rule in it:

```
# Include the recommended configuration
Include /etc/nginx/modsec/modsecurity.conf

# A test rule
SecRule ARGS:testparam "@contains test" "id:1234,deny,log,status:403"
```

- `Include` – Includes the recommended configuration from the **modsecurity.conf** file.
- `SecRule` – Creates a rule that protects the application by blocking requests and returning status code `403` when the `testparam` parameter in the query string contains the string `test`.

For more information about the `SecRule` directive, see: github.com/SpiderLabs/ModSecurity/wiki/Reference-Manual#SecRule".

5. Change the reverse proxy configuration file (**/etc/nginx/conf.d/proxy.conf**) to enable the ModSecurity 3.0:

```
server {
    listen 80;

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

    location / {
        proxy_pass http://localhost:8085;
        proxy_set_header Host $host;
    }
}
```

- `modsecurity on` – Enables the ModSecurity 3.0.
- `modsecurity_rules_file` – Specifies the location of the ModSecurity 3.0 configuration file.

Documentation for `modsecurity*` directives in the NGINX configuration file is available on GitHub at: github.com/SpiderLabs/ModSecurity-nginx#usage.

6. Reload the NGINX configuration:

```
$ sudo nginx -s reload
```

7. Verify that the rule configured in Step 4 works correctly, by making a request that includes the string test in the value of the query string `testparam` parameter:

```
$ curl -D - http://localhost/foo?testparam=thisisatestofmodsecurity
HTTP/1.1 403 Forbidden
Server: nginx/1.11.10
Date: Wed, 3 May 2017 09:00:48 GMT
Content-Type: text/html
Content-Length: 170
Connection: keep-alive

<html>
<head><title>403 Forbidden</title></head>
<body bgcolor="white">
<center><h1>403 Forbidden</h1></center>
<hr/><center>nginx/1.11.10</center>
</body>
</html>
```

The request returns status code `403`, confirming that the WAF is enabled and executing the rule.

Setting Up Logging

By default, the ModSecurity 3.0 logs its activity to the NGINX Plus error log, at the `warn` level for blocked requests and the `info` level for other messages. The `error_log` directive in the main **/etc/nginx/nginx.conf** file provided with NGINX Plus is configured to write messages to **/var/log/nginx/error.log** at the `warn` level and higher. If you want to capture all messages from the ModSecurity 3.0, you need to set the logging level to `info` instead:

```
error_log /var/log/nginx/error.log info;
```

Note: Your existing configuration might include definitions for multiple error logs, in files other than **nginx.conf** and placed in more specific contexts, such as a `location` block. Make sure to update all the relevant `error_log` directives.

Controlling Audit and Debug Logging

ModSecurity also supports audit logging as configured with the `SecAudit*` set of directives. Audit logging is enabled in the recommended configuration that we downloaded to **/etc/nginx/modsec/modsecurity.conf** (in Step 2 of Protecting the Demo Web Application), but we recommend disabling it in production environments, both because audit logging affects ModSecurity performance and because the log file can grow large very quickly and exhaust disk space. To disable audit logging, change the value of the `SecAuditEngine` directive in **modsecurity.conf** to `off`:

```
SecAuditEngine off
```

If you experience problems with the ModSecurity 3.0, you can enable debug logging by changing the `SecDebugLog` and `SecDebugLogLevel` directives in **modsecurity.conf** to the following values. Like audit logging, debug logging generates a large volume of output and affects ModSecurity performance, so we recommend disabling it in production.

```
SecDebugLog /tmp/modsec_debug.log
SecDebugLogLevel 9
```

Note: For more details on logging, see Chapter 5.

Conclusion

In this section, we installed ModSecurity 3.0 for NGINX and NGINX Plus, created a simple rule for quick testing, and set up logging of ModSecurity 3.0 activity.

The simple rule works correctly, In the next chapter we will cover how to install the OWASP Core Rule Set (CRS).

3 Installing the OWASP Core Rule Set

"Break the rules. Not the law, but break the rules."

–Arnold Schwarzenegger

After installing the ModSecurity 3.0 software, the next step is to install ModSecurity's rule set(s). The rules define the attack patterns and determine what ModSecurity will block.

The OWASP Core Rule Set (CRS) is the base rule set that all ModSecurity users should install. The CRS is community-maintained and contains rules to help stop common attack vectors, such as SQL injection, cross-site scripting, and many others. It also has rules to integrate with Project Honeypot (Chapter 4) as well as rules to detect bots and scanners.

On top of the CRS you can install the Trustwave SpiderLabs commercial rules, which provide additional protections. You can find a detailed description of the commercial rules in Appendix A.

This chapter will cover installation of the OWASP Core Rule Set (CRS).

Running the Nikto Scanning Tool

We begin by sending attack traffic to the demo web application created in Example: Configuring ModSecurity 3.0 with a Simple Rule. Many attackers run vulnerability scanners to identify security vulnerabilities in a target website or app. Once they learn what vulnerabilities are present, they can launch the appropriate attacks.

We're using the Nikto scanning tool to generate malicious requests, including probes for the presence of files known to be vulnerable, cross-site scripting (XSS), and other types of attack. The tool also reports which requests passed through to the application, revealing potential vulnerabilities in the application.

Run the following commands to get the Nikto code and run it against the web application. The `324 item(s)` in the output are potential vulnerabilities, revealed by requests that passed through to the application.

```
$ git clone https://github.com/sullo/nikto
Cloning into 'nikto'...
$ cd nikto
$ perl program/nikto.pl -h localhost
- Nikto v2.1.6
...
+ 7531 requests: 0 error(s) and 324 item(s) reported on remote host
```

Next we enable the CRS, and then test how it blocks most of Nikto's requests and so decreases the number of items reported.

Enabling the OWASP CRS

To enable the OWASP CRS, perform the following steps:

1. Download the latest OWASP CRS from GitHub and extract the rules into **/usr/local** or another location of your choice:

```
$ wget https://github.com/SpiderLabs/owasp-modsecurity-crs/archive/
v3.0.2.tar.gz
$ tar -xzvf v3.0.2.tar.gz
$ sudo mv owasp-modsecurity-crs-3.0.2 /usr/local
```

2. Create the **crs-setup.conf** file as a copy of **crs-setup.conf.example**:

```
$ cd /usr/local/owasp-modsecurity-crs-3.0.2
$ sudo cp crs-setup.conf.example crs-setup.conf
```

3. Add `Include` directives in the main ModSecurity configuration file (**/etc/nginx/modsec/main.conf**, created in Step 4 of Protecting the Demo Web Application) to read in the CRS configuration and rules. Comment out any other rules that might already exist in the file, such as the sample SecRule directive created in that section:

```
# Include the recommended configuration
Include /etc/nginx/modsec/modsecurity.conf

# OWASP CRS v3 rules
Include /usr/local/owasp-modsecurity-crs-3.0.2/crs-setup.conf
Include /usr/local/owasp-modsecurity-crs-3.0.2/rules/*.conf
```

4. Reload the NGINX Plus configuration:

```
$ sudo nginx -s reload
```

Testing the CRS

In this section, we explore how rules in the CRS block Nikto's requests based on particular characteristics of the requests. Our ultimate goal is to show that the CRS blocks all of Nikto's requests, so that none of the vulnerabilities Nikto detects are left open for attackers to exploit.

Disabling Blocking of Requests Based on the User-Agent Header

The CRS recognizes requests from scanners, including Nikto, by inspecting the `User-Agent` header. As shown in the following output, the CRS comes preconfigured to block requests that have the default `User-Agent` header for Nikto (`Nikto`).

```
$ curl -H "User-Agent: Nikto" http://localhost/
<html>
<head><title>403 Forbidden</title></head>
<body bgcolor="white">
<center><h1>403 Forbidden</h1></center>
<hr><center>nginx/1.11.10</center>
</body>
</html>
```

(If you want to see exactly how the CRS ranks and blocks requests based on the `User-Agent` header and related characteristics of requests from scanners, you can correlate the rule ID numbers found in the NGINX error log with the rules in the CRS's Scanner Detection rule set (**REQUEST-913-SCANNER-DETECTION.conf**).)

During this testing phase we don't want to block all requests from Nikto, because we're using them to detect possible vulnerabilities in our demo app. To stop the CRS from blocking requests just because their `User-Agent` header is `Nikto`, we reconfigure Nikto not to include `Nikto` and related values in the header. Comment out the current setting for `USERAGENT` in **program/nikto.conf** and replace it with the value shown:

```
# USERAGENT=Mozilla/5.00 (Nikto/@VERSION) (Evasions:@EVASIONS)
(Test:@TESTID)
USERAGENT=Mozilla/5.00
```

Eliminating Requests for Vulnerable Files

When we rerun Nikto against the web application, we see that only 116 of Nikto's requests get through to the application server, compared to 324 when the CRS wasn't enabled. This indicates that the CRS is protecting our application from a large proportion of the vulnerabilities exposed by Nikto's requests.

```
$ perl program/nikto.pl -h localhost
...
+ 7531 requests: 0 error(s) and 116 item(s) reported on remote host
```

The output from Nikto is very long, and so far we have been truncating it to show just the final line, where the number of items is reported. When we look at the output more closely, we see that many of the remaining 116 items refer to a vulnerable file in the application, as in this example:

```
$ perl program/nikto.pl -h localhost
...
+ /site.tar: Potentially interesting archive/cert file found.
...
+ 7531 requests: 0 error(s) and 116 item(s) reported on remote host
```

Recall that in Installing ModSecurity, we configured our demo application to return status code **200** for every request, without actually ever delivering a file. Nikto is interpreting these **200** status codes to mean that the file it is requesting actually exists, which in the context of our application is a false positive.

Now we eliminate such requests so we can better see where actual vulnerabilities might exist. Disable the requests by adding **-sitefiles** in **program/nikto.conf** as shown:

```
# Default plug-in macros
# Remove plug-ins designed to be run standalone
@@EXTRAS=dictionary;siebel;embedded
@@DEFAULT=@@ALL;-@@EXTRAS;tests(report:500);-sitefiles
```

Blocking Requests with XSS Attempts

When we rerun Nikto again, it reports only 26 items:

```
$ perl program/nikto.pl -h localhost
- Nikto v2.1.6
...
+ 7435 requests: 0 error(s) and 26 item(s) reported on remote host
```

Most of the 26 items arise because the OWASP CRS is not currently configured to block requests that contain XSS attempts in the request URL, such as:

```
<script>alert('Vulnerable')</script>
```

To block requests with XSS attempts, edit rules 941160 and 941320 in the CRS's XSS Application Attack rule set (**REQUEST-941-APPLICATION-ATTACK-XSS.conf**) by adding `REQUEST_URI` at the start of the variables list for each rule:

```
SecRule REQUEST_URI|REQUEST_COOKIES|!REQUEST_COOKIES:/__utm/ ...
```

Reload the NGINX Plus configuration to read in the revised rule set:

```
$ sudo nginx -s reload
```

When we rerun Nikto, it reports only four items, and they are false positives for our application:

```
$ perl program/nikto.pl -h localhost
+ The anti-clickjacking X-Frame-Options header is not present.
+ The X-XSS-Protection header is not defined. This header can
hint to the user agent to protect against some forms of XSS
+ The X-Content-Type-Options header is not set. This could allow
the user agent to render the content of the site in a different
fashion to the MIME type
+ No CGI Directories found (use '-C all' to force check all
possible dirs)
+ /smg_Smxcfg30.exe?vcc=3560121183d3: This may be a Trend Micro
Officescan 'backdoor'.
+ 7435 requests: 0 error(s) and 4 item(s) reported on remote host
```

Limitations

Inspecting the response body is not supported, so rules that do so have no effect.

Conclusion

We used the OWASP ModSecurity Core Rule Set (CRS) to protect our web application against a wide range of generic attacks and saw how the CRS blocks malicious requests generated by the Nikto scanning tool.

For information about another supported ModSecurity rule set, see Appendix A: Installing the TrustWave SpiderLabs Commercial Rule Set.

4 Enabling Project Honeypot

"It takes 20 years to build a reputation and five minutes to ruin it. If you think about that, you'll do things differently."

–Warren Buffett

To help fight crime, the FBI maintains a public Ten Most Wanted list of the most dangerous criminals out there. Anyone who sees someone on the list will know to call the police, making it more difficult for these criminals to commit more crimes.

In the world of technology, there's a similar concept called Project Honeypot. Project Honeypot maintains a list of known malicious IP addresses, available free to the public. ModSecurity integrates with Project Honeypot and can automatically block IP addresses on the Project Honeypot list. This process is known as *IP reputation*.

In this chapter, we cover how to configure ModSecurity 3.0 to integrate with Project Honeypot, for both NGINX and NGINX Plus.

How Project Honeypot Works

Project Honeypot is a community-driven online database of IP addresses that are suspected spammers or bots. Each IP address is assigned a threat score between 0 and 255; the higher the number, the more likely the IP address is to be malicious.

The Project Honeypot database is powered by a network of volunteers who set up "honeypots". A honeypot, in this context, is a fake page on a site that shows up when a bot scans a site, but is invisible to regular people accessing

the site with a web browser. When the scanner follows the honeypot link and attempts to interact with the page – harvesting, for example, an embedded honeypot email address – the IP address is added to the database.

Project Honeypot lookups are done in real time when an HTTP request is received, so there will likely be a performance impact for enabling this functionality. The results are cached, however, to minimize the performance impact. Before enabling Project Honeypot integration in a production environment, please be sure to test the potential performance impact it will have on your applications.

For more details on how Project Honeypot works, please see: projecthoneypot.org/services_overview.php

1. Set Up Your Honeypot

To start using Project Honeypot, set up a honeypot on your site using the script provided by Project Honeypot:

1. Sign up a for a free Project Honeypot account. (projecthoneypot.org/create_account.php)
2. Set up your honeypot – Project Honeypot offers the honeypot script in PHP, Python, ASP, and a few other languages. (projecthoneypot.org/manage_honey_pots.php)
3. Download the honeypot script.

In this scenario, we use PHP for the scripting language. If your preferred language is not supported by Project Honeypot, PHP is a good choice, because it's very easy to configure NGINX and NGINX Plus to run PHP scripts using PHP-FPM.

There are plenty of tutorials online on how to install PHP-FPM. For Ubuntu 16.04 and later, you can use these commands:

```
$ apt-get update
$ apt-get -y install php7.0-fpm
```

You can then configure the Project Honeypot PHP script by adding this server block:

```
server {
    server_name www.example.com;

    location ~ \.php$ {
        modsecurity off;
        root /code;
        try_files $uri =404;
        fastcgi_split_path_info ^(.+\.php)(/.+)$;
        fastcgi_pass localhost:9000;
        include fastcgi_params;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_
        script_name;
        fastcgi_param PATH_INFO $fastcgi_path_info;
    }
}
```

Notes:

- In the `server_name` directive, for **www.example.com** substitute the domain name you registered with Project Honeypot.
- ModSecurity must be disabled on the honeypot script for it to function properly.
- In the `root` directive, for **/code**, substitute the directory where you placed the honeypot script.

Once the script is installed, access it in a web browser and click the activation link to activate the honeypot.

2. Add the Honeypot Link to All Pages

The next step is to configure NGINX or NGINX Plus to add the honeypot link to all pages.

To catch bots and scanners, insert a link to the honeypot script on every page. The link is invisible to regular people using a web browser but visible to bots and scanners. Here, we use the `sub_filter` directive to add the link to the bottom of each page:

```
location / {
    proxy_set_header Host $host;
    proxy_pass http://my_upstream;

    sub_filter '</html>'
               '<a href="http://www.example.com/weddingobject.php">
               <!-- hightest --></a></html>';
}
```

In this example, the name of our PHP honeypot file is **weddingobject.php**. The `sub_filter` directive looks for the HTML end-of-page tag, `</html>`, and inserts the invisible link there.

3. Enable IP Reputation in the Core Rule Set

Now that our honeypot is set up, we can configure ModSecurity to query Project Honeypot on all HTTP requests.

1. Request a Project Honeypot http:BL access key. (projecthoneypot.org/httpbl_configure.php)

2. In the file **/usr/local/owasp-modsecurity-crs-3.0.0/crs-setup.conf**, which you installed according to the Installing the OWASP Core Rule Set, locate the `SecHttpBlKey` block:

```
SecHttpBlKey my_api_key
SecAction "id:900500,\
    phase:1,\
    nolog,\
    pass,\
    t:none,\
    :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** is disabled in the above example, as it's unlikely that you want to block search engine crawlers.

3. Reload the configuration for the changes to take effect:

```
$ nginx -t && nginx -s reload
```

At this point, Project Honeypot is fully enabled and ModSecurity queries Project Honeypot on all HTTP requests. To minimize the performance impact, only the first request from a given IP address is sent to Project Honeypot, and the results of the query are cached.

4. Verify It Works

The Project Honeypot queries are based off the client source IP address. It's not easy to spoof a source IP address, so a good way to test that the functionality is working is by adding this custom rule to **/etc/nginx/modsec/main.conf**.

It sends the value of the IP address argument, passed in as part of the request, to Project Honeypot:

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

Reload the configuration for the rule to take effect:

```
$ nginx -t && nginx -s reload
```

Then run the following `curl` command to test the rule with an IP address from Project Honeypot's list of known bad IP addresses (projecthoneypot.org/list_of_ips.php) (substitute that address for the sample address used here, 203.0.113.20, which is a standard address reserved for documentation). If the rule works correctly, the request is blocked with status code `403`:

```
$ curl -i -s -k -X $'GET' 'http://localhost/?IP=203.0.113.20'
HTTP/1.1 403 Forbidden
Server: nginx/1.13.4
Date: Wed, 04 Oct 2017 21:29:17 GMT
Content-Type: text/html
Transfer-Encoding: chunked
Connection: keep-alive
```

Conclusion

In this chapter, we covered the steps for configuring ModSecurity 3.0 to work with Project Honeypot. Project Honeypot is a very useful tool for automatically blocking known bad IP addresses. It's free and is powered by a community of users setting up honeypots on their own sites.

5 Debugging, Logging, and Troubleshooting

"ModSecurity will help you sleep better at night because, above all, it solves the visibility problem: it lets you see your web traffic."

–Ivan Ristić, creator of ModSecurity

When something is not working as you expect it to, logs are always the first place to look. Good logs can provide valuable insights to help you troubleshoot the problems you're facing. One of the reasons Ivan Ristić originally created ModSecurity is that he was frustrated with the lack of visibility in the tools he was using. It's no surprise, then, that ModSecurity has extensive logging and debugging capabilities.

ModSecurity has two types of logs:

- An audit log. For every transaction that's blocked, ModSecurity provides detailed logs about the transaction and why it was blocked.
- A debug log. When turned on, this log keeps extensive information about everything that ModSecurity does.

The audit log is useful for learning not just why an individual attack was blocked, but for finding out more about overall attack patterns. You might be surprised by how much bot and scanner traffic you get just by exposing ports 80 and/or 443 to the Internet.

In this chapter, we'll describe the basics of logging and debugging with ModSecurity.

Audit Log

The main log in ModSecurity is the audit log, which logs all attacks, including potential attacks, that occur. If you've followed our installation instructions in Installing ModSecurity, then by default, ModSecurity will log all transactions that triggered a warning or error, as well as all transactions that resulted in `5xx` and `4xx` responses, except for `404`. (For an Ubuntu 16.04 system only, the audit log is in **/var/log/modsec_audit.log**.)

The ModSecurity audit log is partitioned into sections. This makes it easier to scan the log and find the information you're looking for. The table below outlines what each section contains:

Section	Description
A	Audit log header (mandatory)
B	Request headers
C	Request body
D	Reserved
E	Response body
F	Response headers
G	Reserved
H	Audit log trailer, which contains additional data
I	Compact request body alternative (to part C), which excludes files
J	Information on uploaded files
K	Contains a list of all rules that matched for the transaction
Z	Final boundary (mandatory)

Each transaction that triggers an audit log entry will have any or all of the above sections logged. You can configure which sections are logged.

Audit Log Example

A sample ModSecurity audit log entry might look like this:

```
---ICmPEb5c---A--
[04/Oct/2017:21:45:15 +0000] 150715351558.929952 141.212.122.16 64384
141.212.122.16 80
---ICmPEb5c---B--
GET / HTTP/1.1
Host: 54.183.57.254
User-Agent: Mozilla/5.0 zgrab/0.x
Accept-Encoding: gzip

---ICmPEb5c---D--

---ICmPEb5c---F--
HTTP/1.1 200
Server: nginx/1.13.4
Date: Wed, 04 Oct 2017 21:45:15 GMT
Content-Type: text/html
Connection: keep-alive

---ICmPEb5c---H--
ModSecurity: Warning. Matched "Operator `Rx' with parameter
`^[\d.:]+$' against variable `REQUEST_HEADERS:Host' (Value:
`54.183.57.254' ) [file "/root/owasp
-v3/rules/REQUEST-920-PROTOCOL-ENFORCEMENT.conf"] [line "733"] [id
"920350"] [rev "2"] [msg "Host header is a numeric IP address"]
[data "54.183.57.254"] [s
everity "4"] [ver "OWASP_CRS/3.0.0"] [maturity "9"] [accuracy "9"]
[tag "application-multi"] [tag "language-multi"] [tag "platform-
multi"] [tag "attack-prot
ocol"] [tag "OWASP_CRS/PROTOCOL_VIOLATION/IP_HOST"] [tag "WASCTC/
WASC-21"] [tag "OWASP_TOP_10/A7"] [tag "PCI/6.5.10"] [ref
"o0,13v21,13"]

---ICmPEb5c---I--

---ICmPEb5c---J--

---ICmPEb5c---Z--
```

Though it's not immediately apparent from the table above, the best section to find information on why a particular request was blocked is section H, not section K. From the above audit log example, if we scan through section H, we can see the message **`"Host header is a numeric IP address"`**, which indicates someone tried to access our site by IP address rather than by hostname. This may be indicative of a scanner.

Audit Logging Configuration

If you followed our instructions for installing and configuring ModSecurity, you'll find the audit logging configuration in **/etc/nginx/modsec/modsecurity.conf**. In that file, you'll see the following three directives that control what is put into the audit log:

```
SecAuditEngine RelevantOnly
SecAuditLogRelevantStatus "^(?:5|4(?!04))"
SecAuditLogParts ABIJDEFHZ
```

where

- SecAuditEngine – Controls what should be logged. Options are:
  - `Off` – Disable the audit log.
  - `On` – Log all transactions, which can can useful when debugging.
  - `RelevantOnly` – Log only transactions that have triggered a warning/error, or have a status code that matches what's in the `SecAuditLogRelevantStatus` directive.
- `SecAuditLogRelevantStatus` – `If SecAuditEngine` is set to `RelevantOnly`, then this directive controls what HTTP response status codes should be logged. It's regular expression-based. The above value will log all `5xx` and `4xx` responses, excluding `404`s.
- `SecAuditLogParts` – Controls what sections should be included in the access log. Removing sections you're not interested in reduces the size of the audit log and make it easier to scan.

For additional audit-logging configuration directives, refer to the ModSecurity wiki at:
github.com/SpiderLabs/ModSecurity/wiki/Reference-Manual.

Debug Log

When the debug log is turned on, it provides a wealth of information on everything ModSecurity does. For troubleshooting issues as to why something is not working the way you expect it to, the debug log is your go-to resource. It's also great if you're getting started with ModSecurity and want to observe why it does things a certain way.

Debug Log Example

The debug log looks like the following. It has a lot of details on the actions ModSecurity takes for any and all transactions:

```
[4] (Rule: 1234) Executing operator "Contains" with param "test"
    against ARGS:testparam.
[9] Target value: "test" (Variable: ARGS:testparam)
[9] Matched vars updated.
[4] Running [independent] (non-disruptive) action: log
[9] Saving transaction to logs
[4] Rule returned 1.
[9] (SecDefaultAction) Running action: log
[9] Saving transaction to logs
[9] (SecDefaultAction) Running action: auditlog
[4] (SecDefaultAction) ignoring action: pass (rule contains a
    disruptive action)
[4] Running (non-disruptive) action: auditlog
[4] Running (disruptive)     action: deny
```

The debug log lists the rule ID number for easy searching. In this example, the output is from our test rule with ID number 1234.

Debug Log Configuration

By default, the debug log is disabled, as it can negatively affect performance. Just as with audit logging, the debug log is configured in **/etc/nginx/modsec/modsecurity.conf**. In that file, there are two configuration directives that are commented out. To enable debug logging, uncomment them and change them to the following:

```
SecDebugLog /var/log/modsec_debug.log
SecDebugLogLevel 9
```

where

- SecDebugLog – Specifies the path to the debug log file.
- SecDebugLogLevel – `0–9` indicates how much information to log, with `9` being the most. If you're troubleshooting, setting this value to `9` is the most helpful.

Conclusion

In this chapter, we covered how to get started using the extensive logging capabilities within ModSecurity. ModSecurity has both audit logs, which contain information about all blocked transactions, and a debug log to further assist you if you're having trouble using ModSecurity.

Appendix A:

Installing the TrustWave SpiderLabs Commercial Rule Set

The TrustWave SpiderLabs Commercial Rule Set provides additional protections, such as application specific rule sets for WordPress, Joomla, SharePoint, and others. Learn more about the TrustWave SpiderLabs Commercial Rule Set at: trustwave.com/Products/Application-Security/ModSecurity-Rules-and-Support.

This chapter covers installation of the TrustWave SpiderLabs Commercial Rule Set.

Overview

NGINX Plus Release 12 and NGINX 1.11.5 and later support ModSecurity 3.0. The ModSecurity WAF protects web applications against various Layer 7 attacks; provides DDoS mitigation, real-time blacklisting, audit logging; and enables PCI-DSS 6.6 compliance.

The ModSecurity® Rules from Trustwave SpiderLabs® complement the Open Web Application Security Project Core Rule Set (OWASP CRS) with protection against specific attacks of multiple categories – including SQL injection, cross-site scripting (XSS), local and remote file includes (LFI and RFI) – for many common applications (ASP.NET, Joomla, WordPress, and many others). Additionally, the Trustwave SpiderLabs Rules provide IP reputation along with other capabilities, and are updated daily.

This chapter builds on the basic configuration in Installing ModSecurity, showing how to configure the Trustwave SpiderLabs Rules to protect the demo web application created there.

ModSecurity 3.0 also supports the OWASP CRS as described in Installing the OWASP Core Rule Set.

Prerequisites

You must purchase the Trustwave SpiderLabs Rules directly from Trustwave.

As noted above, this chapter builds on Installing ModSecurity and assumes you have following the instructions there to configure the demo application and NGINX as a reverse proxy.

Configuring the Trustwave SpiderLabs Rules

Purchasing the Trustwave SpiderLabs Rules gives you access to the ModSecurity Dashboard, which is a web portal where you can customize the Trustwave SpiderLabs Rules on individual instances of NGINX with the ModSecurity WAF (and other ModSecurity installations). The Dashboard simplifies configuration compared to the OWASP CRS, in two ways:

- You don't need to download rules onto individual NGINX instances, because the ModSecurity 3.0 dynamic module downloads them automatically when the `SecRemoteRules` directive is included in the ModSecurity configuration (see Step 3 in the next section).
- You enable and disable rules – a significant part of the configuration process – with a GUI on the Dashboard instead of in ModSecurity configuration files.

To configure the Trustwave SpiderLabs Rules for the demo application, first create a profile (or use the default one) that includes selected rules for protecting the application. Then modify the local ModSecurity configuration to make the ModSecurity dynamic module download and apply the rules. The instructions use the Configuration Wizard on the Dashboard for creating a profile.

Detailed instructions for using the Dashboard are not provided here. For more information, log in to the ModSecurity Dashboard at: dashboard.modsecurity.org.

Using the Configuration Wizard

To configure the Trustwave SpiderLabs Rules for the demo application, perform the following steps:

1. Log in to the ModSecurity Dashboard and start the Configuration Wizard. (dashboard.modsecurity.org)
2. Create a profile, enabling rules that are relevant for your application. None of the existing rules actually apply to our demo application, but for the purposes of this step select the WordPress-related rules. You can also enable additional options, such as IP reputation.
3. At the **Configure your server** step, the Wizard presents the `SecRemoteRules` directive that must be added to the ModSecurity configuration, similar to the line below:

```
SecRemoteRules license-key https://url
```

Here, the `SecRemoteRules` directive configures ModSecurity to download rules from the remote server, represented by the *url*, using the provided *license-key*.

The Wizard does not provide an interface for adding the directive, so you need to edit **/etc/nginx/modsec/main.conf** manually and add the `SecRemoteRules` directive presented by the Wizard (we created the file in Step 4 of Protecting the Demo Web Application). Comment out any other rules that might already exist in the file, such as the `SecRule` directive from that section:

```
# Include the recommended configuration
Include "/etc/nginx/modsec/modsecurity.conf"

SecRemoteRules license-key https://url
```

4. By default, the Trustwave SpiderLabs Rules only detect malicious requests and don't block them. To block the requests, add the following lines to **/etc/nginx/modsec/main.conf**, below the SecRemoteRules directive you added in the previous step:

```
SecDefaultAction "phase:2,log,auditlog,deny,status:403"
SecDefaultAction "phase:1,log,auditlog,deny,status:403"
```

The `SecDefaultAction` directive defines the default list of actions for the rules, with the **deny** action blocking malicious requests and returning status code **403**.

5. Reload the NGINX configuration:

```
$ sudo nginx -s reload
```

Reloading takes time as the rules are being downloaded from the remote server.

6. Once the Wizard reports that NGINX downloaded the rules, you can close the wizard and start testing the rules.

Testing the Rules

In Installing the OWASP Core Rule Set, we use the Nikto scanning tool to test how the CRS blocks malicious requests. However, Trustwave SpiderLabs rule are specific rules that cannot detect the generic attacks sent by Nikto.

The Dashboard provides a description of each ModSecurity rule, which you can use to construct and send a malicious request to NGINX to test how the rule behaves.

Caveats for the `SecRemoteRules` Directive

Currently, the only way to download the Trustwave SpiderLabs Rules is with the `SecRemoteRules` directive. While the directive simplifies the process of getting the rules onto an instance of ModSecurity 3.0, the following caveats apply:

- Every time you reload the NGINX configuration or restart NGINX, the rules are freshly downloaded from a remote server. The `SecRemoteRulesFailAction` directive controls what happens when the download fails – for example, when connectivity to the remote server is lost. The directive supports two values: `Abort`, which forces the reload or restart of NGINX to fail, and `Warn`, which lets NGINX reload or restart successfully but with no remote rules enabled. The `SecRemoteRulesFailAction` directive must appear above `SecRemoteRules` directives in a ModSecurity configuration file.
- Downloading the rules takes some time, which delays the reload or restart operation.
- Each `SecRemoteRules` definition leads to a separate download, further increasing the reload/restart time. To avoid that, try to minimize the number of `SecRemoteRules` definitions. Note that even if you define `SecRemoteRules` only in one file (as in the **/etc/nginx/modsec/main.conf** file modified in Step 3 above), each time you include this file into NGINX configuration using the `modsecurity_rules_file` directive (as in the **/etc/nginx/conf.d/proxy.conf** file created in Configuring NGINX as a Reverse Proxy), ModSecurity treats it as a separate definition.
- Merging rules from different contexts (`http`, `server`, `location`) also adds time to the reload/restart operation and consumes a lot of CPU, especially for a huge rule set such as the Trustwave SpiderLabs Rules. In addition to minimizing the number of `SecRemoteRules` definitions, try to include all rule definitions in a single context.

The Trustwave SpiderLabs rule set contains more than 16,000 rules for protecting various applications. The more rules there are, the worse the WAF performs, so it's crucial that you enable only rules that are relevant for your application.

Limitations

Inspecting the response body is not supported, so rules that do so have no effect.

Conclusion

In this chapter, we configured ModSecurity Rules from Trustwave SpiderLabs to protect our application against WordPress-related attacks. We also reviewed caveats for the SecRemoteRules directive.

Appendix B:

Document Revison History

Version	Date	Description
1.0	2017-12-14	Initial release
1.1	2018-01-12	• Page 4: Add mention of NGINX `sub_filter` directive for response filtering • Page 6: Fix typo, "`ibpcre++-dev`" -> "`libpcre++-dev`" • Page 11: Fix indentation issue in code block
1.2	2018-02-15	Page 3, fix typo `libmodesecurity` -> `libmodsecurity`