Red Hat JBoss Core Services 2.4.57

Red Hat JBoss Core Services ModSecurity Guide

For use with Red Hat JBoss middleware products.

Last Updated: 2024-02-06

Red Hat JBoss Core Services 2.4.57 Red Hat JBoss Core Services ModSecurity Guide

For use with Red Hat JBoss middleware products.

Legal Notice

Abstract

Configure and use the Red Hat JBoss Core Services ModSecurity module as a web application firewall.

Table of Contents

PROVIDING FEEDBACK ON RED HAT JBOSS CORE SERVICES DOCUMENTATION

To report an error or to improve our documentation, log in to your Red Hat Jira account and submit an issue. If you do not have a Red Hat Jira account, then you will be prompted to create an account.

Procedure

1. Click the following link to **create a ticket**.
2. Enter a brief description of the issue in the **Summary**.
3. Provide a detailed description of the issue or enhancement in the **Description**. Include a URL to where the issue occurs in the documentation.
4. Clicking **Submit** creates and routes the issue to the appropriate documentation team.

MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see our CTO Chris Wright's message .

CHAPTER 1. MODSECURITY MODULE

The ModSecurity module is a web application firewall (WAF) that you can use to filter, monitor, and block HTTP traffic that web clients send to a web server application. Unlike a regular firewall, a WAF uses filters to determine which applications and users can interact with your Apache HTTP Server application. The effectiveness of ModSecurity relies on user-defined rules that enable ModSecurity to perform configurable and real-time monitoring of HTTP traffic to detect attacks instantly.

NOTE

The Red Hat JBoss Core Services ModSecurity Guide provides information and examples for the ModSecurity version 2.9 module that is available with the Red Hat JBoss Core Services 2.4.57 release. The effectiveness of ModSecurity depends on user-generated rules. This document describes how to create and implement rules. This document does not provide a set of rules to use.

CHAPTER 2. CONFIGURING MODSECURITY ON RHEL

When you install Red Hat JBoss Core Services on Red Hat Enterprise Linux (RHEL), you can configure the ModSecurity module to function as a web application firewall (WAF) for the Apache HTTP Server.

NOTE

JBCS 2.4.57 does not currently provide an archive file distribution of the Apache HTTP Server for RHEL 9.

2.1. MODSECURITY DEPENDENCIES ON RHEL

ModSecurity has several dependencies to function successfully. Some of these dependencies are already included as a part of Red Hat JBoss Core Services.

The following table provides a list of ModSecurity dependencies:

Dependency	Part of JBCS on RHEL?
Apache Portable Runtimes (APR)	Yes
APR-Util	Yes
mod_unique_id	Yes
libcurl	Yes
Perl-Compatible Regular Expressions (PCRE)	Yes
libxml2	No

NOTE

On RHEL, Red Hat JBoss Core Services includes all of these dependencies except the **libxml2** library.

2.2. MODSECURITY INSTALLATION ON RHEL

The ModSecurity module is included as part of a Red Hat JBoss Core Services installation.

You can follow the procedures in the Red Hat JBoss Core Services Apache HTTP Server Installation Guide to download and install the Apache HTTP Server for your operating system.

Additional resources

- Red Hat JBoss Core Services Apache HTTP Server Installation Guide

2.3. LOADING MODSECURITY

You can load the ModSecurity module by using the **LoadModule** command.

Procedure

- To load the ModSecurity module, enter the following command:

```
LoadModule security2_module modules/mod_security2.so
```

2.4. CONFIGURING THE RULES DIRECTORY ON RHEL

ModSecurity functionality requires that you create rules that the system uses. Apache HTTP Server provides a preconfigured **mod_security.conf.sample** file in the ***HTTPD_HOME*/modsecurity.d** directory. To use ModSecurity rules, you must modify the **mod_security.conf.sample** file with settings that are appropriate for your environment. You can store the ModSecurity rules in the **modsecurity.d** directory or the **modsecurity.d/activated_rules** subdirectory.

Procedure

1. Go to the ***HTTPD_HOME*/modsecurity.d** directory.
2. Rename the **mod_security.conf.sample** file to **mod_security.conf**:

```
mv mod_security.conf.sample ./mod_security.conf
```

3. Open the **mod_security.conf** file and specify parameters for all the configuration directives that you want to use with the ModSecurity rules.

2.5. KEY MODSECURITY CONFIGURATION OPTIONS

You can use key ModSecurity configuration options to improve the performance of regular expressions, investigate ModSecurity 2.6 phase one moving to phase two hook, and allow use of certain directives in **.htaccess** files.

enable-pcre-jit

Enables Just-In-Time (JIT) compiler support in the Perl-Compatible Regular Expressions (PCRE) library 8.20 or later to improve the performance of regular expressions.

enable-request-early

Enables testing of the ModSecurity 2.6 move from phase one to phase two hook

enable-htaccess-config

Enables use of directives in **.htaccess** files when **AllowOverride Options** is set

CHAPTER 3. CONFIGURING MODSECURITY ON WINDOWS SERVER

When you install Red Hat JBoss Core Services on Windows Server, you can configure the ModSecurity module to function as a web application firewall (WAF) for the Apache HTTP Server.

3.1. MODSECURITY DEPENDENCIES ON WINDOWS SERVER

ModSecurity has several dependencies to function successfully. Some of these dependencies are already included as a part of Red Hat JBoss Core Services.

The following table provides a list of ModSecurity dependencies:

Dependency	Part of JBCS on Windows Server?
Apache Portable Runtimes	Yes
APR-Util	Yes
mod_unique_id	Yes
libcurl	Yes
Perl-Compatible Regular Expressions (PCRE)	Yes
libxml2	Yes

NOTE

On Windows Server, Red Hat JBoss Core Services includes all of these dependencies.

3.2. INSTALLING MODSECURITY ON WINDOWS SERVER

The ModSecurity module is included as part of a Red Hat JBoss Core Services installation. Apache HTTP Server provides many of the items that are required to run ModSecurity on Windows Server. However, you must ensure that your system complies with certain criteria to allow ModSecurity to function correctly.

Prerequisites

- The folder where you build software from source contains both the Apache source, which you use to build the Apache HTTP Server, and the ModSecurity source.
  For example:
  - Apache source is in **C:\ *sourceFolder*\httpd-2.4.57**
  - Apache has been installed to **C:\Apache2457**
  - ModSecurity source is in **C:\ *sourceFolder*\mod_security**

NOTE

In this case, ***sourceFolder*** is a generic folder that you use in conjunction with the project.

- Your build environment is set up correctly.
  For example:
  - Ensure that the ***PATH*** environment variable includes the Visual Studio variables set by **vsvars32.bat**.
  - Ensure that the ***PATH*** environment variable includes the **bin\** folder for **CMAKE**.
  - Set an environment variable for the Apache source code directory, which is located at **C:\\ *sourceDirectory*\\httpd-2.4.57**.

Procedure

- Follow the procedures in the Red Hat JBoss Core Services Apache HTTP Server Installation Guide to download and install the Apache HTTP Server to the appropriate location on your **C:** drive.

Additional resources

- Red Hat JBoss Core Services Apache HTTP Server Installation Guide

3.3. CONFIGURING THE RULES FOLDER ON WINDOWS SERVER

ModSecurity functionality requires that you create rules that the system uses. Apache HTTP Server provides a preconfigured **mod_security.conf.sample** file in the ***HTTPD_HOME*\\modsecurity.d** folder. To use ModSecurity rules, you must modify the **mod_security.conf.sample** file with settings that are appropriate for your environment. You can store the ModSecurity rules in the **modsecurity.d** folder or the **modsecurity.d\\activated_rules** subfolder.

Procedure

1. Go to the ***HTTPD_HOME*\\modsecurity.d** folder.
2. Rename the **mod_security.conf.sample** file to **mod_security.conf**.
3. Open the **mod_security.conf** file and specify parameters for all the configuration directives that you want to use with the ModSecurity rules.

3.4. KEY MODSECURITY CONFIGURATION OPTIONS

You can use key ModSecurity configuration options to improve the performance of regular expressions, investigate ModSecurity 2.6 phase one moving to phase two hook, and allow use of certain directives in **.htaccess** files.

enable-pcre-jit

Enables Just-In-Time (JIT) compiler support in the Perl-Compatible Regular Expressions (PCRE) library 8.20 or later to improve the performance of regular expressions.

enable-request-early

Enables testing of the ModSecurity 2.6 move from phase one to phase two hook

enable-htaccess-config

Enables use of directives in **.htaccess** files when **AllowOverride Options** is set

CHAPTER 4. CREATING MODSECURITY RULES

ModSecurity primarily functions based on custom user-defined rules. These rules determine the types of security checks that ModSecurity performs.

4.1. MODSECURITY RULES IN THE APACHE REQUEST CYCLE

You can apply rules to any of the five ModSecurity processing phases of the Apache request cycle:

Request headers

Apply a ModSecurity rule to this phase by specifying a **REQUEST_HEADERS** variable in the rule syntax.

Request body

Apply a ModSecurity rule to this phase by specifying a **REQUEST_BODY** variable in the rule syntax.

Response headers

Apply a ModSecurity rule to this phase by specifying a **RESPONSE_HEADERS** variable in the rule syntax.

Response body

Apply a ModSecurity rule to this phase by specifying a **RESPONSE_BODY** variable in the rule syntax.

Logging

Apply a ModSecurity rule to this phase by specifying a **LOGGING** variable in the rule syntax.

Additional resources

- ModSecurity Reference Manual: Processing Phases

4.2. STRUCTURE OF MODSECURITY RULES

A ModSecurity rule typically consists of four main parts:

- A configuration directive
- One or more variables
- One or more operators
- One or more actions

4.3. MODSECURITY CONFIGURATION DIRECTIVES

A ModSecurity rule starts with a configuration directive. The configuration directives for ModSecurity are similar to the Apache HTTP Server directives. You can use most ModSecurity directives within the various Apache scope directives. However, you may only use some ModSecurity directives once in the main configuration file.

You must store these rules and the core rule files outside of the **httpd.conf** file. You can use Apache **Include** directives to call these rules. This facilitates the upgrade and migration of the rules.

Additional resources

- ModSecurity Reference Manual: Configuration Directives

4.4. EXAMPLE OF A SIMPLE MODSECURITY RULE

You can define the following simple ModSecurity rule, for example, to check if the URI portion of a request equals a specific lowercase value:

SecRule REQUEST_URI "@streq /index.php" "id:1,phase:1,t:lowercase,deny"

The preceding ModSecurity rule consists of the following compoonents:

SecRule

A *configuration directive* that creates a rule to analyze the specified variables by using the specified operator

NOTE

Most ModSecurity rules use this configuration directive.

REQUEST_URI

A *variable* that holds the full request URL including the query string data

"@streq /index.php"

An *operator* where **@streq** checks for string values that are equal to **/index.php**

"id:1,phase:1,t:lowercase,deny"

Actions or *transformations* that the rule performs

NOTE

The rule performs the **lowercase** action first before the rule implements the preceding operator instruction.

Based on the preceding example, during phase 1 of the Apache request cycle, the rule obtains the URI portion of the HTTP request and transforms the value to lowercase. The rule then checks if the transformed value equals **/index.php**. If the value does equal **/index.php**, ModSecurity denies the request and does not process any further rules.

4.5. EXAMPLE OF A COMPLEX MODSECURITY RULE

You can define the following complex ModSecurity rule, for example, to check if a request has changed history:

SecRule REQUEST_URI|REQUEST_BODY|REQUEST_HEADERS_NAMES|REQUEST_HEADERS "history.pushstate|history.replacestate" "phase:4,deny,log,msg:'history-based attacks detected'"

The preceding ModSecurity rule consists of the following components:

SecRule

A *configuration directive* that creates a rule to analyze the specified variables by using the specified operator

NOTE

Most ModSecurity rules use this configuration directive.

`REQUEST_URI|REQUEST_BODY|REQUEST_HEADERS_NAMES|REQUEST_HEADERS `
: A pipe-separated list of *variables* that define different parts of the request that the rule checks

"history.pushstate|history.replacestate"
: A pipe-separated pair of *operators* that check for the JavaScript **history.pushstate()** and **history.replacestate()** methods

"phase:4,deny,log,msg:'history-based attacks detected'"
: *Actions* or *transformations* that the rule performs if the specified operator values are found

Based on the preceding example, during phase 4 of the Apache request cycle, the rule checks different parts of the request cycle for **history.pushstate()** and **history.replacestate()** methods. If the rule finds these methods in the request URL string, request body, request header names, or request headers, the rule performs the following actions:

- **deny**
  Stops the rule processing and intercepts the transaction
- **log**
  Logs a successful match of the rule to the Apache error log file and the ModSecurity audit log
- **msg**
  Outputs a message defined as **history-based attacks detected** with the log

4.6. ADDITIONAL RESOURCES (OR NEXT STEPS)

- ModSecurity Reference Manual: Actions
- ModSecurity Reference Manual: Configuration Directives
- ModSecurity Reference Manual: Operators
- ModSecurity Reference Manual: Transformation functions
- ModSecurity Reference Manual: Variables