Fundamentos de Segurança Informática
2025/2026

Apache with ModSecurity
Installation and Getting started

1. Introduction

- The following installation notes illustrate the process for installing: ModSecurity on Linux CentOS
- For your particular Linux installation additional steps may be required (e.g., the installation of additional missing packages), thus the described steps are merely indicative.

2. ModSecurity and ModSecurity CRS installation in CentOS

```
# Install required packages (Apache 2 is required)
yum install httpd

# Install mod_security and mod_security-crs (epel-release is required)
yum install httpd mod_security mod_security_crs
```

3. Checking the installation

To confirm a correct installation, proceed as suggested. The following command allows to confirm if apache is configured to use the mod_security module:

```
# Check loaded modules
httpd -M
```

```
Loaded Modules:
...
security2_module (shared)
...
```

Other modules may be installed, nonetheless for the purpose of this document the required one is security2_module, as in the previous example.

4. mod_security configuration

By default, mod_security configuration in apache is defined in /etc/httpd/conf.d/mod_security.conf. In this file, among other configuration directives the following load other configuration aspects, as well as rules:

```
...
IncludeOptional modsecurity.d/*.conf
IncludeOptional modsecurity.d/activated_rules/*.conf
```

```
...
```

```
# This defines other configuration aspects, for example for enabling the usage of
# rules for attack detection.
SecRuleEngine On

# By default, rules are loaded from files defined in /etc/httpd/modsecurity.d/:
...
IncludeOptional modsecurity.d/*.conf
IncludeOptional modsecurity.d/activated_rules/*.conf
...
```

5. Testing with the previous configurations

After applying the configurations in the previous section, we can now test for specific tests, as in the following examples:

```
# Test attack to make remote execution of bash command line
curl http://<IP address>/?exec=/bin/bash

# Using curl, now for an SQLi attack
curl -d 'id=1 AND 1=1' http://localhost/index.php
```

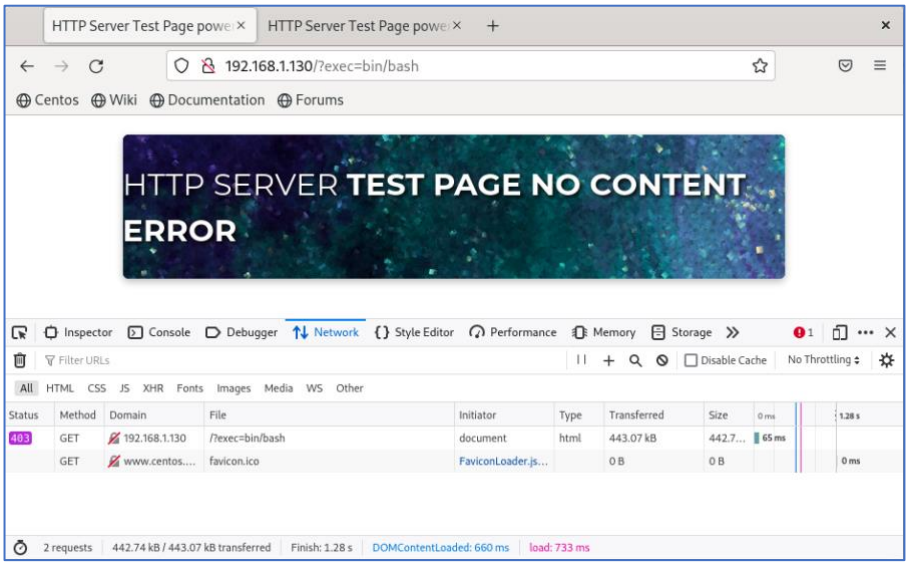
For example, for the previous attack the following information is logged in /var/log/httpd/modsec_audit.log:

Message: Access denied with code 403 (phase 2). Pattern match "(?i:([\\s'\"`\xc2\xb4\xe2\x80\x99\xe2\x80\x98\\(\\)]*?)\\b([\\d\\w]++)([\\s'\"`\xc2\xb4\xe2\x80\x99\xe2\x80\x98\\(\\)]*?)(?:(?:=|<=>|r?like|sounds\\s+like|regexp)([\\s'\"`\xc2\xb4\xe2\x80\x99\xe2\x80\x98\\(\\)]*?)\\2\\b|(?:!=|<=|>=|<>|<|>|\\^|is\\s+not ..." at ARGS:id. [file "/etc/httpd/modsecurity.d/activated_rules/modsecurity_crs_41_sql_injection_attacks.conf"] [line "77"] [id "950901"] [rev "2"] [msg "**SQL Injection Attack: SQL Tautology Detected**."] [data "**Matched Data: 1=1** found within ARGS:id: 1 AND 1=1"] [severity "CRITICAL"] [ver "OWASP_CRS/2.2.9"] [maturity "9"] [accuracy "8"] [tag "OWASP_CRS/WEB_ATTACK/SQL_INJECTION"] [tag "WASCTC/WASC-19"] [tag "OWASP_TOP_10/A1"] [tag "OWASP_AppSensor/CIE1"] [tag "PCI/6.5.2"]

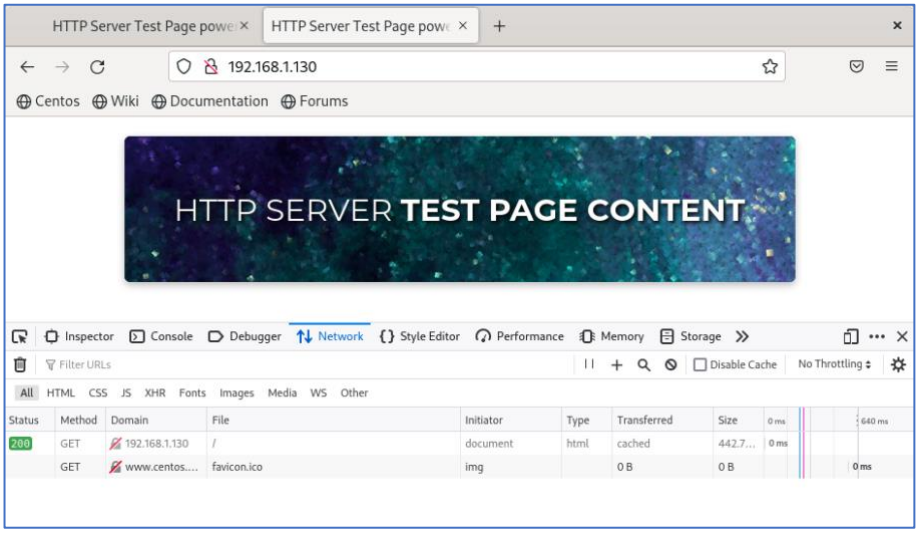
Note 1: Some attacks might be detected differently, according to the rules that are verified in first place.

Note 2: In CentOS an error page is like a regular page, thus it might be confusing to analyse the test results in the browser. Always rely on the logs/debug/audit information of ModSecurity.

The confirmation of results in the tests performed with the web browser need to be checked regarding the HTTP status code that is received. A 403 status corresponds to an access denied, the developer tools available in the browser can be used to examine the results, as illustrated in the following figures.



On the other hand, a normal request will generate a 200 OK code, as illustrated in the figure below.



6. Relevant information

The processing of Modsecurity is divided into five phases, as outlined in the following figure. The most used phase corresponds to phase 2, after the request body has been performed.

Phase number	Phase name	Phase occurs
1	REQUEST_HEADERS	Right after Apache has read the headers of the HTTP request.
2	REQUEST_BODY	After the request body has been read. Most ModSecurity rules are written to be processed in this phase.
3	RESPONSE_HEADERS	Right before the response headers are sent back to the client.
4	RESPONSE_BODY	Before the response body is sent back to client. Any processing of the response body to inspect for example data leaks should take place in this phase.
5	LOGGING	Right before logging takes place. At this point requests can no longer be blocked—all you can do is affect how logging is done.