

Web Server Hardening with ModSecurity & Fail2Ban on Kali Linux

In the world of cybersecurity, protecting web applications from real-time threats is non-negotiable. This report walks you through how I secured a web server using two powerful tools:

ModSecurity: A Web Application Firewall (WAF) that blocks threats like SQL injection, XSS, brute force, and more.

Fail2Ban: An intrusion prevention tool that bans IPs after suspicious activity is detected.

By combining these tools on Kali Linux with Nginx, we establish a strong defense mechanism against both application-layer and network-layer attacks.

Prerequisites

Before starting, ensure:

Kali Linux is updated:

sudo apt update && sudo apt upgrade -y

Nginx is installed:

sudo apt install nginx -y

Part 1: ModSecurity Installation

Step 1: Install Dependencies

ModSecurity requires several development libraries. Install them with:

sudo apt install -y git build-essential libcurl4-openssl-dev libgeoip-dev libltdb-dev libpcre3-dev libtool libxml2-dev libyajl-dev pkgconf wget zlib-dev

Step 2: Compile ModSecurity v3 from Source

```
cd /usr/src/  
sudo git clone --depth 1 -b v3/master --single-branch https://github.com/SpiderLabs/ModSecurity  
cd ModSecurity  
sudo git submodule init  
sudo git submodule update  
sudo ./build.sh  
sudo ./configure  
sudo make  
sudo make install
```

Step 3: Build the Nginx ModSecurity Connector

Since Nginx doesn't natively support ModSecurity, we need a dynamic module:

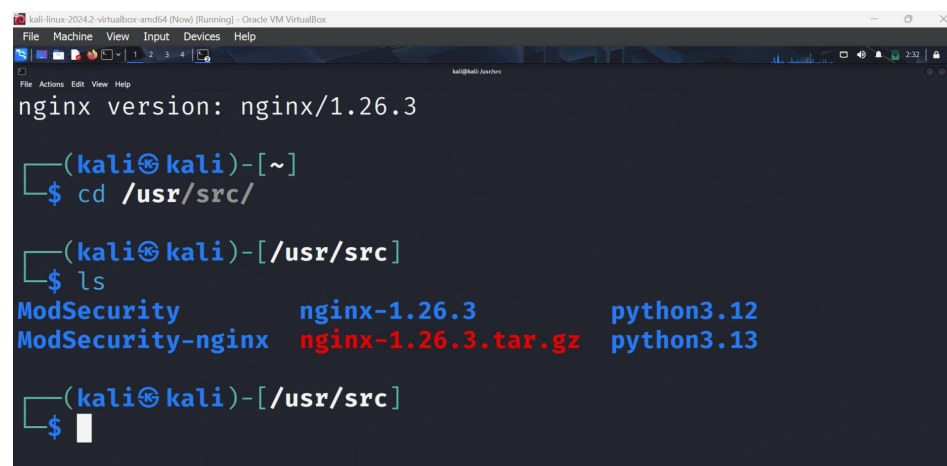
1. Clone the Nginx ModSecurity connector

```
cd /usr/src/  
sudo git clone --depth 1 https://github.com/SpiderLabs/ModSecurity-nginx.git
```

2. Download and compile Nginx with ModSecurity support

#check the version of nginx you installed

Nginx -v



```
kali-linux-2024.2-virtualbox-amd64 (Now) [Running] - Oracle VM VirtualBox  
File Machine View Input Devices Help  
nginx version: nginx/1.26.3  
  
(kali㉿kali)-[~]  
$ cd /usr/src/  
  
(kali㉿kali)-[/usr/src]  
$ ls  
ModSecurity          nginx-1.26.3          python3.12  
ModSecurity-nginx    nginx-1.26.3.tar.gz   python3.13
```

```
cd /usr/src
sudo wget http://nginx.org/download/nginx-1.26.3.tar.gz
sudo tar -xvzf nginx-1.26.3.tar.gz
```

#Now compile the ModSecurity module for Nginx:

```
cd nginx-1.26.3
sudo ./configure --with-compat --add-dynamic-module=../ModSecurity-nginx
sudo make modules
```

#Copy the compiled module to NGINX is configured to look for modules in a different folder:

```
sudo mkdir -p /usr/lib/nginx/modules
cd /usr/src/nginx-1.26.3/objs/
sudo cp ngx_http_modsecurity_module.so /usr/lib/nginx/modules/
```

#Load the module in Nginx configuration. Edit your nginx.conf to load the correct path:

```
sudo nano /etc/nginx/nginx.conf
```

#Add this line at the top of configuration

```
load_module /usr/lib/nginx/modules/ngx_http_modsecurity_module.so;
```

Then save and exit (Ctrl + O, Enter, then Ctrl + X).

Step 4: Copy recommended ModSecurity config and Unicode mapping

```
sudo mkdir -p /etc/nginx/modsec
cd /usr/src/Modsecurity/
sudo cp modsecurity.conf-recommended /etc/nginx/modsec/modsecurity.conf
sudo cp unicode.mapping /etc/nginx/modsec/unicode.mapping
sudo nano /etc/nginx/modsec/modsecurity.conf
```

#change this line SecRuleEngine from Relevantonly to On

SecRuleEngine On

Step 5: Enable OWASP Core Rule Set (CRS)

```
cd /etc/nginx/modsec
sudo git clone https://github.com/coreruleset/coreruleset.git
sudo mv coreruleset crs
sudo cp crs/crs-setup.conf.example crs/crs-setup.conf
```

Create main configuration file

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

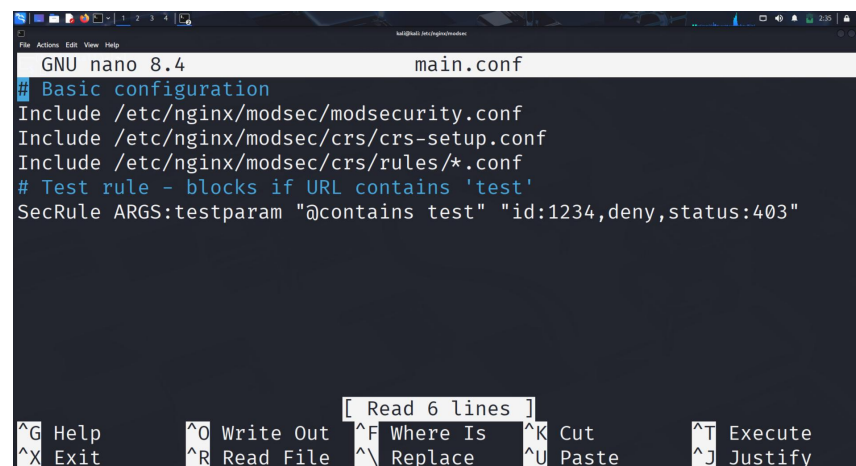
#Add this configurations

```
# Load base ModSecurity config
include /etc/nginx/modsec/modsecurity.conf
```

```
# Load OWASP CRS setup
include /etc/nginx/modsec/crs/crs-setup.conf
```

```
# Load all OWASP CRS rules
include /etc/nginx/modsec/crs/rules/*.conf
```

Save and exit (Ctrl + O, Enter, Ctrl + X)



```
GNU nano 8.4 main.conf
# Basic configuration
Include /etc/nginx/modsec/modsecurity.conf
Include /etc/nginx/modsec/crs/crs-setup.conf
Include /etc/nginx/modsec/crs/rules/*.conf
# Test rule - blocks if URL contains 'test'
SecRule ARGS:testparam "@contains test" "id:1234,deny,status:403"
```

Step 6: Enable ModSecurity in your NGINX site config

```
cd /etc/nginx/sites-available/
```

```
sudo nano default
```

Inside the server { ... } block, add:

```
modsecurity on;
```

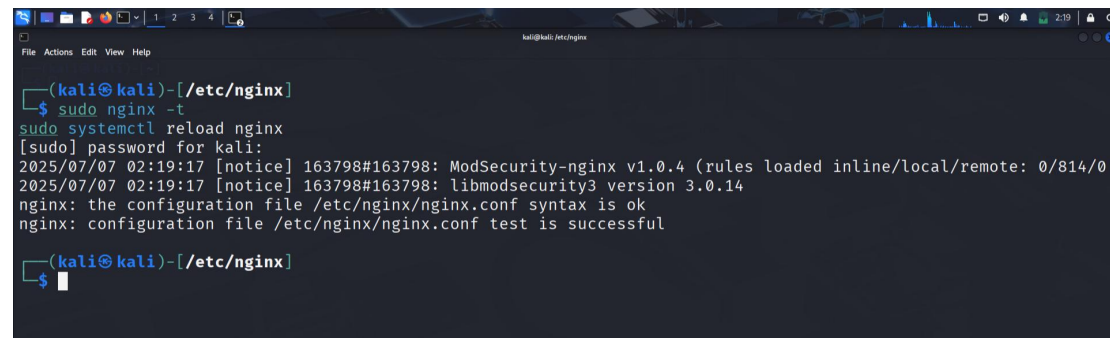
```
modsecurity_rules_file /etc/nginx/modsec/main.conf;
```

Save and exit (Ctrl + O, Enter, Ctrl + X)

Step 7: Test and reload NGINX

```
sudo nginx -t
```

```
sudo systemctl reload nginx
```



```
kali@kali:~/etc/nginx$ sudo nginx -t
sudo systemctl reload nginx
[sudo] password for kali:
2025/07/07 02:19:17 [notice] 163798#163798: ModSecurity-nginx v1.0.4 (rules loaded inline/local/remote: 0/814/0
2025/07/07 02:19:17 [notice] 163798#163798: libmodsecurity3 version 3.0.14
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful

kali@kali:~/etc/nginx$
```

Rules loaded:

inline: 0 → No rules were written directly inside the NGINX config.

local: 814 → 814 OWASP CRS rules were loaded from your main.conf. ✓

remote: 0 → No remote rules are used.

libmodsecurity3 version 3.0.14

I am using ModSecurity v3.0.14, which is the latest stable core version at this time.

nginx: the configuration file /etc/nginx/nginx.conf syntax is ok

nginx: configuration file /etc/nginx/nginx.conf test is successful

My NGINX configuration is correct and valid.

No syntax errors or warnings were found.

NGINX successfully loaded with ModSecurity integrated.

Part 2: Fail2Ban Deployment

Step 1: Install Fail2ban

```
sudo apt update
```

```
sudo apt install fail2ban -y
```

Verify Installation

```
sudo systemctl enable --now fail2ban
```

```
sudo systemctl status fail2ban
```

Should show active (running)

Step 2: Configure Jails

Fail2Ban uses jails to define which services to protect and how to protect them.

What is jail.local?

jail.local is the custom configuration file for Fail2Ban where you define your jails.

It overrides the default /etc/fail2ban/jail.conf.

Each section like [sshd] or [nginx-http-auth] represents a jail for a specific service or attack pattern.

The typical structure includes:

enabled = true: Enables the jail.

filter = <name>: Refers to a filter file in /etc/fail2ban/filter.d/<name>.conf.

logpath: Location of the log file to monitor.

maxretry, findtime, bantime: Control when bans occur.

#My Jail configurations

```
cd /etc/fail2ban/  
sudo nano jail.local
```

#add this configuration

```
[sshd]  
enabled = true  
port = ssh  
filter = sshd  
logpath = /var/log/auth.log  
maxretry = 5  
bantime = 600  
findtime = 600  
action = iptables[name=sshd, port=ssh, protocol=tcp]
```

```
[nginx-modsec]  
enabled = true  
filter = modsecurity  
logpath = /var/log/modsec_audit.log  
maxretry = 3  
bantime = 3600  
findtime = 600  
action = iptables[name=modsecurity, port=http, protocol=tcp]
```

```
[nginx-http-auth]  
enabled = true  
port = http,https  
filter = nginx-http-auth  
logpath = /var/log/nginx/error.log  
maxretry = 5  
bantime = 600  
findtime = 600  
action = iptables[name=nginx-http-auth, port=http, protocol=tcp]
```

```
[nginx-badbots]  
enabled = true  
port = http,https  
filter = nginx-badbots  
logpath = /var/log/nginx/access.log  
maxretry = 2  
bantime = 600  
findtime = 600  
action = iptables[name=nginx-badbots, port=http, protocol=tcp]
```

```
[nginx-http-errors]  
enabled = true  
port = http,https  
filter = nginx-http-errors  
logpath = /var/log/nginx/access.log  
maxretry = 5  
bantime = 3600  
findtime = 600  
action = iptables[name=nginx-http-errors, port=http, protocol=tcp]
```

Step 3: Understanding Filters and filter.d

What is filter.d?

The /etc/fail2ban/filter.d/ directory contains filter definition files (ending in .conf). Each filter defines how to identify suspicious behavior by using regular expressions to match entries in log files.

Why is it needed?

Filters help Fail2Ban understand what to look for. For example:

sshd.conf: Detects failed SSH logins.

nginx-badbots.conf: Detects known bad bots in Nginx logs.

apache-auth.conf: Detects failed login attempts via Apache.

We can create a filter under /etc/fail2ban/filter.d/nginx-http-errors.conf or apache-http-errors.conf:

[Definition]

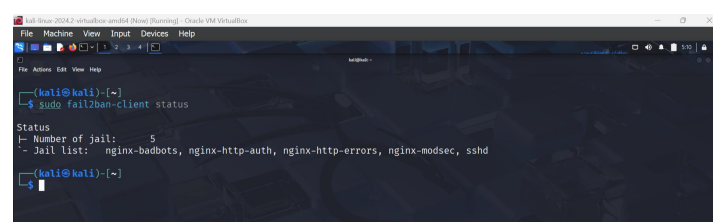
```
failregex = ^<HOST> -.*(GET|POST|HEAD).+ (401|403|404|429|500) .*  
ignoreregex =
```

For each jail.local it is must to have filter to look for

Step 4: Final Steps and Verification

Restart Fail2Ban

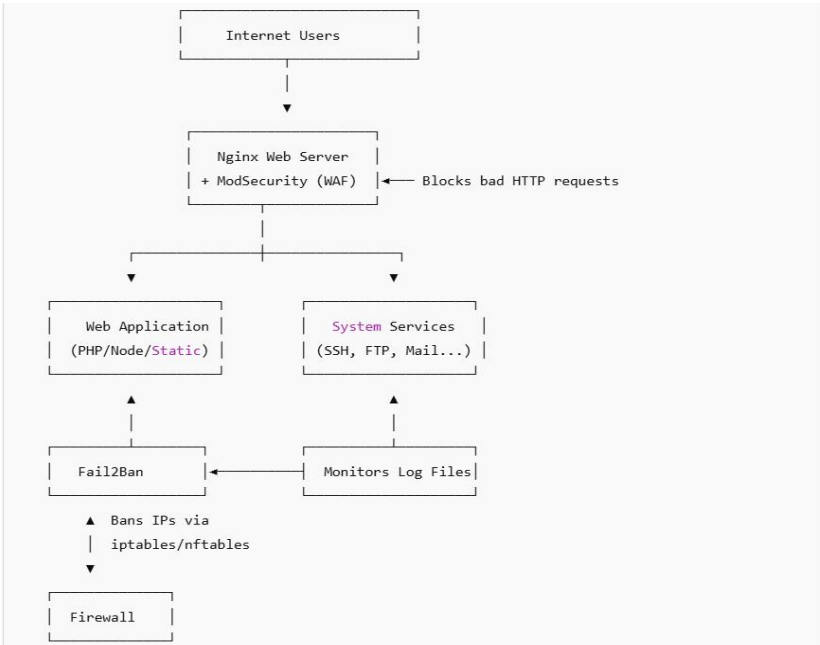
```
sudo systemctl restart fail2ban  
sudo fail2ban-client status
```



```
fail2ban-client status  
  
Status  
- Number of jail:      5  
- Jail list:  nginx-badbots, nginx-http-auth, nginx-http-errors, nginx-modsec, sshd  
  
(kali@kali)~$
```

sudo fail2ban-client status shows that Fail2Ban is running and currently monitoring 5 jails:
nginx-badbots: Blocks bad web bots.
nginx-http-auth: Protects against failed HTTP auth.
nginx-http-errors: Detects repeated HTTP errors.
nginx-modsec: Monitors ModSecurity alerts.
sshd: Protects SSH from brute-force attacks.

Deployment Diagram: ModSecurity + Fail2Ban



Recommended Deployment in Linux Web Application Stack

Component	Tool to Apply	Purpose
Nginx/Apache Web Server	ModSecurity	Filter and block bad HTTP traffic
PHP/Node/Backend App	ModSecurity	Prevent code injection, enforce input validation
SSH Access	Fail2Ban	Ban IPs after failed login attempts
FTP/Mail Services	Fail2Ban	Prevent abuse and brute force attacks
System-Wide Protection	Fail2Ban	Monitor logs for malicious behavior across services
Custom Admin Panels	ModSecurity + Fail2Ban	Use WAF for request filtering and Fail2Ban for abuse rate limiting

With ModSecurity acting as the first line of defense at the application level and Fail2Ban blocking brute-force attempts at the network level, this setup creates a multi-layered security barrier.

Whether you're running a personal website or an enterprise application, this combined strategy ensures your server is ready to defend against common and advanced web threats.