ModSecurity Deep Dive

Stack-Specific Security Configurations

JANUARY 8, 2026

THEDECIPHERIST.COM

TABLE OF CONTENTS

STACK-SPECIFIC SECURITY CONFIGURATIONS FOR NODE.JS/MONGODB VS APACHE/SQL

TABLE OF CONTENTS

UNDERSTANDING OWASP

What is OWASP?

The **Open Web Application Security Project (OWASP)** is a nonprofit foundation that works to improve the security of software. They produce:

- **The OWASP Top 10**: The most critical web application security risks
- **Core Rule Set (CRS)**: ModSecurity rules that protect against the Top 10
- **Testing Guides**: Methodologies for security testing
- **Cheat Sheets**: Quick reference security guides for developers

Why OWASP Matters

OWASP isn't just a checklist - it's the industry standard. When auditors, pen testers, or security teams evaluate your application, they're using OWASP guidelines. Compliance frameworks (PCI-DSS, HIPAA, SOC2) often reference OWASP directly.

Real talk: If you get breached and weren't following OWASP basics, you're going to have a very bad time explaining that to stakeholders, lawyers, and regulators.

THE OWASP TOP 10 (2024-2026)

Understanding each vulnerability is crucial for proper ModSecurity configuration:

A01:2021 - Broken Access Control

What it is: Users acting outside their intended permissions.

Examples:

- Accessing another user's account by changing an ID in the URL
- Privilege escalation (regular user → admin)
- CORS misconfiguration allowing unauthorized API access

ModSecurity Protection:

```
# Detect parameter tampering attempts
SecRule ARGS:user_id "!@eq %{SESSION.user_id}" \
    "id:1001,phase:2,deny,status:403,msg:'User ID tampering attempt'"

# Block directory traversal
SecRule REQUEST_URI "@contains ../" \
    "id:1002,phase:1,deny,status:403,msg:'Directory traversal attempt'"
```

A02:2021 - Cryptographic Failures

What it is: Failures related to cryptography (or lack thereof) that expose sensitive data.

Examples:

- Transmitting data in clear text (HTTP instead of HTTPS)

- Using weak/deprecated algorithms (MD5, SHA1, DES)
- Exposing sensitive data in URLs

ModSecurity Protection:

```
# Detect sensitive data in URLs (credit cards, SSNs)
SecRule REQUEST_URI "@rx \b\d{4}[- ]?\d{4}[- ]?\d{4}[- ]?\d{4}\b" \
    "id:2001,phase:1,deny,status:403,msg:'Potential credit card in URL'"

SecRule REQUEST_URI "@rx \b\d{3}-\d{2}-\d{4}\b" \
    "id:2002,phase:1,deny,status:403,msg:'Potential SSN in URL'"

# Block sensitive data in response bodies
SecRule RESPONSE_BODY "@rx \b\d{4}[- ]?\d{4}[- ]?\d{4}[- ]?\d{4}\b" \
    "id:2003,phase:4,deny,status:500,msg:'Credit card data leakage detected'"
```

A03:2021 - Injection

What it is: Hostile data sent to an interpreter as part of a command or query.

Types:

- **SQL Injection**: Traditional RDBMS attacks
- **NoSQL Injection**: MongoDB, CouchDB attacks
- **Command Injection**: OS command execution
- **LDAP Injection**: Directory service attacks

This is where Node.js/MongoDB vs Apache/SQL configurations diverge significantly. (See detailed sections below)

A04:2021 - Insecure Design

What it is: Flaws in the design and architecture, not implementation bugs.

Examples:

- No rate limiting on authentication
- Unlimited file upload sizes

- Missing business logic validation

ModSecurity Protection:

```
# Rate limit authentication endpoints
SecRule REQUEST_URI "@beginsWith /api/login" \
    "id:4001,phase:1,pass,nolog,setvar:ip.auth_attempts=+1,expirevar:ip.auth_at

SecRule IP:AUTH_ATTEMPTS "@gt 5" \
    "id:4002,phase:1,deny,status:429,msg:'Authentication rate limit exceeded'"

# Limit request body size
SecRequestBodyLimit 10485760
SecRequestBodyNoFilesLimit 131072
```

A05:2021 - Security Misconfiguration

What it is: Missing or incorrect security hardening.

Examples:

- Default credentials
- Unnecessary features enabled
- Verbose error messages exposing stack traces
- Missing security headers

ModSecurity Protection:

```
# Block common default paths
SecRule REQUEST_URI "@rx ^/(admin|manager|console|phpmyadmin|adminer)" \
    "id:5001,phase:1,deny,status:404,msg:'Admin panel access attempt'"

# Hide error details in responses
SecRule RESPONSE_BODY "@rx (stack trace|exception|error in|syntax error|mysql_)
    "id:5002,phase:4,deny,status:500,msg:'Information leakage in error'"

# Block server information disclosure
SecRule RESPONSE_HEADERS:Server "@rx ." \
    "id:5003,phase:3,pass,nolog,setvar:tx.server_header=1"
```

A06:2021 - Vulnerable and Outdated Components

What it is: Using components with known vulnerabilities.

Examples:

- Outdated libraries with CVEs
- Unsupported frameworks
- Unpatched systems

ModSecurity can't directly fix this, but it can:

```
# Block exploitation of known CVEs (example: Log4Shell)
SecRule REQUEST_HEADERS|REQUEST_BODY "@rx \$\{jndi:" \
    "id:6001,phase:2,deny,status:403,msg:'Log4Shell exploit attempt'"

# Block known malicious user agents (scanners, exploit tools)
SecRule REQUEST_HEADERS:User-Agent "@rx (nikto|sqlmap|nmap|masscan|zgrab)" \
    "id:6002,phase:1,deny,status:403,msg:'Security scanner detected'"
```

A07:2021 - Identification and Authentication Failures

What it is: Weaknesses in authentication mechanisms.

Examples:

- Credential stuffing
- Session fixation
- Weak password policies

ModSecurity Protection:

```
# Detect credential stuffing patterns
SecRule REQUEST_URI "@beginsWith /api/login" \
    "id:7001,phase:1,pass,nolog,initcol:ip=%{REMOTE_ADDR}"

SecRule &ARGS:username "@gt 0" \
    "id:7002,phase:2,pass,nolog,setvar:ip.login_usernames=%{ip.login_usernames}

# Alert on multiple usernames from same IP
SecRule IP:LOGIN_USERNAMES "@rx ([^+]+\+){10,}" \
    "id:7003,phase:2,deny,status:429,msg:'Possible credential stuffing - multip

# Session fixation protection
SecRule &REQUEST_COOKIES:session_id "@eq 0" \
    "id:7004,phase:1,pass,nolog,setvar:tx.new_session=1"

SecRule TX:NEW_SESSION "@eq 1" \
    "id:7005,phase:1,pass,nolog,ctl:sessionStart"
```

A08:2021 - Software and Data Integrity Failures

What it is: Code and infrastructure that doesn't protect against integrity violations.

Examples:

- Insecure deserialization
- CI/CD pipeline compromise
- Auto-update without verification

ModSecurity Protection:

```
# Block serialized object patterns (Java, PHP, Python)
SecRule REQUEST_BODY "@rx (rO0AB|O:\d+:\")" \
    "id:8001,phase:2,deny,status:403,msg:'Serialized object in request'"

# Block Node.js prototype pollution patterns
SecRule REQUEST_BODY "@rx (__proto__|constructor\s*\[|prototype\s*\[)" \
    "id:8002,phase:2,deny,status:403,msg:'Prototype pollution attempt'"
```

A09:2021 - Security Logging and Monitoring Failures

What it is: Insufficient logging to detect and respond to attacks.

ModSecurity Configuration for Comprehensive Logging:

```
# Enable detailed audit logging
SecAuditEngine RelevantOnly
SecAuditLogRelevantStatus "^(?:5|4(?!04))"
SecAuditLogFormat JSON
SecAuditLogType Serial
SecAuditLog /var/log/modsec_audit.json

# Log parts to capture
# A=Audit header, B=Request headers, C=Request body,
# E=Response body, F=Response headers, H=Audit trailer,
# I=Compact request body, K=Matched rules, Z=Final boundary
SecAuditLogParts ABCEFHKZ

# Always log authentication attempts
SecRule REQUEST_URI "@rx (login|signin|auth|session)" \
    "id:9001,phase:1,pass,log,auditlog,msg:'Authentication endpoint accessed'"
```

A10:2021 - Server-Side Request Forgery (SSRF)

What it is: Web application fetches a remote resource without validating the user-supplied URL.

Examples:

- Accessing internal services via URL parameter
- Cloud metadata endpoint access (169.254.169.254)
- Port scanning internal network

ModSecurity Protection:

```
# Block internal IP ranges in parameters
SecRule ARGS "@rx (127\.|10\.|172\.(1[6-9]|2[0-9]|3[01])\.|192\.168\.)" \
    "id:10001,phase:2,deny,status:403,msg:'SSRF attempt - internal IP'"

# Block cloud metadata endpoints
SecRule ARGS "@rx 169\.254\.169\.254" \
    "id:10002,phase:2,deny,status:403,msg:'SSRF attempt - AWS metadata'"

SecRule ARGS "@rx metadata\.google\.internal" \
    "id:10003,phase:2,deny,status:403,msg:'SSRF attempt - GCP metadata'"

# Block file:// protocol
SecRule ARGS "@rx ^file://" \
    "id:10004,phase:2,deny,status:403,msg:'SSRF attempt - file protocol'"
```

MODSECURITY FUNDAMENTALS

How ModSecurity Works

ModSecurity operates in **5 phases**:

PHASE	NAME	WHEN IT RUNS	WHAT YOU CAN INSPECT
1	Request Headers	After headers received	Headers, URI, method
2	Request Body	After body received	POST data, file uploads
3	Response Headers	Before headers sent	Response headers
4	Response Body	Before body sent	Response content
5	Logging	After response complete	Everything

Rule Anatomy

```
SecRule VARIABLES "OPERATOR" "ACTIONS"
```

Example breakdown:

```
SecRule REQUEST_URI|ARGS "@rx (union.*select|select.*from)" \
    "id:100001,\
     phase:2,\
     deny,\
     status:403,\
     log,\
     msg:'SQL Injection attempt',\
     severity:CRITICAL,\
     tag:'OWASP_CRS/WEB_ATTACK/SQL_INJECTION'"
```

- **VARIABLES**: `REQUEST_URI|ARGS` - What to inspect
- **OPERATOR**: `@rx` - Regular expression match
- **ACTIONS**:
- `id:100001` - Unique rule ID
- `phase:2` - Run during request body phase
- `deny` - Block the request
- `status:403` - Return 403 Forbidden
- `log` - Write to error log
- `msg` - Human-readable message
- `severity` - Alert level
- `tag` - Categorization

Detection vs Prevention Modes

```
# Detection only (log but don't block) - USE FOR INITIAL DEPLOYMENT
SecRuleEngine DetectionOnly

# Prevention mode (actively block) - USE AFTER TUNING
SecRuleEngine On

# Completely disabled
SecRuleEngine Off
```

Best Practice: Always start in `DetectionOnly` mode for 2-4 weeks to identify false positives before switching to `On`.

CORE RULE SET (CRS) EXPLAINED

What CRS Provides

The OWASP Core Rule Set is a set of generic attack detection rules. It protects against:

- SQL Injection (SQLi)
- Cross-Site Scripting (XSS)
- Local File Inclusion (LFI)
- Remote File Inclusion (RFI)
- Remote Code Execution (RCE)
- PHP/Java/Shell injection
- Session fixation
- Scanner/bot detection

CRS File Structure

```
coreruleset/
├── crs-setup.conf              # Main configuration
├── rules/
│   ├── REQUEST-901-INITIALIZATION.conf
│   ├── REQUEST-905-COMMON-EXCEPTIONS.conf
│   ├── REQUEST-911-METHOD-ENFORCEMENT.conf
│   ├── REQUEST-913-SCANNER-DETECTION.conf
│   ├── REQUEST-920-PROTOCOL-ENFORCEMENT.conf
│   ├── REQUEST-921-PROTOCOL-ATTACK.conf
│   ├── REQUEST-922-MULTIPART-ATTACK.conf
│   ├── REQUEST-930-APPLICATION-ATTACK-LFI.conf
│   ├── REQUEST-931-APPLICATION-ATTACK-RFI.conf
│   ├── REQUEST-932-APPLICATION-ATTACK-RCE.conf
│   ├── REQUEST-933-APPLICATION-ATTACK-PHP.conf
│   ├── REQUEST-934-APPLICATION-ATTACK-GENERIC.conf
│   ├── REQUEST-941-APPLICATION-ATTACK-XSS.conf
│   ├── REQUEST-942-APPLICATION-ATTACK-SQLI.conf
│   ├── REQUEST-943-APPLICATION-ATTACK-SESSION-FIXATION.conf
│   ├── REQUEST-944-APPLICATION-ATTACK-JAVA.conf
│   ├── REQUEST-949-BLOCKING-EVALUATION.conf
│   ├── RESPONSE-950-DATA-LEAKAGES.conf
│   ├── RESPONSE-951-DATA-LEAKAGES-SQL.conf
│   ├── RESPONSE-952-DATA-LEAKAGES-JAVA.conf
│   ├── RESPONSE-953-DATA-LEAKAGES-PHP.conf
│   ├── RESPONSE-954-DATA-LEAKAGES-IIS.conf
│   ├── RESPONSE-959-BLOCKING-EVALUATION.conf
│   └── RESPONSE-980-CORRELATION.conf
```

Paranoia Levels

CRS uses **Paranoia Levels (PL)** to balance security vs false positives:

LEVEL	DESCRIPTION	FALSE POSITIVES	SECURITY
PL1	Default, minimal FPs	Very Low	Good
PL2	More rules enabled	Low-Medium	Better
PL3	Stricter detection	Medium-High	High
PL4	Maximum security	High	Maximum

Configure in crs-setup.conf:

```
SecAction \
    "id:900000,\
     phase:1,\
     nolog,\
     pass,\
     t:none,\
     setvar:tx.paranoia_level=1"
```

Recommendation by environment:

- **Development**: PL1 (minimal interference)
- **Staging**: PL2 (catch more issues before prod)
- **Production (most apps)**: PL1-PL2
- **Production (high security)**: PL3
- **Financial/Healthcare**: PL3-PL4 with extensive tuning

NODE.JS + MONGODB STACK CONFIGURATION

Understanding the Attack Surface

Node.js + MongoDB stacks face **different threats** than traditional LAMP stacks:

ATTACK VECTOR	RISK LEVEL	NOTES
NoSQL Injection	**HIGH**	Different syntax than SQL
Prototype Pollution	**HIGH**	JavaScript-specific
Server-Side JS Injection	**HIGH**	eval(), Function() abuse
JSON Injection	**MEDIUM**	Malformed JSON handling
ReDoS	**MEDIUM**	Regex denial of service
Traditional SQLi	LOW	Not applicable
PHP-specific attacks	LOW	Not applicable

NoSQL Injection: The Hidden Threat

Traditional SQL Injection:

```
SELECT * FROM users WHERE username = 'admin' OR '1'='1'
```

MongoDB NoSQL Injection:

```
// Attacker sends: {"username": {"$gt": ""}, "password": {"$gt": ""}}
db.users.find({username: {"$gt": ""}, password: {"$gt": ""}})
// Returns ALL users because everything is greater than empty string!
```

Another MongoDB attack vector:

```
// Attacker sends: {"$where": "sleep(5000)"}
// Causes 5-second delay - DoS attack
```

ModSecurity Rules for Node.js/MongoDB

Create a dedicated file: `/etc/nginx/modsec/rules/nodejs_mongodb.conf`

```
# ======================================================
# NODE.JS + MONGODB SPECIFIC RULES
# ======================================================

# ------------------------------------------------------
# NOSQL INJECTION PROTECTION
# ------------------------------------------------------

# Block MongoDB operators in request parameters
SecRule ARGS|ARGS_NAMES "@rx \$(?:where|gt|lt|gte|lte|ne|in|nin|not|or|and|nor|
    "id:100001,\
     phase:2,\
     deny,\
     status:403,\
     log,\
     msg:'NoSQL Injection - MongoDB operator in parameter',\
     severity:CRITICAL,\
     tag:'OWASP_CRS/NOSQL_INJECTION'"

# Block MongoDB operators in JSON body
SecRule REQUEST_BODY "@rx \"\$(?:where|gt|lt|gte|lte|ne|in|nin|not|or|and|nor|e
    "id:100002,\
     phase:2,\
     deny,\
     status:403,\
     log,\
     msg:'NoSQL Injection - MongoDB operator in JSON body',\
     severity:CRITICAL,\
     tag:'OWASP_CRS/NOSQL_INJECTION'"

# Block JavaScript execution in MongoDB ($where with function)
SecRule REQUEST_BODY "@rx \$where\s*:\s*[\"']?function" \
    "id:100003,\
     phase:2,\
     deny,\
     status:403,\
     log,\
     msg:'NoSQL Injection - JavaScript function in $where',\
     severity:CRITICAL,\
     tag:'OWASP_CRS/NOSQL_INJECTION'"

# Block mapReduce abuse
SecRule REQUEST_BODY "@rx mapReduce|map\s*:\s*function|reduce\s*:\s*function" \
    "id:100004,\
     phase:2,\
     deny,\
```

```
     status:403,\
     log,\
     msg:'NoSQL Injection - mapReduce attempt',\
     severity:CRITICAL,\
     tag:'OWASP_CRS/NOSQL_INJECTION'"

# ------------------------------------------------------
# PROTOTYPE POLLUTION PROTECTION
# ------------------------------------------------------

# Block __proto__ access
SecRule ARGS|ARGS_NAMES|REQUEST_BODY "@rx __proto__" \
    "id:100010,\
     phase:2,\
     deny,\
     status:403,\
     log,\
     msg:'Prototype Pollution - __proto__ detected',\
     severity:CRITICAL,\
     tag:'NODEJS/PROTOTYPE_POLLUTION'"

# Block constructor.prototype access
SecRule ARGS|ARGS_NAMES|REQUEST_BODY "@rx constructor\s*(\[|\.).*prototype" \
    "id:100011,\
     phase:2,\
     deny,\
     status:403,\
     log,\
     msg:'Prototype Pollution - constructor.prototype detected',\
     severity:CRITICAL,\
     tag:'NODEJS/PROTOTYPE_POLLUTION'"

# Block prototype assignment patterns
SecRule REQUEST_BODY "@rx [\"']prototype[\"']\s*:" \
    "id:100012,\
     phase:2,\
     deny,\
     status:403,\
     log,\
     msg:'Prototype Pollution - prototype key in JSON',\
     severity:CRITICAL,\
     tag:'NODEJS/PROTOTYPE_POLLUTION'"

# ------------------------------------------------------
# SERVER-SIDE JAVASCRIPT INJECTION
# ------------------------------------------------------
```

```
# Block eval() patterns
SecRule REQUEST_BODY "@rx eval\s*\(" \
    "id:100020,\
     phase:2,\
     deny,\
     status:403,\
     log,\
     msg:'SSJS Injection - eval() detected',\
     severity:CRITICAL,\
     tag:'NODEJS/CODE_INJECTION'"

# Block Function constructor
SecRule REQUEST_BODY "@rx new\s+Function\s*\(" \
    "id:100021,\
     phase:2,\
     deny,\
     status:403,\
     log,\
     msg:'SSJS Injection - Function constructor detected',\
     severity:CRITICAL,\
     tag:'NODEJS/CODE_INJECTION'"

# Block setTimeout/setInterval with string (code execution)
SecRule REQUEST_BODY "@rx set(?:Timeout|Interval)\s*\(\s*[\"']" \
    "id:100022,\
     phase:2,\
     deny,\
     status:403,\
     log,\
     msg:'SSJS Injection - setTimeout/setInterval with string',\
     severity:CRITICAL,\
     tag:'NODEJS/CODE_INJECTION'"

# Block require() injection
SecRule REQUEST_BODY "@rx require\s*\(\s*[\"'][^\"']*[\"']\s*\)" \
    "id:100023,\
     phase:2,\
     deny,\
     status:403,\
     log,\
     msg:'SSJS Injection - require() in user input',\
     severity:CRITICAL,\
     tag:'NODEJS/CODE_INJECTION'"

# Block child_process patterns
SecRule REQUEST_BODY "@rx child_process|spawn\s*\(|exec\s*\(|execSync|spawnSync
    "id:100024,\
```

```
     phase:2,\
     deny,\
     status:403,\
     log,\
     msg:'SSJS Injection - child_process module access',\
     severity:CRITICAL,\
     tag:'NODEJS/CODE_INJECTION'"

# ------------------------------------------------------
# JSON-SPECIFIC ATTACKS
# ------------------------------------------------------

# Block oversized JSON (DoS prevention)
SecRequestBodyLimit 10485760
SecRequestBodyNoFilesLimit 1048576

# Validate JSON content type has valid body
SecRule REQUEST_HEADERS:Content-Type "@contains application/json" \
    "id:100030,\
     phase:1,\
     pass,\
     nolog,\
     setvar:tx.json_request=1"

SecRule TX:JSON_REQUEST "@eq 1" \
    "chain,\
     id:100031,\
     phase:2,\
     deny,\
     status:400,\
     msg:'Invalid JSON body'"
    SecRule REQUEST_BODY "!@rx ^[\s]*[\[\{]"

# Block JSON with excessive nesting (DoS)
SecRule REQUEST_BODY "@rx \{[^\}]*\{[^\}]*\{[^\}]*\{[^\}]*\{[^\}]*\{[^\}]*\{[^\
    "id:100032,\
     phase:2,\
     deny,\
     status:400,\
     msg:'JSON nesting too deep - potential DoS',\
     severity:WARNING,\
     tag:'NODEJS/DOS'"

# ------------------------------------------------------
# REGEX DENIAL OF SERVICE (ReDoS)
# ------------------------------------------------------
```

```
# Block patterns known to cause ReDoS
SecRule ARGS "@rx (a{100,}|.{1000,}|\(.*\){50,})" \
    "id:100040,\
     phase:2,\
     deny,\
     status:400,\
     msg:'Potential ReDoS pattern detected',\
     severity:WARNING,\
     tag:'NODEJS/REDOS'"

# ------------------------------------------------------
# NODE.JS SPECIFIC PATHS
# ------------------------------------------------------

# Block access to node_modules
SecRule REQUEST_URI "@contains node_modules" \
    "id:100050,\
     phase:1,\
     deny,\
     status:404,\
     msg:'node_modules access attempt'"

# Block access to package.json
SecRule REQUEST_URI "@endsWith package.json" \
    "id:100051,\
     phase:1,\
     deny,\
     status:404,\
     msg:'package.json access attempt'"

# Block .env files
SecRule REQUEST_URI "@rx \.env" \
    "id:100052,\
     phase:1,\
     deny,\
     status:404,\
     msg:'.env file access attempt'"
```

CRS Rules to DISABLE for Node.js/MongoDB

Some CRS rules cause false positives with JSON APIs:

```
# /etc/nginx/modsec/rules/nodejs_exclusions.conf

# Disable SQL injection rules (not relevant for MongoDB)
# Keep these if you have ANY SQL database in your stack
SecRuleRemoveById 942100-942999

# Disable rules that flag legitimate JSON
SecRule REQUEST_URI "@beginsWith /api/" \
    "id:100100,\
     phase:1,\
     pass,\
     nolog,\
     ctl:ruleRemoveById=921130,\
     ctl:ruleRemoveById=921180"

# Disable PHP rules (not relevant for Node.js)
SecRuleRemoveById 933100-933999

# Disable Java rules (not relevant for Node.js)
SecRuleRemoveById 944100-944999
```

Recommended CRS Rules for Node.js Stack

```
# /etc/nginx/modsec/main.conf for Node.js

Include "/etc/nginx/modsec/modsecurity.conf"
Include "/etc/nginx/modsec/coreruleset/crs-setup.conf"

# Core initialization
include "/etc/nginx/modsec/coreruleset/rules/REQUEST-901-INITIALIZATION.conf"
include "/etc/nginx/modsec/coreruleset/rules/REQUEST-905-COMMON-EXCEPTIONS.conf

# Protocol and method enforcement (KEEP)
include "/etc/nginx/modsec/coreruleset/rules/REQUEST-911-METHOD-ENFORCEMENT.con
include "/etc/nginx/modsec/coreruleset/rules/REQUEST-913-SCANNER-DETECTION.conf
include "/etc/nginx/modsec/coreruleset/rules/REQUEST-920-PROTOCOL-ENFORCEMENT.c
include "/etc/nginx/modsec/coreruleset/rules/REQUEST-921-PROTOCOL-ATTACK.conf"

# File inclusion (KEEP - still relevant)
include "/etc/nginx/modsec/coreruleset/rules/REQUEST-930-APPLICATION-ATTACK-LFI
include "/etc/nginx/modsec/coreruleset/rules/REQUEST-931-APPLICATION-ATTACK-RFI

# Remote code execution (KEEP)
include "/etc/nginx/modsec/coreruleset/rules/REQUEST-932-APPLICATION-ATTACK-RCE

# Generic attacks (KEEP)
include "/etc/nginx/modsec/coreruleset/rules/REQUEST-934-APPLICATION-ATTACK-GEN

# XSS (KEEP - still very relevant)
include "/etc/nginx/modsec/coreruleset/rules/REQUEST-941-APPLICATION-ATTACK-XSS

# SQL Injection - CONDITIONALLY INCLUDE
# Only include if you have ANY SQL database in your architecture
# include "/etc/nginx/modsec/coreruleset/rules/REQUEST-942-APPLICATION-ATTACK-S

# Session fixation (KEEP)
include "/etc/nginx/modsec/coreruleset/rules/REQUEST-943-APPLICATION-ATTACK-SES

# Blocking evaluation
include "/etc/nginx/modsec/coreruleset/rules/REQUEST-949-BLOCKING-EVALUATION.co

# Response leakage (KEEP)
include "/etc/nginx/modsec/coreruleset/rules/RESPONSE-950-DATA-LEAKAGES.conf"

# Final evaluation
include "/etc/nginx/modsec/coreruleset/rules/RESPONSE-959-BLOCKING-EVALUATION.c
include "/etc/nginx/modsec/coreruleset/rules/RESPONSE-980-CORRELATION.conf"

# SKIP PHP RULES - Not relevant for Node.js
```

```
# include "/etc/nginx/modsec/coreruleset/rules/REQUEST-933-APPLICATION-ATTACK-P
# include "/etc/nginx/modsec/coreruleset/rules/RESPONSE-953-DATA-LEAKAGES-PHP.c

# SKIP JAVA RULES - Not relevant for Node.js
# include "/etc/nginx/modsec/coreruleset/rules/REQUEST-944-APPLICATION-ATTACK-J
# include "/etc/nginx/modsec/coreruleset/rules/RESPONSE-952-DATA-LEAKAGES-JAVA.

# SKIP IIS RULES - Not relevant
# include "/etc/nginx/modsec/coreruleset/rules/RESPONSE-954-DATA-LEAKAGES-IIS.c

# Custom Node.js/MongoDB rules
Include "/etc/nginx/modsec/rules/nodejs_mongodb.conf"
Include "/etc/nginx/modsec/rules/nodejs_exclusions.conf"
```

APACHE + SQL STACK CONFIGURATION

Understanding the Attack Surface

Traditional LAMP/Apache+SQL stacks face these primary threats:

ATTACK VECTOR	RISK LEVEL	NOTES
SQL Injection	CRITICAL	Most common, most dangerous
Cross-Site Scripting	HIGH	Especially in PHP apps
PHP-specific attacks	HIGH	File inclusion, object injection
Remote Code Execution	HIGH	Often via PHP
File Upload attacks	MEDIUM	Shell upload
NoSQL Injection	LOW	Not applicable
Prototype Pollution	LOW	Not JavaScript

SQL Injection: The Classic Threat

Types of SQL Injection:

1. **In-band SQLi** (most common)

- Error-based: Uses error messages to extract data
- Union-based: Uses UNION to combine results

1. **Blind SQLi**

- Boolean-based: True/false responses
- Time-based: Uses delays to infer data

1. **Out-of-band SQLi**

- Uses external channels (DNS, HTTP requests)

Example attacks:

```
-- Classic bypass
admin' OR '1'='1' --

-- UNION attack
' UNION SELECT username, password FROM users --

-- Time-based blind
'; WAITFOR DELAY '00:00:05' --

-- Stacked queries
'; DROP TABLE users; --
```

ModSecurity Rules for Apache/SQL

Create: `/etc/nginx/modsec/rules/apache_sql.conf`

```
# =====================================================
# APACHE + SQL SPECIFIC RULES
# =====================================================

# -----------------------------------------------------
# ENHANCED SQL INJECTION PROTECTION
# -----------------------------------------------------

# Block SQL comments used to bypass filters
SecRule ARGS|REQUEST_BODY "@rx (\/\*.*?\*\/|--\s|#\s*$)" \
    "id:200001,\
     phase:2,\
     deny,\
     status:403,\
     log,\
     msg:'SQL Injection - Comment pattern detected',\
     severity:CRITICAL,\
     tag:'OWASP_CRS/SQL_INJECTION'"

# Block UNION-based injection
SecRule ARGS|REQUEST_BODY "@rx (?i)union\s+(all\s+)?select" \
    "id:200002,\
     phase:2,\
     deny,\
     status:403,\
     log,\
     msg:'SQL Injection - UNION SELECT detected',\
     severity:CRITICAL,\
     tag:'OWASP_CRS/SQL_INJECTION'"

# Block stacked queries
SecRule ARGS|REQUEST_BODY "@rx ;\s*(SELECT|INSERT|UPDATE|DELETE|DROP|CREATE|ALT
    "id:200003,\
     phase:2,\
     deny,\
     status:403,\
     log,\
     msg:'SQL Injection - Stacked query detected',\
     severity:CRITICAL,\
     tag:'OWASP_CRS/SQL_INJECTION'"

# Block time-based blind SQLi
SecRule ARGS|REQUEST_BODY "@rx (?i)(WAITFOR\s+DELAY|SLEEP\s*\(|BENCHMARK\s*\(|p
    "id:200004,\
     phase:2,\
     deny,\
```

```
     status:403,\
     log,\
     msg:'SQL Injection - Time-based attack detected',\
     severity:CRITICAL,\
     tag:'OWASP_CRS/SQL_INJECTION'"

# Block boolean-based blind SQLi patterns
SecRule ARGS|REQUEST_BODY "@rx (?i)(AND|OR)\s+\d+\s*=\s*\d+" \
    "id:200005,\
     phase:2,\
     deny,\
     status:403,\
     log,\
     msg:'SQL Injection - Boolean-based pattern detected',\
     severity:HIGH,\
     tag:'OWASP_CRS/SQL_INJECTION'"

# Block SQL functions often used in attacks
SecRule ARGS|REQUEST_BODY "@rx (?i)(CONCAT\s*\(|CHAR\s*\(|ASCII\s*\(|SUBSTRING\
    "id:200006,\
     phase:2,\
     deny,\
     status:403,\
     log,\
     msg:'SQL Injection - Dangerous function detected',\
     severity:CRITICAL,\
     tag:'OWASP_CRS/SQL_INJECTION'"

# Block information_schema access
SecRule ARGS|REQUEST_BODY "@rx (?i)information_schema|sys\.tables|sysobjects" \
    "id:200007,\
     phase:2,\
     deny,\
     status:403,\
     log,\
     msg:'SQL Injection - Schema enumeration attempt',\
     severity:CRITICAL,\
     tag:'OWASP_CRS/SQL_INJECTION'"

# ------------------------------------------------------
# PHP-SPECIFIC PROTECTION
# ------------------------------------------------------

# Block PHP object injection patterns
SecRule ARGS|REQUEST_BODY "@rx [oOcC]:\d+:" \
    "id:200020,\
     phase:2,\
```

```
     deny,\
     status:403,\
     log,\
     msg:'PHP Object Injection attempt',\
     severity:CRITICAL,\
     tag:'PHP/OBJECT_INJECTION'"

# Block dangerous PHP functions in input
SecRule ARGS|REQUEST_BODY "@rx (?i)(system|exec|shell_exec|passthru|popen|proc_
    "id:200021,\
     phase:2,\
     deny,\
     status:403,\
     log,\
     msg:'PHP Code Injection - Dangerous function',\
     severity:CRITICAL,\
     tag:'PHP/CODE_INJECTION'"

# Block PHP wrappers
SecRule ARGS|REQUEST_URI "@rx (?i)(php|data|expect|input|filter)://" \
    "id:200022,\
     phase:2,\
     deny,\
     status:403,\
     log,\
     msg:'PHP Wrapper attack attempt',\
     severity:CRITICAL,\
     tag:'PHP/WRAPPER_ATTACK'"

# Block include/require with user input patterns
SecRule ARGS "@rx (?i)(\.\.\/|\.\.\\|\/etc\/|c:\\)" \
    "id:200023,\
     phase:2,\
     deny,\
     status:403,\
     log,\
     msg:'PHP File Inclusion - Path traversal',\
     severity:CRITICAL,\
     tag:'PHP/FILE_INCLUSION'"

# Block PHP info disclosure
SecRule REQUEST_URI "@rx (?i)phpinfo|php_info" \
    "id:200024,\
     phase:1,\
     deny,\
     status:404,\
     log,\
```

```
     msg:'PHPInfo access attempt'"

# ------------------------------------------------------
# SQL DATA LEAKAGE PROTECTION
# ------------------------------------------------------

# Block SQL error messages in response
SecRule RESPONSE_BODY "@rx (?i)(mysql_fetch|mysqli_|pg_query|sqlite_|ORA-\d{5}|
    "id:200030,\
     phase:4,\
     deny,\
     status:500,\
     log,\
     msg:'SQL Error Information Leakage',\
     severity:CRITICAL,\
     tag:'LEAKAGE/SQL_ERROR'"

# Block stack traces in response
SecRule RESPONSE_BODY "@rx (?i)(stack trace|backtrace|debug|exception in|error
    "id:200031,\
     phase:4,\
     deny,\
     status:500,\
     log,\
     msg:'Application Error Leakage',\
     severity:HIGH,\
     tag:'LEAKAGE/DEBUG'"

# ------------------------------------------------------
# FILE UPLOAD PROTECTION
# ------------------------------------------------------

# Block dangerous file extensions
SecRule FILES_NAMES "@rx (?i)\.(php|phtml|php[3457]|phar|inc|pl|py|rb|cgi|sh|ba
    "id:200040,\
     phase:2,\
     deny,\
     status:403,\
     log,\
     msg:'Dangerous file upload attempt',\
     severity:CRITICAL,\
     tag:'FILE_UPLOAD/DANGEROUS_EXT'"

# Block double extensions
SecRule FILES_NAMES "@rx \.[a-z0-9]+\.(php|phtml|asp|aspx|jsp)[.]*$" \
    "id:200041,\
     phase:2,\
```

```
     deny,\
     status:403,\
     log,\
     msg:'Double extension file upload attempt',\
     severity:CRITICAL,\
     tag:'FILE_UPLOAD/DOUBLE_EXT'"

# Verify file content matches extension
SecRule FILES_TMPNAMES "@inspectFile /path/to/check_file_type.sh" \
    "id:200042,\
     phase:2,\
     deny,\
     status:403,\
     log,\
     msg:'File content does not match extension'"

# ------------------------------------------------------
# APACHE-SPECIFIC PATHS
# ------------------------------------------------------

# Block .htaccess access
SecRule REQUEST_URI "@rx /\.ht" \
    "id:200050,\
     phase:1,\
     deny,\
     status:404,\
     msg:'.htaccess access attempt'"

# Block Apache server-status
SecRule REQUEST_URI "@rx /server-(status|info)" \
    "id:200051,\
     phase:1,\
     deny,\
     status:404,\
     msg:'Apache status page access attempt'"
```

Recommended CRS Rules for Apache/SQL Stack

```
# /etc/nginx/modsec/main.conf for Apache/SQL

Include "/etc/nginx/modsec/modsecurity.conf"
Include "/etc/nginx/modsec/coreruleset/crs-setup.conf"

# Core initialization
include "/etc/nginx/modsec/coreruleset/rules/REQUEST-901-INITIALIZATION.conf"
include "/etc/nginx/modsec/coreruleset/rules/REQUEST-905-COMMON-EXCEPTIONS.conf

# Protocol and method enforcement (KEEP)
include "/etc/nginx/modsec/coreruleset/rules/REQUEST-911-METHOD-ENFORCEMENT.con
include "/etc/nginx/modsec/coreruleset/rules/REQUEST-913-SCANNER-DETECTION.conf
include "/etc/nginx/modsec/coreruleset/rules/REQUEST-920-PROTOCOL-ENFORCEMENT.c
include "/etc/nginx/modsec/coreruleset/rules/REQUEST-921-PROTOCOL-ATTACK.conf"
include "/etc/nginx/modsec/coreruleset/rules/REQUEST-922-MULTIPART-ATTACK.conf"

# File inclusion (KEEP - critical for PHP)
include "/etc/nginx/modsec/coreruleset/rules/REQUEST-930-APPLICATION-ATTACK-LFI
include "/etc/nginx/modsec/coreruleset/rules/REQUEST-931-APPLICATION-ATTACK-RFI

# Remote code execution (KEEP)
include "/etc/nginx/modsec/coreruleset/rules/REQUEST-932-APPLICATION-ATTACK-RCE

# PHP attacks (KEEP - CRITICAL for PHP stack)
include "/etc/nginx/modsec/coreruleset/rules/REQUEST-933-APPLICATION-ATTACK-PHP

# Generic attacks (KEEP)
include "/etc/nginx/modsec/coreruleset/rules/REQUEST-934-APPLICATION-ATTACK-GEN

# XSS (KEEP)
include "/etc/nginx/modsec/coreruleset/rules/REQUEST-941-APPLICATION-ATTACK-XSS

# SQL Injection (KEEP - CRITICAL for SQL stack)
include "/etc/nginx/modsec/coreruleset/rules/REQUEST-942-APPLICATION-ATTACK-SQL

# Session fixation (KEEP)
include "/etc/nginx/modsec/coreruleset/rules/REQUEST-943-APPLICATION-ATTACK-SES

# Java attacks (include if using any Java components)
# include "/etc/nginx/modsec/coreruleset/rules/REQUEST-944-APPLICATION-ATTACK-J

# Blocking evaluation
include "/etc/nginx/modsec/coreruleset/rules/REQUEST-949-BLOCKING-EVALUATION.co

# Response leakage detection (KEEP ALL)
include "/etc/nginx/modsec/coreruleset/rules/RESPONSE-950-DATA-LEAKAGES.conf"
```

```
include "/etc/nginx/modsec/coreruleset/rules/RESPONSE-951-DATA-LEAKAGES-SQL.cor
include "/etc/nginx/modsec/coreruleset/rules/RESPONSE-953-DATA-LEAKAGES-PHP.cor

# Final evaluation
include "/etc/nginx/modsec/coreruleset/rules/RESPONSE-959-BLOCKING-EVALUATION.c
include "/etc/nginx/modsec/coreruleset/rules/RESPONSE-980-CORRELATION.conf"

# Custom Apache/SQL rules
Include "/etc/nginx/modsec/rules/apache_sql.conf"
```

STACK COMPARISON: WHAT CHANGES AND WHY

Quick Reference Table

RULE CATEGORY	NODE.JS/MONGODB	APACHE/SQL	WHY
SQL Injection (942xxx)	❌ Disable	✅ Enable	No SQL database
NoSQL Injection	✅ Custom rules	❌ Not needed	No NoSQL database
PHP Attacks (933xxx)	❌ Disable	✅ Enable	No PHP runtime
Java Attacks (944xxx)	❌ Disable	⚠️ Optional	Depends on stack
XSS (941xxx)	✅ Enable	✅ Enable	Both vulnerable
Prototype Pollution	✅ Custom rules	❌ Not needed	JavaScript specific
LFI/RFI (930-931xxx)	✅ Enable	✅ Enable	Both vulnerable
RCE (932xxx)	✅ Enable	✅ Enable	Both vulnerable
File Upload	⚠️ Modified	✅ Enable	Different risk profile

Unless you have ANY SQL database in your architecture

Configuration Files Summary

Node.js/MongoDB Stack Files:

```
/etc/nginx/modsec/
├── modsecurity.conf          # Base config
├── main.conf                 # Rule includes (see Node.js section)
├── coreruleset/
│   └── crs-setup.conf       # PL1 or PL2
└── rules/
    ├── nodejs_mongodb.conf   # Custom NoSQL/prototype pollution rules
    └── nodejs_exclusions.conf # Rules to disable
```

Apache/SQL Stack Files:

```
/etc/nginx/modsec/
├── modsecurity.conf          # Base config
├── main.conf                 # Rule includes (see Apache section)
├── coreruleset/
│   └── crs-setup.conf       # PL2 or PL3
└── rules/
    └── apache_sql.conf       # Enhanced SQL/PHP rules
```

Performance Considerations

STACK	CPU IMPACT	MEMORY	NOTES
Node.js/MongoDB	Lower	Lower	Fewer rules enabled
Apache/SQL	Higher	Higher	More regex processing for SQLi

Tip: For high-traffic Node.js APIs, consider moving some ModSecurity rules to the application layer using libraries like `express-validator` or `joi` for better performance.

ADVANCED MODSECURITY TUNING

Anomaly Scoring Mode

Instead of blocking on single rule matches, accumulate a score:

```
# In crs-setup.conf
SecAction \
    "id:900110,\
     phase:1,\
     nolog,\
     pass,\
     t:none,\
     setvar:tx.inbound_anomaly_score_threshold=5,\
     setvar:tx.outbound_anomaly_score_threshold=4"
```

Scoring thresholds by paranoia level:

PL	INBOUND THRESHOLD	OUTBOUND THRESHOLD
1	5	4
2	10	8
3	15	12
4	20	16

Writing Exclusion Rules

When legitimate requests trigger false positives:

```
# Method 1: Disable rule for specific URI
SecRule REQUEST_URI "@beginsWith /api/webhook" \
    "id:999001,phase:1,pass,nolog,\
     ctl:ruleRemoveById=942100"

# Method 2: Disable rule for specific parameter
SecRule REQUEST_URI "@beginsWith /api/content" \
    "id:999002,phase:1,pass,nolog,\
     ctl:ruleRemoveTargetById=941100;ARGS:body"

# Method 3: Disable rule for specific IP (internal services)
SecRule REMOTE_ADDR "@ipMatch 10.0.0.0/8" \
    "id:999003,phase:1,pass,nolog,\
     ctl:ruleRemoveById=913100"

# Method 4: Whitelist entire path (use sparingly!)
SecRule REQUEST_URI "@beginsWith /internal/health" \
    "id:999004,phase:1,pass,nolog,\
     ctl:ruleEngine=Off"
```

Performance Optimization

```
# Limit what gets inspected
SecRequestBodyLimit 10485760        # 10MB max
SecRequestBodyNoFilesLimit 1048576  # 1MB without files
SecRequestBodyInMemoryLimit 131072  # 128KB in memory

# Skip static files
SecRule REQUEST_URI "@rx \.(css|js|gif|jpg|jpeg|png|ico|woff|woff2|ttf|svg)$" \
    "id:999100,phase:1,pass,nolog,\
     ctl:ruleEngine=Off"

# Skip health checks
SecRule REQUEST_URI "@streq /health" \
    "id:999101,phase:1,pass,nolog,\
     ctl:ruleEngine=Off"

# Disable response body inspection for large responses
SecResponseBodyLimit 524288
SecResponseBodyLimitAction ProcessPartial
```

MONITORING AND INCIDENT RESPONSE

Log Analysis

JSON audit log format:

```
SecAuditLogFormat JSON
SecAuditLog /var/log/modsec_audit.json
```

Sample log entry:

```
{
  "transaction": {
    "time_stamp": "2026-01-15T10:30:45.123Z",
    "unique_id": "abc123",
    "remote_address": "192.168.1.100",
    "request": {
      "method": "POST",
      "uri": "/api/login",
      "headers": {...}
    },
    "response": {
      "status": 403
    },
    "messages": [
      {
        "ruleId": "942100",
        "message": "SQL Injection Attack Detected",
        "severity": "CRITICAL"
      }
    ]
  }
}
```

Alerting Rules

Set up alerts for:

CONDITION	ALERT LEVEL	RESPONSE
>10 blocks from same IP in 1 min	HIGH	Auto-ban IP
Any rule 200xxx (custom critical)	CRITICAL	Immediate review
>100 blocks in 5 minutes (any IP)	HIGH	Possible DDoS
SQL injection patterns	CRITICAL	Review + potential ban
New attack pattern trending	MEDIUM	Update rules

Integration with SIEM

```
# Send logs to syslog for SIEM integration
SecAuditLog "| /usr/bin/logger -p local3.info -t modsecurity"

# Or pipe to a log shipper
SecAuditLog "| /usr/bin/filebeat -c /etc/filebeat/modsec.yml"
```

COMMON PITFALLS AND HOW TO AVOID THEM

Pitfall 1: Going Straight to Production with Full Rules

Problem: Enable all rules in blocking mode → Legitimate traffic blocked → Angry users

Solution:

1. Start with `SecRuleEngine DetectionOnly`
2. Monitor logs for 2-4 weeks
3. Create exclusions for false positives
4. Gradually enable blocking mode
5. Start with PL1, increase as needed

Pitfall 2: Not Updating Rules

Problem: CRS rules are updated regularly. Old rules miss new attacks.

Solution:

```
# Create update script
#!/bin/bash
cd /etc/nginx/modsec/coreruleset
git pull origin v4.0/main
nginx -t && nginx -s reload
```

Run monthly or subscribe to OWASP CRS announcements.

Pitfall 3: Over-Blocking APIs

Problem: JSON APIs trigger WAF rules because JSON looks like attacks.

Solution:

```
# Identify API endpoints and tune specifically
SecRule REQUEST_URI "@beginsWith /api/" \
    "id:999200,phase:1,pass,nolog,\
     setvar:tx.api_request=1"

# Relax certain rules for API requests
SecRule TX:API_REQUEST "@eq 1" \
    "id:999201,phase:2,pass,nolog,\
     ctl:ruleRemoveById=921130,\
     ctl:ruleRemoveById=921180"
```

Pitfall 4: Ignoring Response Body Rules

Problem: Focusing only on request filtering, missing data leakage.

Solution: Always enable response body inspection:

```
SecResponseBodyAccess On
SecResponseBodyMimeType text/plain text/html text/xml application/json
```

Pitfall 5: Not Testing Rule Changes

Problem: Rule changes break production.

Solution:

1. Test in staging first
2. Use `SecRuleEngine DetectionOnly` initially
3. Have rollback plan ready
4. Test with known attack payloads:

```
# Test SQL injection detection
curl -X POST "https://staging.example.com/api/test" \
    -d "user=admin' OR '1'='1"

# Test XSS detection
curl "https://staging.example.com/search?q=<script>alert(1)</script>"

# Test NoSQL injection (for MongoDB)
curl -X POST "https://staging.example.com/api/login" \
    -H "Content-Type: application/json" \
    -d '{"username":{"$gt":""},"password":{"$gt":""}}'
```

FINAL CHECKLIST

Before Going to Production

- [ ] Choose correct stack configuration (Node.js/MongoDB vs Apache/SQL)
- [ ] Set appropriate Paranoia Level (start with PL1)
- [ ] Enable anomaly scoring mode
- [ ] Configure proper logging (JSON format)
- [ ] Set up log rotation
- [ ] Test with `DetectionOnly` mode for 2+ weeks
- [ ] Create exclusions for all false positives
- [ ] Set up monitoring and alerting
- [ ] Document all custom rules
- [ ] Create rollback procedure
- [ ] Train team on log analysis

Monthly Maintenance

- [ ] Review blocked requests for new patterns
- [ ] Update CRS rules
- [ ] Review and adjust thresholds
- [ ] Check for new CVEs affecting your stack
- [ ] Audit exclusion rules (are they still needed?)
- [ ] Test detection with new attack payloads

This guide represents real-world production experience. Security is an ongoing process, not a one-time setup. Stay vigilant, keep learning, and update your defenses regularly.