MAGEBEAN SECURITY BASELINE V1.0

Author: Son Cao

Date: 2025-08-01

Version: 1.0

Introduction

The Magebean Security Baseline defines 12 Controls and 81 Logic Rules to evaluate the security, configuration, and dependency hygiene of Magento 2 stores.

These controls are explicitly designed to provide concrete verification steps to address OWASP Top 10: The Ten Most Critical Web Application Security Risks.

This baseline is designed to be implemented with the magebean-cli tool, which provides automated validation, reporting, and CI/CD integration.

Table of Contents

Chapter 1. Key Terminology

The Magebean Baseline v1 defines a set of **12 Controls and 81 Rules** for auditing security, configuration, and operations in Magento 2.

This chapter introduces the key terms used throughout the document, designed for readers who may not be familiar with audit and AppSec terminology.

Control

A Control is a high-level domain of security assurance, representing a critical aspect of application security or compliance (e.g., access control, data encryption, extension governance). Controls group multiple verification checks or requirements under a common theme, ensuring that all relevant risk areas are addressed.

Where a Control is missing or weak, risks inevitably emerge, making Controls the foundation for preventing or mitigating application security risks.

Example:

Control: Password Management — requires strong password policies.

Control: Extension Governance — requires extensions to be free of known vulnerabilities.

Rule

A Rule is a specific, measurable check that determines whether a system complies with a Control. Rules are typically automatable and serve as the unit of measurement for an audit.

Rules bring detail and precision, allowing evidence-based verification of compliance. A Control may consist of multiple Rules.

Example:

Rule: Admin passwords must be at least 8 characters long and contain uppercase, lowercase, numeric, and special characters.

Rule: Extension vendor/module found in composer.lock must not appear in any public CVE list.

Relationship Between Controls and Rules

Each Control consists of multiple Rules that together enforce its intent.

Example:

Control: Administrator Account Security

Associated Rules:

Rule 1: Admin passwords must meet minimum complexity requirements.

Rule 2: Two-Factor Authentication must be enabled for all admin accounts.

Rule 3: No admin account should use the default username admin.

Baseline

A Baseline in the Magebean context is the consolidated standard consisting of 12 Controls and 81 Rules. It represents the minimum recommended security posture for a Magento 2 deployment.

The baseline serves as a reference framework for developers, agencies, and merchants to assess and improve their security posture.

Scan and Audit

Audit: The structured process of reviewing and evaluating a system against Controls and Rules.

Scan: The technical execution (performed by Magebean CLI) of automated checks against Rules, producing a report with findings and remediation guidance.

Chapter 2. Scope & Objectives

The Magebean Security Baseline is designed to establish a minimum security standard for Magento 2 deployments.

It defines the scope, intended audience, and objectives of applying the **12 Controls and 81 Rules**.

Scope

System Components: Magento 2 application codebase, extensions, server configuration, dependencies, and integrations.

Security Domains: File permissions, admin interface hardening, secure coding practices, cryptography, deployment hygiene, extension governance, and third-party integrations.

Environments: Applicable to development, staging, and production environments, with emphasis on production hardening.

Intended Audience

Developers: To validate code quality and security practices before deployment.

Agencies: To ensure delivery quality, compliance, and client assurance.

Merchants: To monitor live stores and maintain operational security hygiene.

Objectives

- Provide a repeatable audit framework for Magento 2 security.
- Align Magento security checks with OWASP Top 10 and industry standards.
- Enable automated validation using magebean-cli in CI/CD pipelines.
- Facilitate remediation guidance and continuous improvement across teams.

Chapter 3. Controls & Rules Catalog

Magebean 12 Controls

MB-C01 File & Folder Permissions

MB-C02 Admin Hardening

MB-C03 Secure Coding Practices

MB-C04 HTTPS & TLS Enforcement

MB-C05 Production Mode & Deployment Hygiene

MB-C06 Cache & Indexing Health

MB-C07 Logging & Monitoring

MB-C08 Cron Job Reliability

MB-C09 Extension Vulnerability Management

MB-C10 Abandoned Extensions Removal

MB-C11 Composer Dependency Hygiene

MB-C12 Third-party Config Security

Rule Catalog (81 Rules)

MB-C01 File & Folder Permissions (5 rules)

MB-R001 — No chmod 777 (High, A05)

Magento project files and directories must never be world-writable using chmod 777. Overly permissive permissions allow attackers or rogue processes to modify code and configuration. Always apply least-privilege settings, restricting write access to only the system user that runs Magento.

MB-R002 — Secure env.php permissions (High, A05)

The app/etc/env.php file contains sensitive credentials, including database and encryption keys. It should be restricted to mode 640 or stricter, owned by the correct application user

and group. Insecure permissions may expose secrets to unauthorized local users or processes.

MB-R003 — Webroot hygiene (High, A05)

The webroot (pub/) must be free of leftover developer artifacts such as .git, .env, or backup files. These files often reveal credentials, configuration details, or source code. Attackers frequently probe for them as an easy first step to compromise.

MB-R004 — Restrict code dirs not writable (High, A05)

Core application directories like app/, vendor/, and lib/ must be read-only for the web server user. If writable, an attacker exploiting Magento could plant malicious PHP code directly into dependencies. Enforcing read-only integrity helps prevent supply-chain style compromises.

MB-R005 — No directory listing (Medium, A05)

Public web directories should not allow automatic directory indexing. Directory listing can expose file names, configuration fragments, and sensitive assets. Ensure the web server (Nginx/Apache) is configured to return 403/404 for requests without index files to reduce information leakage.

MB-C02 Admin Hardening (6 rules)

MB-R006 — Non-default admin path (High, A07)

The default /admin path is predictable and targeted by bots and automated scanners. Configuring a custom backend route significantly reduces attack surface. It forces attackers to guess the admin location, improving security against brute-force and credential stuffing attempts.

MB-R007 — Admin 2FA enabled (Critical, A07)

Two-Factor Authentication (2FA) is required to secure administrative logins against stolen or weak credentials. Enabling Magento_TwoFactorAuth enforces a second verification step, making it far harder for attackers to hijack accounts even if passwords are compromised.

MB-R008 — Strong password policy (High, A07)

Administrative accounts must use passwords with minimum length and complexity requirements. Enforcing rules for length, mixed characters, and rotation reduces the risk of brute-force attacks. Weak policies leave accounts vulnerable to dictionary or credential stuffing techniques.

MB-R009 — Session timeout ≤ 900s (Medium, A07)

Idle admin sessions should expire within 15 minutes to limit the window of abuse. Long session lifetimes make it easier for hijacked cookies or unattended terminals to be exploited. A strict timeout reduces exposure in case of compromise.

MB-R010 — Limit admin exposure (Medium, A07)

The admin route should not be publicly discoverable or exposed unnecessarily. Blocking default redirects, disabling /admin fallbacks, and restricting access with firewalls or IP allow-lists reduce discovery by attackers and automated scans.

MB-R011 — Login rate-limit (Medium, A07)

Implement rate-limiting or CAPTCHA to prevent unlimited login attempts on the admin panel. Brute-force and credential stuffing are common attack vectors. Throttling login attempts helps slow automated attacks and improves defense-in-depth when combined with strong credentials.

MB-C03 Secure Coding Practices (14 rules)

MB-R012 — No raw SQL queries (Critical, A03)

Directly concatenating variables into raw SQL queries introduces SQL injection risks. Magento provides database abstraction layers and bound parameters that must be used. Detecting and avoiding raw SQL ensures input is sanitized, protecting against data theft or manipulation.

MB-R013 — Template output escaping (High, A03)

Values printed in PHTML templates must be properly escaped using $block->escapeHtml(), escapeHtmlAttr(), or similar functions. Without output escaping, user-controlled data may trigger cross-site scripting (XSS). Escaping prevents malicious scripts from executing in the browser context.

MB-R014 — Avoid superglobals (High, A05)

Using PHP superglobals like $_GET or $_POST directly bypasses Magento's filtering and validation mechanisms. Developers should rely on Magento's request API or input filters. Superglobals increase the attack surface for injection and cross-site scripting if not sanitized properly.

MB-R015 — CSRF protection (High, A01/A05)

All form submissions and POST requests must include Magento's built-in form key validation. Without Cross-Site Request Forgery (CSRF) tokens, attackers can trick logged-in users into performing unintended actions. Enforcing CSRF checks protects critical workflows like checkout or admin actions.

MB-R016 — SSRF safeguards (High, A10)

External HTTP requests must restrict target hosts and validate URLs. Without safeguards, attackers can abuse Server-Side Request Forgery (SSRF) to reach internal services or metadata endpoints. Implement allow-lists, strict protocols, and timeouts to reduce this risk.

MB-R017 — Deserialization safety (High, A08)

PHP's unserialize() can lead to arbitrary code execution if user-controlled input is deserialized. Use JSON or Magento's safe serializers instead. Any unavoidable use of unserialize must enforce allowed class whitelists to prevent gadget chains and object injection.

MB-R018 — Command injection guards (Critical, A03)

Dangerous functions like exec(), shell_exec(), or system() must never execute user-controlled input. Attackers may inject arbitrary shell commands. If shell usage is unavoidable, inputs must be sanitized against strict allow-lists. Safer alternatives are preferred wherever possible.

MB-R019 — No unsafe eval/dynamic code (Critical, A03/A08)

Functions like eval(), assert(), or create_function() should not be used with dynamic input. They allow arbitrary code execution and are a common vector in malware. Modern PHP and Magento provide safer alternatives to handle dynamic behavior securely.

MB-R020 — Path traversal protections (High, A01/A05)

File access functions (fopen, file_get_contents, etc.) must validate and normalize paths. Without protections, attackers can perform directory traversal (../) to read or overwrite sensitive files. Always use realpath() checks and enforce base directory allow-lists.

MB-R021 — Secure file uploads (High, A08)

Upload handlers must validate MIME types, enforce extension allow-lists, limit file size, and store files outside webroot. Attackers often upload PHP shells disguised as images. Files should be re-encoded where possible and named randomly to avoid collisions.

MB-R022 — JS-context escaping (High, A03)

Variables injected into <script> blocks must be escaped with escapeJs() or JSON-encoded safely. Failing to escape JavaScript context enables reflected or stored XSS. This rule ensures Magento templates protect users against malicious scripts in dynamic content.

MB-R023 — Cryptographically secure RNG (High, A02)

Security tokens, nonces, and session identifiers must use CSPRNG functions (random_bytes, random_int). Insecure RNG (rand, mt_rand) can be predicted by attackers, enabling replay or token guessing. Cryptographically secure generators are mandatory for sensitive operations.

MB-R024 — Sensitive data not logged (High, A09)

Logs must not contain sensitive information such as passwords, API tokens, or card data. Attackers with log access can exploit leaked data. Magento logging should use sanitization filters and redact sensitive values before writing to files or monitoring systems.

MB-R025 — Use Magento APIs for crypto/session (Medium, A02/A07)

Magento provides APIs for encryption, hashing, and session handling. Using PHP's raw openssl_encrypt or custom session logic introduces misconfigurations and weak defaults. Always rely on platform-provided APIs to ensure consistency, upgrades, and strong security defaults.

MB-C04 HTTPS & TLS Enforcement (5 rules)

MB-R026 — Force HTTPS (High, A02)

All storefront and admin endpoints must use HTTPS to protect data in transit against eavesdropping and tampering. Enforce secure base URLs, redirect HTTP to HTTPS, and ensure cookies are only transmitted over TLS to maintain confidentiality and integrity.

MB-R027 — HSTS header enabled (Medium, A02)

HTTP Strict Transport Security (HSTS) instructs browsers to always use HTTPS, preventing downgrade and SSL-strip attacks. Configure a strong max-age, include subdomains where appropriate, and preloading if eligible to ensure persistent TLS enforcement for returning users.

MB-R028 — TLS ≥ 1.2 only (High, A02)

Disable legacy protocols (SSLv3, TLS 1.0/1.1) and weak ciphers to reduce exposure to known cryptographic flaws. Require TLS 1.2+ with modern ciphers, forward secrecy, and robust key exchange settings to harden transport encryption across all entry points.

MB-R029 — No mixed content (Medium, A02)

Pages served over HTTPS must not load HTTP assets (scripts, styles, images). Mixed content undermines TLS guarantees and enables injection. Audit templates and CDN references, update asset URLs to HTTPS, and implement Content Security Policy to prevent accidental regressions.

MB-R030 — Secure cookies flags (High, A02/A07)

Session and authentication cookies must set Secure, HttpOnly, and appropriate SameSite attributes. These flags reduce leakage over plaintext channels and mitigate XSS-based theft or CSRF abuse. Validate headers and Magento configuration to consistently apply cookie protections.

MB-C05 Production Mode & Deployment Hygiene (6 rules)

MB-R031 — Magento in production mode (High, A05)

Production mode disables developer-specific features and optimizes performance. Running Magento in developer mode on production systems exposes debug output and slows execution. Always verify that production deployments explicitly set the correct mode to reduce risk and improve stability.

MB-R032 — No Xdebug on prod (Medium, A05)

Debugging extensions like Xdebug should never run on production servers. They introduce significant performance overhead and may expose sensitive data via traces or profiling endpoints. Confirm that php.ini does not load debugging modules in live environments.

MB-R033 — Display errors off (High, A05)

PHP error reporting must be logged, not displayed to users. Displaying stack traces or warnings reveals internal code paths and sensitive details. Ensure display_errors=Off in php.ini and configure Magento to mask detailed error messages in production.

MB-R034 — Compiled DI enabled (Medium, A05)

Magento's Dependency Injection (DI) compilation generates optimized factories for production. Running without DI compilation forces runtime lookups, increasing attack surface and latency. Ensure setup:di:compile is executed during deployment, and the generated code is committed or built properly.

MB-R035 — Static assets deployed (Medium, A05)

Static view files (CSS, JS, images) should be deployed in advance using setup:static-content:deploy. Without pre-deployment, Magento may generate assets on the fly, leaking error details or causing downtime. Pre-built assets improve performance and reduce misconfiguration exposure.

MB-R036 — No dev configs on prod (High, A05/A08)

Development configurations such as sandbox API keys, test SMTP servers, or verbose logging must not remain in production. They often bypass security controls and can leak sensitive data. Audit environment configs to ensure only production values are applied on live systems.

MB-C06 Cache & Indexing Health (5 rules)

MB-R037 — FPC enabled (High, A05)

Full Page Cache (FPC) significantly improves performance and reduces backend load. Disabling or misconfiguring FPC forces every request to hit PHP and the database, slowing responses and exposing bottlenecks. Always ensure FPC is enabled and serving cached content on production.

MB-R038 — Redis/Varnish configured (Medium, A05)

Magento supports advanced cache backends like Redis or Varnish. Without proper configuration, stores rely on file-based cache, which scales poorly and risks corruption. Using Redis or Varnish ensures faster cache invalidation, distributed caching, and higher resilience under heavy traffic.

MB-R039 — Indexers READY (Medium, A05)

Magento relies on indexers for search, pricing, and catalog performance. If indexers remain in "REINDEX REQUIRED" state, queries degrade severely, and features may fail. Regularly monitor and reindex to ensure indexers are healthy and data is up-to-date in production.

MB-R040 — Hardened session storage (High, A05)

Session data should never be stored in insecure file paths or world-writable locations. Redis with authentication or a database-backed session handler provides better isolation. Weak session storage exposes the risk of hijacking or leakage of sensitive authentication tokens.

MB-R041 — No dev cache backends (Medium, A05)

File-based caching is acceptable for local development but not suitable for clustered or production setups. Using file cache on shared hosts causes race conditions and stale content. Ensure production systems are configured to use Redis or Varnish backends exclusively.

MB-C07 Logging & Monitoring (4 rules)

MB-R042 — Protect application logs (High, A09)

Application logs must not be publicly accessible from the web. Attackers often scan for exposed var/log or server log files to gather sensitive information. Configure web server rules and file permissions to prevent direct download or browsing of log files.

MB-R043 — Log rotation (Medium, A09)

Large, unrotated log files can fill disks and cause denial of service. Rotation ensures logs are archived, compressed, and safely stored. Configure logrotate or equivalent tools to prevent excessive growth and maintain the availability of logging infrastructure.

MB-R044 — Safe exception handling (High, A09/A05)

Exceptions must not expose stack traces or internal details to end users. Instead, return generic error pages while logging technical details securely. Exposed traces reveal file paths, libraries, and vulnerabilities that aid attackers in reconnaissance.

MB-R045 — PII sanitized in logs (High, A09/A02)

Logs must avoid storing personally identifiable information (PII) such as names, emails, or payment data. If logging user data is unavoidable, values must be masked or hashed. Sanitizing logs reduces privacy risk and regulatory compliance issues.

MB-C08 Cron Job Reliability (3 rules)

MB-R046 — Crontab entries present (High, A05)

Magento relies heavily on cron jobs to run indexing, email sending, cache cleanup, and scheduled tasks. Missing crontab entries break essential background processes, leading to instability and delayed order processing. Verify that all required cron jobs are configured and active.

MB-R047 — Cron heartbeat healthy (Medium, A05)

A cron heartbeat confirms that scheduled jobs are running on time. If the heartbeat is stale or missing, tasks may be failing silently. Monitoring the cron schedule and alerting on failures ensures the continuous operation of business-critical background jobs.

MB-R048 — Cron backlog threshold (Medium, A05)

Excessive pending cron jobs indicate performance or configuration issues. A growing backlog delays time-sensitive tasks like order invoicing or email notifications. Regularly audit the queue length and alert if thresholds are exceeded to maintain healthy job execution.

MB-C09 Extension Vulnerability Management (12 rules)

MB-R049 — CVE match via OSV (Critical, A06)

Installed extensions must be scanned against OSV.dev and other vulnerability databases for known CVEs. Unpatched vulnerabilities in third-party modules are one of the top entry points for attackers. Regularly check advisories and flag modules with unresolved security issues.

MB-R050 — Core module advisories flagged (Critical, A06)

Vulnerabilities in official Magento core modules or Adobe-maintained packages must be tracked and patched immediately. Exploits against core components are widely weaponized, making this category highly critical. Automated scanning should alert whenever core advisories apply to the current version.

MB-R051 — Suggest fixed versions (High, A06)

When vulnerabilities are detected, stores should identify the patched version that resolves the issue. Providing recommended fixed versions helps developers upgrade quickly and avoid insecure builds. This minimizes exposure windows by guiding remediation paths clearly.

MB-R052 — High-risk surface modules flagged (High, A06)

Modules handling payment, authentication, or customer data represent high-risk attack surfaces. Even without known CVEs, they should be highlighted for extra scrutiny. Prioritizing code reviews and updates for these modules reduces overall compromise risk.

MB-R053 — Temporary mitigations documented (Medium, A06/A08)

If patches are not yet available, temporary mitigations (e.g., disabling a feature, adding firewall rules) should be documented and applied. This ensures critical vulnerabilities are managed proactively, reducing risk while awaiting vendor-supplied fixes.

MB-R054 — Known exploited vulns prioritized (High, A06)

Vulnerabilities listed in CISA KEV or similar databases indicate active exploitation in the wild. Such extensions must be patched or disabled immediately. Prioritizing known exploited vulnerabilities is essential to reduce the likelihood of compromise by automated campaigns.

MB-R055 — Transitive dependency CVEs flagged (Critical, A06)

Extensions often rely on third-party PHP libraries that may contain vulnerabilities. Transitive dependencies must also be scanned for CVEs, not only top-level modules. Attackers frequently exploit weaknesses in bundled libraries overlooked by store operators.

MB-R056 — Constraints blocking fixes flagged (High, A06)

Composer version constraints can prevent applying patched releases. Overly strict constraints increase exposure windows by locking stores to insecure versions. Detecting and warning about such conflicts helps maintainers adjust constraints and apply timely updates.

MB-R057 — Yanked/withdrawn versions flagged (High, A06)

Composer packages marked as yanked or withdrawn should be flagged immediately. Yanked releases often indicate serious defects or vulnerabilities. Continuing to run these versions exposes stores to unpatched risks and should be remediated quickly.

MB-R058 — Outdated Marketplace extensions flagged (High, A06)

Marketplace modules without updates for long periods pose potential risks due to unpatched security issues. Highlighting outdated extensions encourages maintainers to review or replace them proactively. Staying current reduces the chance of silent exposure to known flaws.

MB-R059 — Advisory age/patch latency reported (Medium, A06)

Track how long advisories have been available compared to patch adoption. Long patch latency indicates operational risk. Reporting advisory age helps teams measure responsiveness and prioritize overdue updates across extensions.

MB-R060 — Extension with no vendor support flagged (Medium, A06)

Modules from vendors who no longer provide updates or have abandoned maintenance represent a critical long-term risk. Unsupported extensions should be flagged for replacement. Relying on unmaintained code increases exposure to unresolved vulnerabilities indefinitely.

MB-C10 Abandoned Extensions Removal (4 rules)

MB-R061 — Abandoned on Packagist (High, A06)

Extensions marked as “abandoned” on Packagist or Composer metadata should be immediately flagged. Abandoned modules no longer receive updates or security patches, leaving them permanently vulnerable. Replace such modules with maintained alternatives to reduce long-term security exposure.

MB-R062 — No release in >24 months (Medium, A06)

Extensions with no new release or update in over two years are strong indicators of abandonment. Old versions accumulate unpatched vulnerabilities over time. Monitoring release activity ensures outdated, stagnant modules are reviewed for replacement or removal.

MB-R063 — Archived repositories (Medium, A06)

GitHub or GitLab repositories marked “archived” signal that no further development will occur. Depending on the archived code poses a serious risk as future vulnerabilities will

remain unaddressed. Such modules should be deprecated in production environments and migrated away from.

MB-R064 — Risky forks replacing originals (Medium, A06/A08)

Forked modules maintained by unknown or unverified sources may lack security review. Depending on these forks introduces supply chain risk if malicious code is inserted. Always validate the reputation of maintainers and prefer official, actively supported repositories.

MB-C11 Composer Dependency Hygiene (7 rules)

MB-R065 — No wildcard constraints (High, A06)

Composer version constraints like * or dev-master allow uncontrolled upgrades, introducing unexpected vulnerabilities. Strict, semantic versioning ensures dependencies are predictable and security patches are applied consistently. Avoiding wildcards prevents accidental installation of insecure or unstable releases.

MB-R066 — No dev branches (High, A06)

Depending on the development branches (dev-branch) pulls unstable, unreviewed code into production. Such code may contain incomplete features or insecure implementations. Production environments must rely only on tagged, stable releases that receive security patches and long-term maintenance.

MB-R067 — Stable by default (High, A06)

Composer must be configured with prefer-stable=true to ensure stable packages are selected over unstable ones. Without this, unstable libraries may be chosen during dependency resolution. Enforcing stable preference reduces exposure to untested, insecure builds.

MB-R068 — Composer audit clean (High, A06)

Running composer audit checks dependencies against known CVE databases. Stores should fail builds if unresolved vulnerabilities are detected. Maintaining a clean audit report ensures critical vulnerabilities are addressed proactively and reduces exposure to compromised libraries.

MB-R069 — Direct deps up-to-date (Medium, A06)

Outdated direct dependencies accumulate unfixed vulnerabilities. Regularly updating core libraries and Magento packages ensures access to the latest security patches. Monitoring update frequency helps maintainers avoid lagging behind on critical dependency fixes.

MB-R070 — Lockfile integrity (Medium, A08)

The composer.lock file should be committed to version control and remain consistent with composer.json. Missing or outdated lockfiles cause uncontrolled dependency drift, leading to insecure or inconsistent builds. Validating lockfile integrity ensures deterministic, secure deployments.

MB-R071 — Disallow abandoned PHP libs (Medium, A06)

PHP libraries marked abandoned by maintainers should not remain in production. Relying on unsupported libraries increases long-term risk as vulnerabilities will not be patched. Audit dependency trees regularly and replace abandoned libraries with supported alternatives.

MB-C12 Third-party Config Security (10 rules)

MB-R072 — No secrets in VCS (Critical, A08/A02)

API keys, tokens, and credentials must never be committed to source control or copied into sample configs. Secret exposure enables account takeover and data exfiltration. Use environment variables, secret managers, and pre-commit scanners to prevent accidental leaks across repositories.

MB-R073 — HTTPS-only endpoints (High, A02)

All third-party integrations—including payment, shipping, email, and analytics—must use HTTPS with valid certificates. Plain HTTP exposes tokens and PII to interception or tampering. Validate endpoint schemes in configuration, enforce TLS at proxies, and block insecure URLs during deployment.

MB-R074 — Debug/verbose disabled in prod (Medium, A05)

Third-party modules often include debug or verbose logging modes that reveal requests, headers, or credentials. These settings must be disabled in production scopes. Audit configuration per environment and verify no debug endpoints or inspector UIs are reachable on public networks.

MB-R075 — Webhook signature validation (High, A08/A07)

Inbound webhooks must be authenticated using HMAC or signed tokens to prevent spoofing. Without validation, attackers can forge events, trigger refunds, or alter orders. Configure shared secrets, rotate periodically, and reject requests lacking correct signatures or timestamps.

MB-R076 — Outbound allow-list enforced (High, A10/A05)

Egress traffic for integrations should be restricted to an allow-listed set of hostnames or IP ranges. Unrestricted outbound requests increase SSRF blast radius and data exfiltration risk. Enforce DNS pinning, firewall rules, or HTTP client allow-lists to constrain destinations.

MB-R077 — PII minimization in configs (Medium, A02/A08)

Configuration values must avoid storing unnecessary personal data such as full addresses, phone numbers, or payment metadata. Prefer transient tokens and references. Minimizing PII reduces breach impact, simplifies compliance, and limits incidental exposure through logs, backups, or misconfigurations.

MB-R078 — Payment gateway configs use strong TLS ciphers (High, A02)

Payment integrations must negotiate modern TLS versions and recommended cipher suites. Weak protocols or ciphers invite downgrade and interception attacks. Validate with automated TLS scanners, monitor gateway announcements, and enforce strict security profiles at load balancers and edge proxies.

MB-R079 — API keys stored in env.php, not DB/plaintext (High, A08)

Secrets should live in app/etc/env.php or a secret manager, never in database rows or plaintext admin settings. Database-stored keys tend to proliferate across environments and backups. Centralized storage simplifies rotation and restricts access via filesystem permissions.

MB-R080 — Third-party logging sanitized (Medium, A09)

Integration logs should redact tokens, session IDs, customer identifiers, and request bodies containing PII. Many SDKs log full payloads by default. Configure allow-lists for safe fields, enable masking filters, and periodically review log samples for accidental sensitive data exposure.

MB-R081 — Cloud/SaaS integrations restricted by ACL (Medium, A05)

Accounts used for SaaS connectors must follow least privilege. Assign scoped API roles, restrict IP ranges, and separate production from sandbox tenants. Over-permissioned API users amplify blast radius when credentials leak, enabling destructive actions beyond the integration’s intended purpose.

Chapter 4. Implementation Guidance

The Magebean CLI (magebean-cli) is the primary tool to operationalize this baseline.

It allows automated scanning of Magento 2 projects, validation of Controls and Rules, and generation of actionable reports.

Installation

Magebean CLI is distributed as a self-contained `.phar` package.
It requires **PHP 8.1+** and can be downloaded from the official Magebean distribution site.

```
wget https://files.magebean.com/magebean-cli.phar -O magebean.phar

chmod +x magebean.phar
```

Basic Usage

Run a scan against a Magento 2 installation:

```
./magebean.phar scan --path=/var/www/magento
```

`--path` specifies the root directory of the Magento 2 project.

The tool will automatically apply all 12 Controls and 81 Rules.

Output Formats

Magebean CLI supports multiple report formats:

CLI (default): Human-readable summary directly in the console.

JSON: Machine-readable output for integration.

HTML/PDF: Styled reports for auditors and stakeholders.

```
./magebean scan --path=/var/www/magento --format=html
                --output=report.html
```

CI/CD Integration

Magebean CLI is designed to run inside CI/CD pipelines.

Exit codes are mapped to severity levels, allowing builds to fail when critical issues are found.

Example GitHub Actions workflow:

jobs:

```
 security-audit:

    runs-on: ubuntu-latest

    steps:

      - uses: actions/checkout@v3

      - name: Run Magebean Audit

        run: /path-to-file/magebean.phar scan --path=. --format=json
                      --output=report.json
```

Operational Recommendations

Audit Frequency: Run baseline audits before every release and weekly on production environments.

Remediation Workflow: Prioritize fixes for Critical and High severity findings.

Version Control: Store reports in CI artifacts for traceability and compliance.

Offline Mode: Magebean CLI runs fully offline to preserve privacy; CVE databases can be updated manually when needed.

Chapter 5. Severity & Risk Rating

Each Rule in this baseline is assigned a severity level based on potential impact and likelihood of exploitation.

This chapter explains the rating system to help prioritize remediation.

Severity Levels

Critical: Exploitation can lead to full system compromise, data breach, or payment fraud. Must be remediated immediately.

High: Exploitation can cause significant security or operational risk, such as privilege escalation or data leakage. Should be prioritized for remediation.

Medium: Exploitation has a limited impact or requires additional conditions. Important to fix, but may be scheduled.

Low: Minor security hygiene issues or best practices. Fix as part of normal maintenance.

Risk Mapping

OWASP Top 10: Each Rule is mapped to a relevant OWASP category (e.g., A01: Broken Access Control, A06: Vulnerable Components).

CVSS Alignment: Critical/High ratings generally align with CVSS base scores ≥ 7.0, while Medium/Low correspond to lower CVSS ranges.

Usage in Reporting

Magebean CLI can generate reports grouped by severity.

Exit codes may be configured for CI/CD pipelines (e.g., fail build if Critical issues are detected).

Severity levels guide triage and remediation priorities for development and operations teams.

References

OWASP Top 10 (2021) — https://owasp.org/Top10/

OWASP ASVS 4.0 — https://owasp.org/ASVS/

Magento 2 Documentation — https://developer.adobe.com/commerce/docs/

OSV.dev Vulnerability Database — https://osv.dev/

Appendix C. Glossary of Terms

Audit

A structured review process to evaluate a Magento system against Controls and Rules. In Magebean CLI, an audit is executed via automated scans.

Baseline

The minimum recommended standard consists of 12 Controls and 81 Rules. It provides a reference point for measuring Magento 2 security posture.

Control

A high-level category of checks that represents a key security or compliance area (e.g., Admin Hardening, HTTPS Enforcement).

Rule

A specific, measurable requirement that enforces a Control. Rules are automatable and serve as the unit of compliance in an audit.

Scan

The technical execution of automated checks against all Rules, producing pass/fail results and reports.

OWASP Top 10

A globally recognized standard for the top ten most critical web application security risks (2021 edition is referenced in this baseline).

CVE (Common Vulnerabilities and Exposures)

A standardized identifier for publicly known security vulnerabilities. Used in Magebean to flag risky Magento extensions or dependencies.

Composer/composer.lock

PHP's dependency manager. The composer.lock file ensures deterministic dependency versions. Weak constraints or outdated lockfiles can lead to insecure builds.

Extension (Magento Module)

A third-party or custom Magento add-on. Extensions increase functionality but may also introduce vulnerabilities if unmaintained or insecure.

Dependency / Transitive Dependency

A software package required by Magento or its extensions. Transitive dependencies are nested libraries pulled indirectly, often overlooked but exploitable.

Misconfiguration

An insecure or unintended system setting (e.g., display_errors=On, HTTP enabled instead of HTTPS). A common source of compromise.

Hardening

Strengthening security by reducing attack surface and enforcing best practices. Examples include Admin Hardening and TLS Hardening.

2FA (Two-Factor Authentication)

An additional authentication step beyond passwords, required to secure Magento admin logins.

CSRF (Cross-Site Request Forgery)

An attack where a user's authenticated session is abused to perform unwanted actions. Magento mitigates CSRF with form keys.

XSS (Cross-Site Scripting)

An injection attack where malicious scripts execute in the browser. Prevented by proper output escaping in templates and JavaScript contexts.

SQL Injection (SQLi)

An injection flaw where unsanitized input alters database queries. Prevented by Magento's query abstraction and bound parameters.

SSRF (Server-Side Request Forgery)

An attack where the server is tricked into making unintended HTTP requests. Prevented by allow-listing outbound destinations.

CSPRNG (Cryptographically Secure Random Number Generator)

A random generator suitable for security-sensitive tokens (e.g., random_bytes). Prevents predictability in sessions or nonces.

PII (Personally Identifiable Information)

Any data that can identify an individual (e.g., name, email, address). Must be protected and never logged in plaintext.

HSTS (HTTP Strict Transport Security)

A security header forcing browsers to use HTTPS, preventing downgrade or SSL-stripping attacks.

TLS (Transport Layer Security)

The cryptographic protocol that secures data in transit. Magebean requires TLS 1.2 or higher with strong ciphers.

FPC (Full Page Cache)

Magento's built-in caching mechanism. Ensures better performance and reduces backend exposure.

Indexers

Magento background processes that pre-compute data (e.g., search, catalog, pricing). Must remain healthy to avoid performance degradation.

Note: Magebean Security Baseline is an original framework authored by Son Cao.

It is aligned with OWASP standards but tailored specifically for Magento 2.