

shopsys

github.com/shopsys/shopsys

Scanned 24 September 2026 at 08:24 · Report generated 29 October 2026 · PHP · 350 stars

OVERALL ASSESSMENT

Critical

Combined score 487 · 261 findings after filtering

Summary

This repository has a critical vulnerability score of 485.0, which means there are serious insecure coding patterns that an attacker could exploit to break into your system. While the code does not appear to be intentionally malicious, it contains many instances of unsafe commands like `child_process` and shell execution that could lead to command injection attacks if used incorrectly. You should avoid using these dangerous functions unless you have a very specific reason and implement strict input validation to prevent them from being abused. Additionally, there are several medium-priority issues related to Regular Expression Denial-of-Service (ReDoS) risks in your codebase that could slow down or crash applications if exploited. Overall, the repository is not safe for production use without a thorough security review and refactoring of these specific patterns before deployment.

Written by a language model running locally on the scanning machine, from the findings in this report. The risk levels above and below were decided by the scoring model, not by this summary.

Risk, by type

MalDet scores two different questions separately, because they call for different responses: code can be badly written without being hostile, and hostile code is not always badly written.

VULNERABILITY

Critical

 (485)

Insecure coding an attacker could exploit. Usually mistakes, and usually fixable in place.

MALICIOUS PATTERN

Low

 (3)

Signs the code may be trying to do harm: hidden payloads, credential access, persistence, exfiltration.

What each detector found

DETECTOR	LOOKS FOR	HIGH	MEDIUM	LOW	TOTAL
semgrep	Insecure coding patterns across many languages	19	229	0	248
yara	Signatures of malicious behaviour in source files	1	0	7	8
guarddog	Malicious package and install-time behaviour	0	0	3	3
dep_checker	Vulnerable, unpinned or typosquatted dependencies	0	0	2	2

214 of 261 findings were rated more likely genuine than noise by the classifier.

Evidence

Identical findings are grouped, so a single rule firing many times takes one row. Ordered worst first. "Seen" is how many times that finding occurred.

SEVERITY	DETECTOR	FINDING	SEEN	EXAMPLE LOCATION
HIGH	semgrep	Detected calls to <code>child_process</code> from a function argument <code>`answers`</code> . This could lead to a command injection if the input is user controllable. Try to avoid calls to <code>child_process</code> , and if it is needed ensure user input is correctly sanitized or sandboxed.	1	<code>/project-base/storefront/cypress/runSelectedTests.js:41</code>
HIGH	semgrep	Found <code>'\$SPAWN'</code> with <code>'{shell: true}'</code> . This is dangerous because this call will spawn the command using a shell process. Doing so propagates current shell settings and variables, which makes it much easier for a malicious actor to execute commands. Use <code>'{shell: false}'</code> instead.	1	<code>/project-base/storefront/cypress/runSelectedTests.js:41</code>
HIGH	semgrep	This workflow uses <code>`secrets: inherit`</code> to pass all of the calling workflow's secrets to a reusable workflow. This violates the principle of least privilege because the called workflow receives access to every secret in the repository, not just the ones it needs. If the called workflow is compromised	2	<code>/.github/workflows/ai-acceptance-criteria-ready-for-review.yaml:34</code>

SEVERITY	DETECTOR	FINDING	SEEN	EXAMPLE LOCATION
HIGH	semgrep	Using variable interpolation `\${{...}}` with `github` context data in a `run:` step could allow an attacker to inject their own code into the runner. This would allow them to steal secrets and code. `github` context data can have arbitrary user input and should be treated as untrusted. Instead, use	9	/.github/workflows/monorepo-force-split-branch.yaml:31
HIGH	semgrep	Executing non-constant commands. This can lead to command injection.	3	/packages/framework/src/Command/DatabaseDumpCommand.php:62
HIGH	semgrep	By not specifying a USER, a program in the container may run as 'root'. This is a security hazard. If an attacker can control a process running as root, they may have control over the container. Ensure that the last USER in a Dockerfile is a USER other than 'root'.	3	/docker/mkdocs/Dockerfile:11
HIGH	yara	YARA rule matched: detect_persistence_mechanism — Modifies a startup/scheduling location - legitimate application code essentially never does this	1	/packages/php-image/Dockerfile
MEDIUM	semgrep	RegExp() called with a `route` function argument, this might allow an attacker to cause a Regular Expression Denial-of-Service (ReDoS) within your application as RegExp blocks the main thread. For this reason, it is recommended to use hardcoded regexes instead. If your regex is run on user-controlle	1	/project-base/storefront/utils/skeleton/getSkeletonTypeFromRouteUrl.ts:65
MEDIUM	semgrep	RegExp() called with a `graphqlEndpoint` function argument, this might allow an attacker to cause a Regular Expression Denial-of-Service (ReDoS) within your application as RegExp blocks the main thread. For this reason, it is recommended to use hardcoded regexes instead. If your regex is run on user	1	/project-base/storefront/utils/sentry/tracePropagationTargets.ts:30

SEVERITY	DETECTOR	FINDING	SEEN	EXAMPLE LOCATION
MEDIUM	semgrep	RegExp() called with a `directiveRegexps` function argument, this might allow an attacker to cause a Regular Expression Denial-of-Service (ReDoS) within your application as RegExp blocks the main thread. For this reason, it is recommended to use hardcoded regexes instead. If your regex is run on use	1	/project-base/storefront/urql/fetcher.ts:108
MEDIUM	semgrep	RegExp() called with a `parameterName` function argument, this might allow an attacker to cause a Regular Expression Denial-of-Service (ReDoS) within your application as RegExp blocks the main thread. For this reason, it is recommended to use hardcoded regexes instead. If your regex is run on user-c	1	/project-base/app/assets/js/admin/grapesjs/plugins/web/video.js:96
MEDIUM	semgrep	RegExp() called with a `internalEndpoint` function argument, this might allow an attacker to cause a Regular Expression Denial-of-Service (ReDoS) within your application as RegExp blocks the main thread. For this reason, it is recommended to use hardcoded regexes instead. If your regex is run on use	1	/project-base/storefront/utils/sentry/tracePropagationTargets.ts:44
MEDIUM	semgrep	`\$queryBuilder` Detected string concatenation with a non-literal variable in a Doctrine QueryBuilder method. This could lead to SQL injection if the variable is user-controlled and not properly sanitized. In order to prevent SQL injection, use parameterized queries or prepared statements instead.	2	/packages/framework/src/Model/Product/Parameter/ParameterGridFactory.php:100
MEDIUM	semgrep	`\$this->em->createQueryBuilder()->select(\$tmpTableAlias . '.id')` Detected string concatenation with a non-literal variable in a Doctrine QueryBuilder method. This could lead to SQL injection if the variable is user-controlled and not properly sanitized. In order to prevent SQL injection, use parame	1	/packages/framework/src/Component/Router/FriendlyUrl/FriendlyUrlRepository.php:221

SEVERITY	DETECTOR	FINDING	SEEN	EXAMPLE LOCATION
MEDIUM	semgrep	`\$this->em->createQueryBuilder()->select(\$tmpTableAlias . '.id') ->from(\$entity, \$tmpTableAlias)` Detected string concatenation with a non-literal variable in a Doctrine QueryBuilder method. This could lead to SQL injection if the variable is user-controlled and not properly	1	/packages/framework/src/Component/Router/FriendlyUrl/FriendlyUrlRepository.php:222
MEDIUM	semgrep	`\$queryBuilder ->addSelect('pt_' . \$locale)` Detected string concatenation with a non-literal variable in a Doctrine QueryBuilder method. This could lead to SQL injection if the variable is user-controlled and not properly sanitized. In order to prevent SQL injection, use parameter	2	/packages/framework/src/Model/Product/Parameter/ParameterGridFactory.php:105
MEDIUM	semgrep	`\$this->em->createQueryBuilder()` Detected string concatenation with a non-literal variable in a Doctrine QueryBuilder method. This could lead to SQL injection if the variable is user-controlled and not properly sanitized. In order to prevent SQL injection, use parameterized queries or prepared stat	1	/packages/framework/src/Component/Router/FriendlyUrl/FriendlyUrlRepository.php:220
MEDIUM	semgrep	`pattern.replace` method will only replace the first occurrence when used with a string argument ('*'). If this method is used for escaping of dangerous data then there is a possibility for a bypass. Try to use sanitization library instead or use a Regex with a global flag.	1	/project-base/storefront/cypress/cypress.config.ts:161
MEDIUM	semgrep	This location block contains a 'proxy_pass' directive but does not contain the 'internal' directive. The 'internal' directive restricts access to this location to internal requests. Without 'internal', an attacker could use your server for server-side request forgeries (SSRF). Include the 'internal'	1	/project-base/docker/nginx/nginx.conf:38

SEVERITY	DETECTOR	FINDING	SEEN	EXAMPLE LOCATION
MEDIUM	semgrep	This Dependabot configuration does not set a cooldown period. Newly published packages can be malicious or unstable. Add a `cooldown` block with `default-days: 7` to each `package-ecosystem` entry under `updates` to wait 7 days before proposing updates to newly published package versions. Reference:	2	<code>/.github/dependabot.yml:8</code>
MEDIUM	semgrep	CSRF protection is disabled for this configuration. This is a security risk. Make sure that it is safe or consider setting `csrf_protection` property to `true`.	1	<code>/packages/framework/src/Form/Admin/QuickSearch/QuickSearchFormType.php:30</code>
MEDIUM	semgrep	Conditions for Nginx H2C smuggling identified. H2C smuggling allows upgrading HTTP/1.1 connections to lesser-known HTTP/2 over cleartext (h2c) connections which can allow a bypass of reverse proxy access controls, and lead to long-lived, unrestricted HTTP traffic directly to back-end servers. To mit	1	<code>/project-base/docker/nginx/nginx.conf:174</code>
MEDIUM	semgrep	RegExp() called with a `pattern` function argument, this might allow an attacker to cause a Regular Expression Denial-of-Service (ReDoS) within your application as RegExP blocks the main thread. For this reason, it is recommended to use hardcoded regexes instead. If your regex is run on user-control	1	<code>/project-base/storefront/cypress/cypress.config.ts:161</code>
MEDIUM	semgrep	Detected string concatenation with a non-literal variable in a Doctrine DBAL query method. This could lead to SQL injection if the variable is user-controlled and not properly sanitized. In order to prevent SQL injection, use parameterized queries or prepared statements instead.	2	<code>/packages/mcp/src/Component/Database/Query/SqlExecutor.php:64</code>

SEVERITY	DETECTOR	FINDING	SEEN	EXAMPLE LOCATION
MEDIUM	semgrep	GitHub Actions step uses a mutable tag or branch reference. Tags and branch names can be silently repointed by the action owner, enabling supply-chain attacks — as seen in the trivy-action and kics-github-action compromises. Pin the reference to a full 40-character commit SHA instead, e.g. `uses: ac	177	/.github/workflows/docker-build.yaml:729
MEDIUM	semgrep	The `redirect()` method does not check its destination in any way. If you redirect to a URL provided by end-users, your application may be open to the unvalidated redirects security vulnerability. Consider using literal values or an allowlist to validate URLs.	11	/packages/administration/src/Controller/AbstractCrudController.php:330
MEDIUM	semgrep	Detected the use of eval(). eval() can be dangerous if used to evaluate dynamic content. If this content can be input from outside the program, this may be a code injection vulnerability. Ensure evaluated content is not definable by external sources.	1	/project-base/storefront/cypress/generateSnapshotsInfoTable.js:29
MEDIUM	semgrep	Detection of dangerouslySetInnerHTML from non-constant definition. This can inadvertently expose users to cross-site scripting (XSS) attacks if this comes from user-provided input. If you have to use dangerouslySetInnerHTML, consider using a sanitization library such as DOMPurify to sanitize your HT	13	/project-base/storefront/components/Blocks/StoreList/StoreExpandedInfo.tsx:33
MEDIUM	semgrep	Calling `unserialize()` with user input in the pattern can lead to arbitrary code execution. Consider using JSON or structured data approaches (e.g. Google Protocol Buffers).	1	/packages/framework/src/Model/Administrator/Security/AdministratorFrontSecurityFacade.php:77
MEDIUM	semgrep	Using user input when deleting files with `unlink()` is potentially dangerous. A malicious actor could use this to modify or access files they have no right to.	5	/packages/framework/src/Component/FileSystem/Flysystem/VolumeDriver.php:135

SEVERITY	DETECTOR	FINDING	SEEN	EXAMPLE LOCATION
LOW	guarddog	Detects reading of environment variables (credential access, often contains secrets) [threat-runtime-environment-read]	3	/project-base/storefront/buildPublicEnvConfig.ts:23
LOW	dep_checker	Loose version constraint: 'fs-extra^11.2.0' — attacker could push malicious update	1	package.json
LOW	dep_checker	Loose version constraint: 'globby^14.0.2' — attacker could push malicious update	1	package.json
LOW	yara	YARA rule matched: detect_downloader — Downloads a remote resource - common and low-signal alone; see detect_download_and_execute for the composite version	7	/docker/conf/docker-compose-mac.yml.dist

How to read this report

- **This is a static scan.** The code was read, never executed. Behaviour that only appears at runtime is outside what any of these detectors can see.
- **Findings are not all equally real.** A classifier rates each finding on how likely it is to be genuine rather than noise, and that rating weights the scores. Measured against a hand-reviewed sample it reached precision 0.52 and recall 0.62 — useful for ranking, not a verdict on any single finding.
- **A high score is not proof of malice.** Security tools, penetration-testing frameworks and malware research repositories contain the same patterns as real malware by design. The two risk types above exist to keep that distinction visible.
- **Absence of findings is not proof of safety.** A detector that is not installed, or a language none of them parse, produces silence rather than a warning.
- **This scan is a snapshot.** The repository may have changed since 24 September 2026.