

pbm

github.com/pinballmap/pbm

Scanned 24 September 2026 at 07:57 · Report generated 29 October 2026 · Ruby · 141 stars

OVERALL ASSESSMENT

Critical

Combined score 438 · 119 findings after filtering

Summary

This repository has two distinct security issues that require immediate attention before it can be used in production. The most critical finding is a high vulnerability risk score of 437.0, which indicates the code contains serious insecure coding patterns that an attacker could exploit to break into the system or steal data. While there are also signs of potential malicious intent with a low malicious-pattern score of 2.0, these appear to be standard security warnings rather than evidence of intentional harm. The primary concerns involve user input being used to build SQL queries and regular expressions without proper protection, which could lead to SQL injection attacks or excessive resource usage depending on how the code is applied. Additionally, the database service runs with a writable root filesystem and allows privilege escalation via setuid binaries, creating significant risks for container security. These issues are typical of insecure coding practices rather than malicious intent, but they must be fixed urgently to prevent data breaches and system compromise.

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 (437)

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

MALICIOUS PATTERN

Low (2)

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	15	102	0	117
guarddog	Malicious package and install-time behaviour	0	0	2	2

118 of 119 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	Found a potentially user-controllable argument in the construction of a regular expressions. This may result in excessive resource consumption when applied to certain inputs, or when the user is allowed to control the match target. Avoid allowing users to specify regular expressions processed by the	1	/app/controllers/operators_controller.rb:7
HIGH	semgrep	Detected controller which does not enable cross-site request forgery protections using 'protect_from_forgery'. Add 'protect_from_forgery :with => :exception' to your controller class.	1	/app/controllers/application_controller.rb:1
HIGH	semgrep	Detected user input used to manually construct a SQL string. This is usually bad practice because manual construction could accidentally result in a SQL injection. An attacker could use a SQL injection to steal or modify contents of the database. Instead, use a parameterized query which is available	6	/app/controllers/maps_controller.rb:123
HIGH	semgrep	Untrusted input might be used to build a database query, which can lead to a SQL injection vulnerability. An attacker can execute malicious SQL statements and gain unauthorized access to sensitive data, modify, delete data, or execute arbitrary system commands. To prevent this vulnerability, use pre	6	/app/controllers/api/v1/locations_controller.rb:312

SEVERITY	DETECTOR	FINDING	SEEN	EXAMPLE LOCATION
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'.	1	/ Dockerfile:1 2
MEDIUM	semgrep	Service 'db' is running with a writable root filesystem. This may allow malicious applications to download and run additional payloads, or modify container files. If an application inside a container has to save something temporarily consider using a tmpfs. Add 'read_only: true' to this service to p	2	/docker-compose.yml:5
MEDIUM	semgrep	Found an unscoped `find(...)` with user-controllable input. If the ActiveRecord model being searched against is sensitive, this may lead to Insecure Direct Object Reference (IDOR) behavior and allow users to read arbitrary records. Scope the find to the current user, e.g. `current_user.accounts.find	80	/app/controllers/api/v1/regions_controller.rb:73
MEDIUM	semgrep	'html_safe()' does not make the supplied string safe. 'html_safe()' bypasses HTML escaping. If external data can reach here, this exposes your application to cross-site scripting (XSS) attacks. Ensure no external data reaches here.	5	/app/helpers/locations_helper.rb:8
MEDIUM	semgrep	Detected a unquoted template variable as an attribute. If unquoted, a malicious actor could inject custom JavaScript handlers. To fix this, add quotes around the template expression, like this: "<%= expr %>".	1	/app/views/active_storage/blobs/_blob.html.erb:1
MEDIUM	semgrep	Service 'db' allows for privilege escalation via setuid or setgid binaries. Add 'no-new-privileges:true' in 'security_opt' to prevent this.	2	/docker-compose.yml:5
MEDIUM	semgrep	When a redirect uses user input, a malicious user can spoof a website under a trusted URL or access restricted parts of a site. When using user-supplied values, sanitize the value before using it for the redirect.	4	/app/controllers/api_tokens_controller.rb:20

SEVERITY	DETECTOR	FINDING	SEEN	EXAMPLE LOCATION
MEDIUM	semgrep	This code includes user input in <code>`link_to`</code> . In Rails 2.x, the body of <code>`link_to`</code> is not escaped. This means that user input which reaches the body will be executed when the HTML is rendered. Even in other versions, values starting with <code>`javascript:`</code> or <code>`data:`</code> are not escaped. It is better to create	1	/app/helpers/application_helper.rb:6
MEDIUM	semgrep	Found that the setting for providing detailed exception reports in Rails is set to true. This can lead to information exposure, where sensitive system or internal information is displayed to the end user. Instead, turn this setting off.	2	/config/environments/development.rb:13
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. <code>`uses: ac</code>	5	/.github/workflows/main.yml:26
LOW	guarddog	Detects access to sensitive files (credentials, configs, keys) [threat-filesystem-read]	2	/config/initializers/rack_attack.rb:23

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.