False Positives Documentation

FSC Readiness Scanner 0.5.0 (04tfn000005ucSjAAI) — Atlas Advisory LLC — chase@atlasys.io
Scanners: Source Code Scanner / Checkmarx (Scan Id a0OKX000001JNB42AO, run 2026-07-18) and Salesforce Code Analyzer v5 (Security and Recommended rule selectors, run 2026-07-17)

This document explains every flagged result from both scanners. The Checkmarx scan reported **no issues** for any Critical or Serious security query (SOQL/SOSL injection, reflected/stored XSS, client DOM XSS and code injection, sharing, FLS create/update, CRUD delete, XSRF, URL redirection, password APIs, Lightning API version). The remaining results are two Medium-severity query groups and one code-quality group, justified in Part A. Part B covers the Salesforce Code Analyzer findings, all triggered by field *names* that pattern-match secret-storage heuristics.

Part A — Source Code Scanner (Checkmarx) Results

A1. Apex SOQL SOSL User Mode Missing — 8 results (Medium)

Locations	All 8 result paths terminate at the same two sinks: verification SOQL queries in the unit test class `ScannerServiceTest.cls` (L112 and L131). The paths differ only by which parameter of `ScannerService.createManualFinding` they trace (componentName, componentType, detail, severity — two test invocations each).
Justification	The flagged queries without `WITH USER_MODE` exist only in **unit test code**, where they verify that a record was written correctly. Apex test methods cannot be executed at runtime in a subscriber org; they run only under the test framework, so they present no security exposure to subscriber data. The production method these paths trace through, `createManualFinding`, performs its DML with `insert as user` (ScannerService.cls L166), enforcing CRUD/FLS in user mode, after an explicit CRUD pre-check (`ensureWriteAccess()`). Every production (non-test) SOQL statement in the package specifies `WITH USER_MODE` or executes via `Database.queryWithBinds(..., AccessLevel.USER_MODE)`.

A2. Apex CSRF In Aura LWC — 4 results (Medium)

Locations	`scannerConsole.js` L477/L480 (`handleStartScan` → `ScannerService.startScan`, insert at L20), L532 (`createManualFinding`, insert at L166), L497 (`updateFindingStatus`, update at L131).
Justification	This query flags DML in `@AuraEnabled` methods reachable from a Lightning Web Component. The CSRF concern this rule targets is DML executed automatically during component initialization (page load), where a crafted URL could trigger a state change without user intent. None of the flagged paths run on load: `handleStartScan` is invoked only by the "Run Scan" button click, `updateFindingStatus` only by the explicit triage button group acting on user-selected rows, and `createManualFinding` only by the save button of a user-opened dialog. The component's lifecycle hooks

	(`connectedCallback` , `renderedCallback`) and wire adapters perform reads only. In addition, the Lightning framework applies anti-CSRF tokens to all Aura/LWC server requests, and each flagged method performs its DML with `as user` semantics after a CRUD permission pre-check.

A3. DML Statements Inside Loops — 15 results (Code Quality)

Locations	Paths through `ScanOrchestratorTest.cls` (do/while at L101), `FindingRepository.persist` (upsert at L28), `ScanOrchestrator` status updates (L72, L93, L100), and the scanner module loops in `ApexBodyScanner` , `SchemaScanner` , and `DataFootprintScanner` .
Justification	All production DML is bulkified. The scanner-module loops accumulate records into collections; the single DML sink is `FindingRepository.persist` , which performs one `upsert as user` on the full collection *outside* its loop (L28). The orchestrator's status updates (`markRunning` , `advance` , `finalizeScan`) each execute at most once per Queueable transaction. The loop that several result paths originate from is the do/while in the **unit test** `ScanOrchestratorTest` (L101), which re-invokes `execute()` synchronously to simulate checkpoint chaining; at runtime each iteration is a separate asynchronous Queueable transaction with its own governor limits, by design. No path can accumulate unbounded DML within a single transaction: module checkpointing caps the work performed per execution.

Part B — Salesforce Code Analyzer Results

The Code Analyzer Security ruleset produced four findings, all from the same PMD AppExchange rule (`ProtectSensitiveData`), and all triggered by field *names* that pattern-match secret-storage heuristics ("token", "key"). None of these fields store credentials, secrets, or sensitive data.

B1. ProtectSensitiveData — Finding__c.Token__c

Rule / Severity	ProtectSensitiveData (PMD AppExchange), Moderate
Location	`force-app/main/default/objects/Finding__c/fields/Token__c.field-meta.xml`
Scanner message	"Token__c is a potential auth token in a custom object: Finding__c"
Justification	In this product's domain, a "token" is a **FinServ API name** — for example `FinServ__FinancialAccount__c` — detected in the subscriber's own Apex code or schema. The field stores the matched metadata API name so the console can map it to migration guidance. It never contains authentication material, session identifiers, or secrets of any kind. The value is a public Salesforce metadata name that is already visible to any user with the corresponding metadata access.

B2. ProtectSensitiveData — Finding__c.NaturalKey__c

Rule / Severity	ProtectSensitiveData (PMD AppExchange), Moderate
Location	`force-app/main/default/objects/Finding__c/fields/NaturalKey__c.field-meta.xml`
Scanner message	"NaturalKey__c is a potential auth token in a custom object: Finding__c"
Justification	This field stores a **SHA-256 hash of the finding's identity attributes** (component type + component name + matched API name), used purely as a deduplication key so that re-running a scan upserts rather than duplicates findings. It is a deterministic digest of non-secret metadata names — it neither contains nor derives from any credential.

B3. ProtectSensitiveData — Finding__c.RuleKey__c

Rule / Severity	ProtectSensitiveData (PMD AppExchange), Moderate
Location	`force-app/main/default/objects/Finding__c/fields/RuleKey__c.field-meta.xml`
Scanner message	"RuleKey__c is a potential auth token in a custom object: Finding__c"
Justification	This field stores the **developer name of the matching `FindingRule__mdt` custom metadata record** (for example `FinancialAccount_Lookup`), used to join a finding to its packaged remediation guidance. It is a foreign-key-style reference to packaged custom metadata, not an authentication token.

B4. ProtectSensitiveData — FindingRule__mdt.TokenPattern__c

Rule / Severity	ProtectSensitiveData (PMD AppExchange), Moderate
Location	`force-app/main/default/objects/FindingRule__mdt/fields/TokenPattern__c.field-meta.xml`
Scanner message	"TokenPattern__c is a potential auth token in the object: FindingRule__mdt with public visibility"
Justification	This custom metadata field stores a **matching pattern for FinServ API names** (for example `FinServ__FinancialAccount__c`) that the scanner uses to classify findings and attach remediation guidance. Its contents are shipped by us in the package as public product configuration — the FinServ API names being matched are published in Salesforce's own documentation. No secret or subscriber data is ever written to this field; custom metadata is not writable at runtime by the package.

B5. Dynamic SOQL (defense-in-depth notes)

Two methods build SOQL strings dynamically. The Checkmarx SOQL/SOSL Injection query confirmed no injection issues, but for completeness, both are safe by construction:

Location	Why it is not exploitable
`ScannerService.getFindings`	Filter clauses are assembled from fixed string literals only; every user-supplied value is passed exclusively through bind variables via `Database.queryWithBinds(query, binds, AccessLevel.USER_MODE)`. No user input is ever concatenated into the query text, and the query runs in user mode, enforcing CRUD/FLS and sharing.
`DataFootprintScanner.countRows`	The only dynamic element is an object API name that (a) originates from the org's own Schema describe results, not user input; (b) must match the strict allowlist regex `FinServ__[A-Za-z0-9_]+`; and (c) is additionally escaped before use. The query is a `COUNT()` aggregate executed in USER_MODE and returns no field data.

Non-security informational findings in the Recommended report

The attached Recommended-ruleset report also lists style and documentation items that are not security findings: missing ApexDoc comments, PMD's `ApexUnitTestClassShouldHaveRunAs` best-practice suggestion in test classes, `ExcessiveParameterList` (4 parameters), SLDS linter styling-hook suggestions in component CSS, and one LWC `no-async-operation` flag for `setInterval`. The `setInterval` usage is intentional and bounded: it polls the packaged Apex dashboard endpoint every 5 seconds only while a scan is running, and the interval is cleared on scan completion and in `disconnectedCallback`. None of these items involve data access, injection surface, or privilege changes.