HackerOne

Excom

APPLICATION SECURITY ASSURANCE

CONFIDENTIAL

Source Code Security Assessment

`vault-doc-api` · Document & Report Service

ASSESSED COMPONENT
vault-doc-api (Java / Spring Boot 2.7.14)

ASSESSMENT TYPE
Automated white-box source code review

ASSESSMENT WINDOW
27 - 28 August 2026

REPORT ISSUED
28 August 2026

ENGINE
Autonomous multi-agent source code security analysis

RUN REFERENCE
`vda-ae7e35b47769`

FINDINGS ISSUED
7 (1 critical, 1 high, 1 medium, 3 low, 1 informational)

REPORT VERSION
1.0 · Final

Purpose of this document. This report records the scope, method, coverage and results of an automated white-box security review of the `vault-doc-api` source tree. It is written to be presented in internal governance and external assurance reviews as evidence that the component was subjected to a documented security review, that the review's boundaries are stated, and that its conclusions are traceable to specific lines of source code.

Every finding in Section 6 is anchored to a file path and line number in the assessed revision. Section 7 records the lines of enquiry that closed without a finding, and Section 9 records the limits of the method, so that a reader can judge what this assessment does and does not establish.

1. Document control

Document title	Source Code Security Assessment - vault-doc-api
Classification	Confidential. Contains unremediated vulnerability detail and reproduction steps.
Intended audience	Internal and external assurance reviewers, and the engineering team that owns `vault-doc-api`.
Assessment engine	Autonomous multi-agent source code security analysis, run in phases
Run reference	`vda-ae7e35b47769`
Evidence retention	Full machine-readable output, phase artefacts, execution log and step-level trace retained with the run reference. Manifest at Appendix C.
Version	1.0, final, issued 28 August 2026

Contents

> **A note on attribution of claims.** Findings, severities, path-evidence verdicts, data-flow chains and code citations in Section 6 are the output of the assessment engine, and the remediation direction under each finding is generated by agentic reasoning. CWE names, OWASP Top 10 2025 category assignments and NIST SSDF practice references are added by this report template and were not produced by the engine. They are marked as such where they appear.

2. Executive summary

Excom commissioned an automated white-box security review of the `vault-doc-api` service, a multi-tenant Java application that generates documents and reports and shares them with external recipients. The review read the application's source code, traced untrusted input from every HTTP entry point to the operations that act on it, and attempted to establish or rule out a concrete attack for each candidate weakness it identified.

The review issued seven findings. One is critical and warrants immediate action.

Findings at a glance

1	1	1	4
CRITICAL	HIGH	MEDIUM	LOW / INFORMATIONAL

REF	SEVERITY	FINDING	CWE	OWASP 2025	PATH EVIDENCE
F-01	CRITICAL	Unsandboxed expression evaluation of user-supplied report fields yields code execution in the shared process	CWE-917	A05 Injection	Fully traced
F-02	HIGH	Polymorphic default typing on the only JSON mapper lets request bodies choose which classes the server instantiates	CWE-502	A08 Integrity Failures	Fully traced
F-03	MEDIUM	Unvalidated share webhook URL drives blind server-side requests to internal hosts, replayable without credentials	CWE-918	A01 Access Control	Fully traced
F-04	LOW	Missing tenant-ownership check on share creation lets one tenant publish a link over another tenant's report	CWE-639	A01 Access Control	Partially traced
F-05	LOW	Unbounded asynchronous fan-out on the credential-free share endpoint starves a thread pool shared by all tenants	CWE-770	A06 Insecure Design	Fully traced
F-06	LOW	Unbounded share metadata map persists attacker-sized JSON with per-read write amplification	CWE-770	A06 Insecure Design	Partially traced
F-07	INFORMATIONAL	Mutable process-wide static JSON mapper is writable through a public bean setter	CWE-502	A08 Integrity Failures	Partially traced

What the pattern of findings says about the codebase

Five of the seven findings have their root cause in the document-sharing and webhook delivery feature. The authentication layer, by contrast, read as sound: JSON Web Token signatures are verified with a keyed hash, the security context is cleared when verification fails, and no path was found that reaches a protected operation without a valid token. The weaknesses are concentrated in authorization and in input handling rather than in identity.

One finding turns on how a caller-supplied identifier is resolved. In F-04 a report identifier arrives in the request body as a plain string, and the code that resolves it confirms the record exists without confirming who owns it. Tenant identifiers themselves come from the verified token claim rather than from the body, and share tokens travel in the URL path, so the pattern is contained to that one route. Asserting tenant ownership wherever a caller-supplied identifier is resolved would close F-04 and narrow F-03.

The review also recorded what it examined and did not find. Section 7 sets out fifteen lines of enquiry that the assessment pursued at depth and closed without issuing a finding, together with the control or structural property that closed each one. Four of the fifteen were candidate findings that the verification stage rejected. That section exists to evidence the breadth of the review rather than to catalogue near misses.

Recommended sequence

ORDER	FINDING	ACTION
1	`F-01`	Avoid evaluating request-supplied strings as expressions. Either remove the version 2 computed-fields feature or evaluate it in a restricted context that exposes no type references, constructors or reflection.
2	`F-02`	Disable polymorphic default typing on the application's JSON mapper. If polymorphism is required for a specific field, declare it per type rather than globally.
3	`F-03`	Constrain webhook destinations to an allow-list of external hosts, validate the destination immediately before each delivery rather than only at registration, and stop the delivery client following redirects.
4	`F-04`	Assert tenant ownership wherever a caller-supplied report identifier is resolved, using the tenant-scoped repository methods that already exist.
5	`F-05`, `F-06`, `F-07`	Bound the share metadata field, bound the webhook executor's queue and rejection behaviour, and remove the reassignable static mapper in favour of an instance the type cannot replace at runtime.

3. Scope and boundaries

This section states exactly what was assessed. A reader working through the findings should be able to establish from this section alone which artefact was reviewed and which categories of risk the review was not designed to address.

3.1 Assessed artefact

Component	`vault-doc-api`, Excom document and report service
Language and framework	Java, Spring Boot 2.7.14, Spring Security, Spring Data JPA / Hibernate, PostgreSQL, AWS S3
Artefact form	Source tree snapshot, reviewed statically. No running instance was exercised.

3.2 What the assessment covered

The review had read access to the complete source tree and treated the following as in scope:

- All application Java source, including controllers, services, repositories, entities, security configuration, filters and data-type adapters.
- All HTTP request handlers and the request data-transfer objects bound to them, including the validation constraints declared on those objects.
- Framework and application configuration: the Spring Security filter chain, method-level security, JSON serialization configuration, asynchronous executor configuration, outbound HTTP client configuration, and the application property files including profile variants.
- Database schema migrations, to establish which constraints the database enforces independently of the application.
- The declared dependency set and its versions, to establish which framework behaviours apply at those versions.
- Test sources, read to establish what behaviour the project asserts rather than as a target.

3.3 What the assessment did not cover

The following are outside the boundary of this review. Each is a category a reader may expect to see addressed by a separate control.

OUT OF SCOPE	WHY, AND WHERE IT BELONGS INSTEAD
Dynamic and runtime testing	No instance of the service was deployed, authenticated against or attacked. Every conclusion is derived from source. Behaviour that depends on runtime state, live data volumes or infrastructure configuration is outside what this method can settle.
Third-party dependency vulnerabilities	The review read declared dependency versions to reason about framework behaviour. It did not enumerate known vulnerabilities in those dependencies and issues no finding of that class. Software composition analysis is a separate control.
Secret detection	The review did not systematically scan history or working tree for committed credentials, and issues no finding of that class. Secret scanning is a separate control.
Infrastructure, network and cloud configuration	Container images, orchestration manifests, network policy, load-balancer and edge configuration, cloud identity and access policy, and database server hardening were not assessed. Several findings note that their terminal impact depends on deployment properties this review cannot observe.
Business logic correctness	Defects that are wrong behaviour without a security consequence were not pursued.
Client-side and front-end code	The assessed artefact is a JSON API with no server-rendered user interface. No browser-side code was in scope.
Components referenced but not present	The assessment established that no code in the reviewed tree writes report records, and that the `document_shares` table used by the sharing feature is absent from the migration set. Those components are supplied outside this artefact and were not assessed. Section 9.3 records what this means for the findings that depend on them.

3.4 Trust boundaries used in the analysis

The review reasoned about three classes of caller. Every finding names which class it applies to, which is what makes the severities comparable to each other.

CALLER CLASS	DEFINITION USED
Anonymous	No credential presented. Reaches only the routes the security configuration explicitly permits without authentication.
Authenticated tenant user	Holds a legitimately issued, signature-valid token identifying a real user in a real tenant. The review treats the token's claims as unforgeable, because signature verification was confirmed effective.
Authenticated user of another tenant	As above, but acting against data belonging to a different tenant. This is the class that matters for the multi-tenancy findings.

4. Assessment methodology

The assessment was performed by an autonomous multi-agent source code security analysis system, which reasons about the code in phases rather than matching patterns. The method is documented here in enough detail that a reader can judge the weight to give its conclusions.

4.1 Agentic analysis phases

STAGE	WHAT HAPPENS	PURPOSE
1. Survey	The engine reads the repository structure, configuration, dependency manifest, schema and entry points, then selects the regions of code where a security defect would have the greatest consequence.	Determines where effort is spent. Regions the survey does not select receive less scrutiny, which is a coverage limit stated in Section 5.
2. Investigate	Independent investigators work the selected regions in parallel. Each reads code, follows calls across files, and records a candidate weakness only when it can name the untrusted input, the path it travels and the operation it reaches.	Produces candidates with a stated mechanism rather than pattern matches. Investigators work without shared context, so the same defect is often recorded more than once by different routes. Section 5.3 gives the count before and after grouping.
3. Consolidate	Candidates that share a root cause are grouped. One representative is carried forward and the rest are recorded as further manifestations of the same underlying defect.	Prevents one root cause from appearing as many findings. The manifestation count on each finding in Section 6 shows how many independent observations collapsed into it.
4. Verify	Each group is examined twice, in two passes that do not share context. The first re-derives the claim from the source without seeing the original reasoning. The second is directed to attack the claim: to find the control, the unreachable step or the missing precondition that makes it wrong, and to reduce the severity if the evidence supports less than was claimed.	This is the step that governs report quality. It is why some findings in Section 6 argue *against* their own most severe reading, and it is the source of most of the material in Section 7.
5. Report	Verified groups are ranked into a reporting tier, assigned a severity, and written up with the citations that support each claim.	Produces the machine-readable output this document renders.

4.2 Evidence and proof standard

Each finding carries a path-evidence verdict. The distinction is load-bearing: it states what the source evidence reached, not how dangerous the defect is.

VERDICT	WHAT IT ASSERTS
Fully traced	Every step from an attacker-reachable entry point to the security-relevant effect is established in the source, with a citation for each step, and no control on the path prevents it. There is no unresolved gap in the chain.
Partially traced	The defect itself is established in the source, but at least one step needed to complete an attack could not be settled from the source alone. The report states which step and why. These findings are real defects whose terminal impact is unconfirmed.

Where the assessment could not settle a question, it says so rather than assuming the unfavourable answer. Section 9 collects those cases.

4.3 How severity was assigned

Severity reflects who is harmed and how, not the name of the weakness class. Two properties drove the ratings in this report:

- **Whether the victim is someone other than the attacker.** A defect that only lets a caller degrade their own experience is rated below one that reaches another tenant or the operator, even where the code defect is identical.
- **Whether the impact was traced or inferred.** F-04 describes an unambiguous authorization defect rated low because the step that would turn it into a cross-tenant breach depends on knowing a report identifier, which nothing in the reviewed code discloses. The rating states what the evidence reached, not a judgement that the defect is acceptable.

Appendix A gives the full definitions.

5. Coverage statement

The first question to ask of an automated assessment is how much of the application it actually looked at. This section answers that from the run's own records. Every number below is taken from the retained phase artefacts and execution log listed in Appendix C, and each is labelled with what it counts.

5.1 Source coverage

ARTEFACT	EXTENT REVIEWED	HOW THIS IS ESTABLISHED
Java source files	**33 of 34**	The investigation record names 33 of the repository's 34 Java files as the location of a hypothesis, a candidate weakness or a control. The one file it never names is `VaultDocApplication.java`, the Spring Boot bootstrap class, which contains no security logic.
HTTP entry points	**6 of 6**	Every request handler declared in the three controllers was traced from the request body to the operations it reaches. Enumerated in section 5.2.
Security configuration	Reviewed in full	`SecurityConfig.java`, the JWT authentication filter, the principal type, the Jackson configuration, both asynchronous executor definitions and the HTTP client configuration were each read and cited as controls or as defects.
Application configuration	Reviewed in full	`application.yml` and `application-dev.yml` were read. Absent settings were treated as findings in their own right, including the missing request body size cap and the absent egress policy.
Database schema	Reviewed in full	Both Flyway migrations were read, including a check for row-level security policies and for consumers of the stored template table.
Dependency manifest	Reviewed, no bill of materials supplied	`pom.xml` was read and two dependencies carrying published advisories were investigated by searching the source for calls into them. No software bill of materials was provided to the assessment, so this is a source-reachability review and not a dependency inventory. See section 9.
Build, CI and container definitions	PARTLY REVIEWED	The Dockerfile, the compose files and `.env.example` were read and cited during investigation. The CI workflow was named in survey notes in connection with a committed test credential, addressed at 7.9. None of these files was assessed as a build-integrity or supply-chain surface. Stated as a limitation in section 9.
Test sources	NOT REVIEWED FOR DEFECTS	Test files were read only to establish whether a production caller exists for a given method. They were not assessed as an attack surface.

5.2 Entry-point coverage

REQUEST HANDLER	AUTHORIZATION REQUIRED	TRACED	FINDINGS REACHING THIS ENTRY POINT
`POST /api/v1/reports/generate`	`ROLE_REPORT_GENERATE`	Yes	None. Examined for expression injection and cleared. See 7.3.
`GET /api/v1/reports/templates`	Authenticated	Yes	None.
`POST /api/v2/reports/build`	`ROLE_REPORT_GENERATE`	Yes	**F-01** (critical)
`POST /api/v2/shares`	`ROLE_REPORT_SHARE`	Yes	**F-02, F-03, F-04, F-06, F-07**
`GET /api/v2/shares`	`ROLE_REPORT_SHARE`	Yes	**F-02** (the availability effect lands here)
`GET /api/v2/shares/access/{token}`	None. The share token is the credential.	Yes	**F-03, F-04, F-05, F-06**

5.3 Depth of investigation

The assessment did not stop at the seven findings it reported. It recorded a written hypothesis and a written outcome for every attack it considered, including the ones that failed. Those negative records are the substance of the coverage claim, and they are the source of Section 7.

60	338	198	6
CODE REGIONS SELECTED FOR INVESTIGATION	ATTACK HYPOTHESES RECORDED AND RESOLVED	HYPOTHESES TESTED AND RULED OUT WITH A WRITTEN REASON	CANDIDATE FINDINGS REJECTED AT VERIFICATION

STAGE	COUNT	DESCRIPTION
Code regions selected	60	The run was configured to carry forward at most 60 regions, and the survey stage filled that allocation with the regions it judged highest-consequence. 33 were directed at a specific file or package; 27 were given no target and chose their own.
Hypotheses recorded	338	Every attack the investigators formulated. Each carries the code location it concerns and a written resolution.
carried forward as candidates	140	Hypotheses where the investigator could name the untrusted input, the path and the operation reached.
tested and ruled out	198	Hypotheses closed against a control, an unreachable step or a missing precondition, with the closing reason written down and cited.
Distinct root causes after grouping	109	The 140 candidates collapsed to 109 distinct underlying defects. Investigators working without shared context reach the same defect by different routes, which is why the ratio is this high.
Root causes taken to verification	109 of 109	No group was skipped. The run's verification capacity was not exhausted.
upheld	99	Both verification roles agreed the claim holds.
rejected	6	Rejected on re-derivation from the source. Each is described in Section 7.
left unresolved	4	A step could not be settled from source alone. Three of the four were published anyway, with the unsettled step named in the finding. See section 9.
Findings issued	7	The 99 upheld root causes reduce to 7 reported defects, because one reported defect commonly accounts for many root-cause groups that share a mechanism. F-01 alone absorbs 38 of them.

5.4 Vulnerability classes examined

The table below lists every weakness class the analysis agents focused on, with the volume of investigative effort it received and where that effort ended up. Classes with no candidates and a high ruled-out count are the ones the assessment looked hard at and did not find. The right-hand column points to the entry in Section 7 that explains each one.

WEAKNESS CLASS	CANDIDATES CARRIED FORWARD	HYPOTHESES RULED OUT	OUTCOME
Expression injection, Spring EL (CWE-917)	49	53	**F-01**, critical
Cross-tenant object access (CWE-639)	36	58	**F-04**, low
Server-side request forgery (CWE-918)	33	5	**F-03**, medium
Deserialization, polymorphic typing (CWE-502)	16	33	**F-02** high, **F-07** informational. Escalation to code execution ruled out, see 7.1

WEAKNESS CLASS	CANDIDATES CARRIED FORWARD	HYPOTHESES RULED OUT	OUTCOME
Unbounded resource consumption (CWE-770)	3	4	**F-05** low, **F-06** low
JavaScript execution, script engine (CWE-94)	0	31	No finding. See 7.2
Signature bypass, token forgery (CWE-347)	0	18	No finding. See 7.7
Trusted-context shadowing (CWE-915)	1	15	Rejected at verification. See 7.10
Server-side template injection (CWE-1336)	0	10	No finding. See 7.4
Predictable identifier, enumeration (CWE-340)	0	11	No finding. See 7.14
Path traversal into storage keys (CWE-22)	0	7	No finding. See 7.12
Missing function-level authorization (CWE-862, CWE-863)	0	6	No finding. See 7.13
Vulnerable third-party component (CWE-1395)	0	5	No finding. See 7.15
Race condition, time-of-check to time-of-use (CWE-362)	0	5	No finding. Closed on consequence: the affected counter is read by nothing that makes a decision.
Insufficient session or link expiration (CWE-613)	1	6	Rejected at verification. See 7.8
Error-message information disclosure (CWE-209)	0	4	No finding. Every reflected message is a static string or an echo of the caller's own input.
Stored, second-order expression injection (CWE-917)	0	2	No finding. See 7.5
Hardcoded credential (CWE-798)	1	0	Rejected at verification. See 7.9
SQL and JPQL injection (CWE-89)	0	1	No finding. See 7.6
Timing side channel (CWE-208)	0	1	No finding. An existence oracle yields no search-space reduction against a 122-bit token.
Management-endpoint exposure (CWE-497)	0	1	No finding. The configured Actuator routes are described at the end of Section 8.
Authorization granularity on share access (CWE-863)	0	1	No finding, with a stated expiry condition. See 8.2.
Fail-open on swallowed exception (CWE-390)	0	1	No finding as a defect in its own right; carried inside F-01 as an aggravating factor.
Spreadsheet formula injection (CWE-1236)	0	1	No finding. No spreadsheet writer exists in the application.
HTTP parameter pollution (CWE-235)	0	1	No finding.

The candidate column sums to 140, the full set carried forward. The ruled-out column does not sum to 198, because a single written hypothesis can bear on more than one class and is counted under each class it addresses. Class attribution in the ruled-out column is assigned from each hypothesis's own text and code location rather than from a label the engine emitted. Classes absent from this table

were not examined; see section 9.2.

6. Findings

Seven findings follow, ordered by severity. Each begins with a summary block, then explains the defect, traces the path from the request to the affected operation, states what an attacker achieves, and records the bounds the assessment established on its own claim. Code excerpts are quoted from the reviewed source at the file and line each one cites.

Remediation direction is generated by agentic reasoning. Excom should validate each one against its own design before implementing it.

Read the **Assessment bounds** block on each finding. It is where the verification stage recorded what it tried to establish and could not. That block is the reason several findings here are rated lower than the name of their weakness class would suggest.

REF	FINDING	SEVERITY	EVIDENCE	CWE	OWASP 2025
F-01	Unsandboxed expression evaluation of request-body strings	CRITICAL	Fully traced	917	A05 Injection
F-02	Polymorphic deserialization lets the request choose instantiated classes	HIGH	Fully traced	502	A08 Integrity failures
F-03	Unvalidated webhook destination drives server-side requests	MEDIUM	Fully traced	918	A01 Access control
F-04	No ownership check when sharing another tenant's report	LOW	Partially traced	639	A01 Access control
F-05	Credential-free trigger saturates a shared executor	LOW	Fully traced	770	A06 Insecure design
F-06	Unbounded metadata field with per-read write amplification	LOW	Partially traced	770	A06 Insecure design
F-07	Mutable process-wide static writable through a public setter	INFORMATIONAL	Partially traced	502	A08 Integrity failures

Finding 1 of 7 · Critical · CWE-917 · OWASP A05:2025 Injection

F-01 · engine reference `e3f1159e589e`

Request-body strings are evaluated as Spring Expression Language with no sandbox, giving code execution in the shared process

SEVERITY	STANDARD OF EVIDENCE
CRITICAL	Fully traced: every step from the entry point to code execution is cited. One precondition on the shape of the request body is recorded in the bounds block
ROOT CAUSE `src/main/java/com/vaultdoc/engine/TemplateEngine.java:39`, method `evaluateExpression`	ENTRY POINT `POST /api/v2/reports/build`
WHO CAN EXPLOIT IT Any authenticated customer holding `ROLE_REPORT_GENERATE`, the routine report-generation authority	SECURITY PROPERTIES LOST Confidentiality, integrity, availability and privilege separation, for every tenant on the platform
INDEPENDENT OBSERVATIONS 38 root-cause groups shared this mechanism and were collapsed into one finding at the reporting stage	FRAMEWORK REFERENCES CWE-917 Improper Neutralization of Special Elements used in an Expression Language Statement · OWASP A05:2025 · NIST SSDF PW.5.1, PW.7.2 (added by this report, see Section 1)

PRIORITY REASONING

An ordinary customer with the permission needed to generate a routine report can run arbitrary Java, and therefore arbitrary operating-system commands, inside the Java process that serves every tenant. From that position the platform-wide token signing secret, the shared database credentials and every tenant's stored reports are all in reach. The victims are other tenants and Excom itself, not the caller. The assessment established the execution path end to end against the source.

HOW THE DEFECT WORKS

`TemplateEngine.evaluateExpression` parses a caller-supplied string with a default `SpelExpressionParser` and evaluates it against a bare `new StandardEvaluationContext()`. Under the Spring version this application pins, that default context installs a type locator and reflective method and constructor resolvers, so `T(java.lang.Runtime)`, `new java.lang.ProcessBuilder(...)` and arbitrary reflective invocation all resolve. No restricted context, type allow-list, expression allow-list, length cap or sanitizer exists anywhere on the path.

The design intends this evaluator to receive administrator-registered text only. `ReportService` records that intent in a comment at lines 30 to 31, and `TemplateDefinition` repeats it at lines 14 to 15. `ReportService.buildCustomReport` breaks that contract by routing raw request-body strings through the identical helper. Validation does not compensate: the strict character allow-list on `computedFields` is bound to the map *key*, while the expression *value* carries only a non-blank check. The controller's own comment claiming that expression syntax is validated by the template engine is not true of this path.

Authorization is genuinely enforced on the endpoint. That is not the problem. The problem is that the authority it enforces, `ROLE_REPORT_GENERATE`, is the same everyday permission that gates the benign version 1 report endpoint, so the population able to reach this sink is the population able to generate reports at all.

PATH FROM REQUEST TO CODE EXECUTION

1. Request body deserialized into `CustomReportRequest` during handler-argument resolution; the role check then admits the call to the handler
   api/v2/ReportController.java:42
2. Map values carry only `@NotBlank`; the character pattern applies to keys
   model/CustomReportRequest.java:36–38
3. Raw values handed to the legacy adapter alongside trusted template expressions
   service/ReportService.java:69
4. Each value forwarded verbatim; per-entry exceptions caught, logged and nulled
   compat/LegacyTemplateAdapter.java:44–50
5. Parsed and evaluated against an unrestricted evaluation context
   engine/TemplateEngine.java:39

WHAT AN ATTACKER ACHIEVES

One authenticated request executes attacker-chosen Java as the service account. Exploitation is blind with respect to the HTTP response, because computed values are never echoed and the stub layout renders only a timestamp, so results come back out of band through the payload itself. The side effect happens before any rendering or storage step, and the per-entry exception swallowing means failed probes return without error, so an attacker can iterate silently on an endpoint the code itself marks as not yet rate-limited.

Escalation past the attacker's own tenant is grounded in configuration rather than inferred. The symmetric token signing key is injected from an environment variable into the authentication filter, and identity, tenant and authorities are taken verbatim from signed claims, so recovering that key permits minting a token for any user, tenant or role. The same process holds the shared database credentials and a single storage client over one bucket in which every tenant's reports are separated only by a key prefix.

REMEDIATION DIRECTION

Avoid evaluating caller-supplied strings. The narrow change is to give this call site its own evaluator built on Spring's `SimpleEvaluationContext`, which removes type references, constructors and reflective invocation, rather than sharing the unrestricted context that `TemplateEngine` builds for registered template text, and to reject any `computedFields` value that is not drawn from a registered expression. Keep the character allow-list on the map key and add a separate constraint on the value: moving the existing constraint would reject registered expression syntax without protecting anything. Treat the swallowed per-entry exception as part of the fix, because silent failure is what makes probing free.

EVIDENCE

LOCATION	SOURCE	WHAT IT ESTABLISHES
engine/TemplateEngine.java:37–39	`StandardEvaluationContext evalContext = new StandardEvaluationContext(); context.forEach(evalContext::setVariable); return parser.parseExpression(expression) .getValue(evalContext);`	Caller-supplied strings are evaluated as expressions against an unrestricted context, enabling type references, constructors and reflective method invocation.
service/ReportService.java:69	`Map<String, Object> customComputed = legacyAdapter.resolveExpressions( request.getComputedFields(), context);`	Raw request-body strings are routed into the evaluator that the design reserves for administrator-registered static expressions.
compat/LegacyTemplateAdapter.java:44–50	`Object value = templateEngine .evaluateExpression(entry.getValue(), context); resolved.put(entry.getKey(), value); } catch (Exception e) { log.warn("Failed to evaluate expression for field '{}': {}", entry.getKey(), e.getMessage()); resolved.put(entry.getKey(), null);`	Each attacker-supplied value reaches the sink verbatim, and per-entry exception swallowing means side effects persist while probing stays silent.
model/CustomReportRequest.java:36–38	`private Map<@Pattern(regexp = "^[a-zA-Z_][a-zA-Z0-9_]{0,63}$", ...) String, @NotBlank String> computedFields;`	The character allow-list is a container-element constraint on the map key only. The expression value carries just a non-blank check, so expression text is unconstrained.
api/v2/ReportController.java:42	`@PostMapping("/build") @PreAuthorize("hasAuthority('ROLE_REPORT_GENERATE')")`	The entry point requires only the ordinary report-generation authority, which also gates the benign version 1 endpoint.
config/jwt/JwtAuthenticationFilter.java:41–50	`Claims claims = Jwts.parserBuilder() .setSigningKey(Keys.hmacShaKeyFor(jwtSecret... List<String> roles = claims.get("roles", List.class); String tenantId = claims.get("tenant_id", String.class);`	Identity, tenant and authorities are trusted verbatim from a token signed with a symmetric secret resident in the compromised process, so recovering that secret yields cross-tenant impersonation.
application.yml:39–41	`vaultdoc: jwt: secret: ${JWT_SECRET}`	The signing key is a process environment value, readable by in-process code obtained through this sink.

LOCATION	SOURCE	WHAT IT ESTABLISHES
service/StorageService.java:24–28	`String key = String.format("reports/%s/%s.%s", tenantId, reportId, format); s3Client.putObject(PutObjectRequest.builder() .bucket(bucketName)...`	All tenants' reports live in one bucket separated only by key prefix, with a single in-process client, so code execution reaches other tenants' stored reports.

ASSESSMENT BOUNDS

- **The exploit document must satisfy the mapper configuration reported in F-02.** Polymorphic default typing is enabled on the only JSON mapper in the application context, and it applies to the root value of every request body. Because this endpoint's request type is not final, a body reaching this sink has to carry the class identifier that configuration expects; a plain JSON object is rejected during binding, before any expression is read. The assessment established the path from binding to execution and did not construct that document. The coupling runs both ways: F-02's configuration is a requirement an attacker here must meet, and removing that configuration removes the requirement without touching this sink. Excom should treat neither fix as sufficient on its own.
- **Unauthenticated exploitation was considered and ruled out from the source.** The only route in the security configuration that permits anonymous access is the share-access prefix, and it has no call path to the template engine. A valid token is required.
- **The token signature cannot be forged to obtain the required authority.** Signature verification was examined separately and holds. The attacker must be a legitimately issued account.
- **Data read-back needs outbound network access from the application.** Code execution itself does not. If Excom's deployment blocks egress, that reduces exfiltration, not execution.

Finding 2 of 7 · High · CWE-502 · OWASP A08:2025 Software or Data Integrity Failures

F-02 · engine reference `d667f1298806`

Polymorphic deserialization is enabled on the only JSON mapper, so the request document chooses which classes the server instantiates

SEVERITY **HIGH**	STANDARD OF EVIDENCE **Fully traced: class instantiation and two denial-of-service effects are each traced to cited lines**
ROOT CAUSE `src/main/java/com/vaultdoc/config/ShareJacksonConfig.java:39`, bean `shareObjectMapper`	ENTRY POINT **Any request carrying a JSON body. The effect surfaces on `POST /api/v2/shares` and `GET /api/v2/shares`**
WHO CAN EXPLOIT IT **Any authenticated user. No particular authority is needed, because body binding runs before the authorization check**	SECURITY PROPERTIES LOST **Integrity and availability, including availability for other members of the affected tenant**
INDEPENDENT OBSERVATIONS **9 observations across 6 grouped root causes**	FRAMEWORK REFERENCES **CWE-502 Deserialization of Untrusted Data · OWASP A08:2025 · NIST SSDF PW.9.1, PW.9.2 (added by this report)**

PRIORITY REASONING

The analysis sought remote code execution here and did not establish it. The type validator on this mapper excludes the third-party namespaces that public gadget research targets, so the established ceiling is attacker-chosen class instantiation and denial of service rather than code execution. The validator is not a complete barrier: `java.util.logging` and the whole of `com.vaultdoc` pass its prefix rule, and two partially traced escalation routes inside that admitted set are recorded in the bounds block below. The finding is rated on what was established in the source. Section 7.1 records the code-execution attempt in full, because the part of that control that does hold is itself a result Excom should keep.

HOW THE DEFECT WORKS

`ShareJacksonConfig` builds a hand-rolled JSON mapper and enables global polymorphic default typing for every non-final type. The only guard is a validator built from three coarse package-name prefixes: `java.util`, `java.time` and `com.vaultdoc`. Prefix matching admits the whole subtree under each, so the admitted set includes `java.util.concurrent`, `java.util.logging`, `java.util.zip` and every class in the application's own package tree. The adjacent comment describing these as known-safe packages overstates what the rule does.

Under default typing for non-final types, each such position in the document, including the root object and every `Object`-typed map value, is read as a wrapper array carrying a class name. The document, rather than the server-side schema, selects the class the runtime loads, constructs and populates.

Two facts make this reachable by any caller. First, this bean is the only JSON mapper in the application context, so the framework's own auto-configured mapper stands down and this one parses every request body. Second, body binding happens during handler-argument resolution, which runs before the method-level authorization check and before bean validation. Holding any accepted token is sufficient.

PATH FROM REQUEST TO INSTANTIATION

1. Token signature verified; any authority accepted
   `config/jwt/JwtAuthenticationFilter.java:41–45`
2. Body converted during argument resolution, before `@PreAuthorize` and before `@Valid`
   `api/v2/ShareController.java:43–46`
3. Attacker-controlled `Object` slot: the one field on the DTO with no constraint
   `model/ShareRequest.java:41`
4. Default typing resolves the class named in the document and instantiates it
   `config/ShareJacksonConfig.java:39`
5. Second-order effect: the same mapper writes and re-reads the stored metadata column
   `repository/JsonbType.java:82 and :68`

WHAT AN ATTACKER ACHIEVES

Two effects were established from the admitted set.

Thread exhaustion in the shared process. `java.util.Timer` passes the prefix rule, has a public no-arg constructor, exposes no properties the mapper would object to, and its constructor starts a non-daemon thread that this code never cancels. No request-body size cap is configured, so a single body containing repeated timer entries leaks one live operating-system thread per entry, in the process serving every tenant. The request's eventual rejection is immaterial, because the instantiation already happened.

A stored denial of service that survives restarts. On the write side the mapper emits a class identifier for the runtime type without consulting the validator. A value whose class falls outside the three prefixes is therefore written successfully and then rejected permanently on read. Because the tenant share listing loads every share row belonging to the tenant, one poisoned row returns a server error to every co-tenant, and no delete or patch endpoint exists to clear it. Recovery requires an operator editing the database.

REMEDIATION DIRECTION

Remove global default typing from this mapper. Where polymorphism is genuinely required, annotate the specific types that need it and enumerate permitted subtypes explicitly rather than by package prefix. Add a request body size limit at the container. Give the stored metadata column a validated shape so a value that cannot be read back can never be written. Independently, add a path for operators to repair or remove an unreadable share row without direct database access.

EVIDENCE

LOCATION	SOURCE	WHAT IT ESTABLISHES

config/ShareJacksonConfig.java:34–39	`BasicPolymorphicTypeValidator ptv =`
	
	`BasicPolymorphicTypeValidator.builder()`
	`  .allowIfSubType("java.util")`
	`  .allowIfSubType("java.time")`
	`  .allowIfSubType("com.vaultdoc")`
	`  .build();`
	`mapper.activateDefaultTyping(ptv,`
	
	`ObjectMapper.DefaultTyping.NON_FINAL);`
		Default typing is enabled with only package-prefix rules, so the request document selects the class instantiated across all of two standard-library subtrees and the application's own package tree.

config/ShareJacksonConfig.java:25–27	`@Bean("shareObjectMapper")`
	`public ObjectMapper shareObjectMapper() {`
	`  ObjectMapper mapper = new ObjectMapper();`
		This is the only mapper bean in the context, so the framework's conditional auto-configuration backs off and the HTTP message converter is built from this one.

model/ShareRequest.java:41	`private Map<String, Object> metadata;`	An unconstrained object-typed request slot supplies the polymorphic value position the attacker controls.

api/v2/ShareController.java:43–46	`@PostMapping`
	`@PreAuthorize("hasAuthority('ROLE_REPORT_SHARE')")`
	`public ResponseEntity<ShareResponse> createShare(`
	`  @Valid @RequestBody ShareRequest request,`
		Deserialization occurs during argument resolution, before the method-security check and before validation, so no particular authority is needed to reach the sink.

repository/JsonbType.java:68 and :82	`return objectMapper.readValue(json, Map.class);`
	`...`
	`st.setObject(index,`
	
	`objectMapper.writeValueAsString(value),`
	`  Types.OTHER);`
		The same mapper writes and reads the stored metadata column. Serialization emits a runtime-class identifier without consulting the validator, so a value outside the allowed prefixes is stored successfully and then permanently rejected on read.

service/ShareService.java:86	`return shareRepository`
	
	`.findByCreatedByTenantIdOrderByCreatedAtDesc(tenantId)`
		The tenant listing loads every share row, so one unreadable value denies the endpoint to all members of that tenant.

exception/GlobalExceptionHandler.java:52–55	`log.error("Unhandled exception", ex);`
	`return ResponseEntity`
	
	`.status(HttpStatus.INTERNAL_SERVER_ERROR)`
	`  .body(...)`
		Hydration failures surface as a generic server error. This confirms the availability effect and confirms that no error text leaks back to the attacker.

ASSESSMENT BOUNDS

- **Code execution was sought and not established.** The gadget classes public research relies on live in third-party libraries under packages the prefix rules exclude. Inside the admitted set, which includes the whole of `com.vaultdoc`, no side-effecting constructor or setter chain was found to reach code execution. See Section 7.1.
- **Two escalation routes are recorded as partially traced.** File creation through a logging handler, and nulling the process-wide mapper described in F-07. Both depend on library internals that are not resolvable from this repository.
- **The attacker must hold a valid token.** Signature verification holds, and the one anonymous route accepts no request body.
- **This configuration is also a precondition on F-01.** Default typing applies to the root value of every request body, so an exploit of the expression sink in F-01 must carry a class identifier of the kind described above. Removing default typing removes that requirement and leaves the expression sink exactly as exposed, which is why the two remediations are independent rather than alternatives.

Finding 3 of 7 · Medium · CWE-918 · OWASP A01:2025 Broken Access Control (server-side request forgery)

F-03 · engine reference `a4859d4de6f4`

An unvalidated webhook address makes the server issue attacker-directed requests to internal hosts, replayable without credentials

SEVERITY	STANDARD OF EVIDENCE
MEDIUM	Fully traced: the outbound request, the absence of any destination policy and the credential-free replay route are each cited
ROOT CAUSE	ENTRY POINT
`src/main/java/com/vaultdoc/service/WebhookNotificationService.java:74`, method `deliverWebhook`	`POST /api/v2/shares` to register; `GET /api/v2/shares/access/{token}` to replay with no credentials
WHO CAN EXPLOIT IT	SECURITY PROPERTIES LOST
An authenticated user holding `ROLE_REPORT_SHARE` to register the destination, then anyone at all to replay it	Integrity of internal requests, and availability of the shared notification pool across tenants
INDEPENDENT OBSERVATIONS	FRAMEWORK REFERENCES
28 observations across 22 grouped root causes	CWE-918 Server-Side Request Forgery · OWASP A01:2025 · NIST SSDF PW.1.3 (added by this report)

PRIORITY REASONING

The attacker fully controls the destination of a request the server makes from inside Excom's network, and can replay it without credentials. What holds the rating at medium is that the primitive is blind and the request body is fixed by the server. The attacker learns nothing back, and cannot shape an arbitrary internal API call. Whether a reachable internal endpoint acts on a bare JSON POST is a property of Excom's deployment, not of this code, so the assessment declined to assume one exists.

HOW THE DEFECT WORKS

The delivery method passes a caller-supplied string straight into the HTTP client. No scheme allow-list, address-range check, name-resolution check, interceptor or egress proxy exists on the path. The client bean sets connect and read timeouts and nothing else, and the application configuration declares no proxy or egress policy.

The only constraint on the value is a syntactic URL validation annotation declared with no protocol, host or port attributes, which is a parse check and nothing more. Loopback addresses, private address ranges and the cloud link-local metadata address all satisfy it.

Reachability has two hops, and the second is the significant one. Registering the destination requires a verified token bearing the share authority. Re-triggering it requires nothing: the share-access route is anonymous by design, and every hit on it re-fires the stored destination for as long as the share is valid, up to one week.

PATH FROM REQUEST TO OUTBOUND CALL

1. Share created by an authenticated caller holding the share authority
   `api/v2/ShareController.java:43–46`

2. Destination constrained only by a syntactic URL check with no protocol or host attributes
   model/ShareRequest.java:29
3. Stored verbatim, then dispatched on the sharing executor
   service/ShareService.java:43, dispatched at :54
4. Server POSTs to the attacker's chosen address; response and errors discarded
   service/WebhookNotificationService.java:74
5. Anonymous replay: every share redemption re-fires the same outbound call
   service/ShareService.java:71

WHAT AN ATTACKER ACHIEVES

The attacker chooses scheme, host, port, path and query of a JSON POST issued from the application's network position, reaching loopback, private-range and link-local addresses that are not reachable to them directly, and replays it arbitrarily often with no credentials and no rate limit anywhere in the codebase.

A second effect is availability, and it crosses tenants. The notification executor is a single pool shared by every tenant's deliveries, sized at two core threads, four maximum and a fifty-slot queue, with no custom rejection policy. A destination that accepts the connection and then never answers holds a thread for the ten-second read timeout. A destination that trickles one byte at a time holds one indefinitely, because that timeout governs each read rather than the whole exchange. Sustained replay saturates the pool, and the resulting rejection propagates synchronously out of the unguarded asynchronous calls, so other tenants' share creations fail with a server error after their row has already been committed, and other tenants' recipients receive an error instead of report metadata.

REMEDIATION DIRECTION

Validate the destination, not just the syntax. Restrict the scheme to HTTPS, resolve the host and reject loopback, private, link-local and multicast results, stop the client following redirects, and prefer an explicit allow-list of customer-registered endpoints over open registration. Webhook delivery already runs on its own executor, so the change there is a rejection policy that does not propagate into the request thread, plus a per-share delivery cap so a single share cannot be used as an amplifier.

EVIDENCE

LOCATION	SOURCE	WHAT IT ESTABLISHES

service/WebhookNotificationService.java:74–77	`webhookRestTemplate.postForEntity(`
	`  url, request, String.class);`
	`...`
	`} catch (Exception e) {`
	`  log.warn("Webhook delivery failed for url={}...`
		A fully attacker-chosen address is POSTed to from the server with no destination validation, and the response and all errors are discarded, which is what makes the primitive blind.

model/ShareRequest.java:29	`@org.hibernate.validator.constraints.URL(` `  message = "Webhook URL must be a valid URL")`	The only guard is a syntactic check with no protocol, host or port attributes, so loopback, private-range and link-local destinations all validate.

LOCATION	SOURCE	WHAT IT ESTABLISHES

config/WebhookRestTemplateConfig.java:20–23	`return builder`
	
	`.setConnectTimeout(Duration.ofSeconds(5))`
	
	`.setReadTimeout(Duration.ofSeconds(10))`
	`  .build();`
		The HTTP client is configured with timeouts only. No interceptor, request-factory restriction, proxy or destination policy exists.

service/ShareService.java:43 and :54	`entity.setWebhookUrl(request.getWebhookUrl());`
	`...`
	`webhookService.notifyShareCreated(entity);`
		The request-body address is persisted verbatim and dispatched, connecting the authenticated boundary to the outbound sink.

config/SecurityConfig.java:30–32	`.antMatchers("/api/v2/shares/access/**").permitAll()`
	`.antMatchers("/api/**").authenticated()`
	`.anyRequest().denyAll()`
		Registration is authenticated, but the route that re-fires the stored address is anonymous, giving credential-free replay.

service/ShareService.java:71	`if (share.getWebhookUrl() != null) {`
	
	`webhookService.notifyShareAccessed(share);`
		Every hit on the anonymous endpoint re-issues the outbound call, with no rate limit and no access-count cap.

config/AsyncConfig.java:28–30	`executor.setCorePoolSize(2);`
	`executor.setMaxPoolSize(4);`
	`executor.setQueueCapacity(50);`
		One bounded pool serves every tenant's deliveries, so black-holed destinations plus unthrottled replay starve other tenants' notifications and can fail their requests.

service/WebhookNotificationService.java:41–45	`payload.put("event",`
	`"share.created");`
	`payload.put("shareId",`
	`share.getId());`
	`payload.put("reportId",`
	`share.getReportId());`
	`payload.put("recipientEmail",`
	`    share.getRecipientEmail());`
		The POST body is a fixed server-built key set, so the attacker controls the destination but cannot shape an arbitrary internal API request.

ASSESSMENT BOUNDS

- **Confidentiality impact is not established.** The response, status, error and timing are all discarded, and delivery is asynchronous, so no channel returns anything to the attacker.
- **Cloud credential theft through the metadata service is out of reach.** The sink issues only POST requests. The version 1 metadata interface requires GET and version 2 requires PUT. Apache HttpClient 4.5 reaches the classpath transitively through the AWS SDK, and its default redirect strategy follows only GET and HEAD, so a redirect cannot convert this POST into either.
- **The reachable scheme set is narrower than the annotation allows.** The URL constraint accepts any scheme `java.net.URL` can parse, but delivery goes through `RestTemplate`, which handles `http` and `https` only. Schemes such as `file` are not reachable through this sink.
- **Attacker-supplied metadata is not forwarded.** The data-transfer object's own documentation claims metadata is returned in webhook payloads. The assessment read both payload builders and confirmed the documentation is stale, so there is no exfiltration channel here.
- **No actionable internal target was identified in this repository.** The application's own management surface exposes only health and information endpoints, and everything beyond those is denied. Whether Excom's wider network offers an endpoint that acts on a bare JSON POST is outside the scope of a source review.

Finding 4 of 7 · Low · CWE-639 · OWASP A01:2025 Broken Access Control

`F-04 · engine reference ca365fab114c`

Share creation checks that a report exists but never checks who owns it, so a tenant can publish a link over another tenant's report

SEVERITY	STANDARD OF EVIDENCE
LOW	Partially traced: the missing check is certain; reaching a victim identifier is not established
ROOT CAUSE `src/main/java/com/vaultdoc/api/v2/ShareController.java:50`, method `createShare`	ENTRY POINT `POST /api/v2/shares`, redeemed at `GET /api/v2/shares/access/{token}`
WHO CAN EXPLOIT IT An authenticated user in a different tenant holding `ROLE_REPORT_SHARE`	SECURITY PROPERTIES LOST Confidentiality and integrity of the tenant boundary, and privilege separation between tenants
INDEPENDENT OBSERVATIONS 34 observations across 31 grouped root causes, the second-largest concentration in the run	FRAMEWORK REFERENCES CWE-639 Authorization Bypass Through User-Controlled Key · OWASP A01:2025 · NIST SSDF PW.1.1, PW.5.1 (added by this report)

WHY A BROKEN TENANT BOUNDARY IS RATED LOW

The defect is real and the code is wrong: the tenant boundary is not enforced on this operation. The rating reflects what an attacker can do with it today. Exploitation requires the victim's report identifier, which is a random version 4 identifier with roughly 122 bits of entropy, so it cannot be guessed or enumerated. If it is obtained, what the share link discloses is the report title, the access level and the format. The stored object path is never returned, and no endpoint in this application serves report content. Excom should treat this as a defect to fix on schedule rather than an active exposure, and should re-rate it if either of the bounds below stops holding.

HOW THE DEFECT WORKS

The controller looks the requested report up by identifier alone, using an unscoped repository method. Finding a row establishes that the report exists; it says nothing about which tenant owns it. The caller's tenant, available on the authenticated principal, is never compared with the report's owning tenant. The service then writes a share row whose creating tenant is the attacker's, binding a foreign report to the attacker's tenant for the life of the link.

The same repository interface declares two tenant-scoped query methods, at `ReportRepository.java:12` and `:14–15`. Neither is a lookup by identifier, and the assessment found no production caller for either, so there is no in-repository precedent this path departed from. What makes the omission read as a defect rather than a design choice is that the caller's tenant is present on the principal at the point of the lookup and is simply not consulted.

PATH FROM REQUEST TO CROSS-TENANT EFFECT

1. Authenticated caller holding the share authority
   `api/v2/ShareController.java:44`

2. Report identifier taken from the request body with only a non-blank check
   model/ShareRequest.java:15
3. Unscoped lookup by identifier: existence verified, ownership never compared
   api/v2/ShareController.java:50
4. Share row persisted with the attacker's tenant recorded as creator
   service/ShareService.java:38–39
5. Redemption returns the victim tenant's report title and format to an anonymous caller
   service/ShareService.java:78–82

WHAT AN ATTACKER ACHIEVES

Holding another tenant's report identifier, the attacker mints a publicly redeemable link over that report, sets its recipient and access level, and can direct its webhook notifications to a destination of their choosing. The victim tenant has no visibility into the share, because the listing endpoint is scoped to the creating tenant, which is the attacker. The link survives for up to one week and there is no revocation endpoint.

WHERE THE IDENTIFIER COULD COME FROM

The assessment did not demonstrate a route to a victim identifier, and says so. Two routes were considered and left open as plausible: identifiers shared through legitimate business channels such as support tickets or forwarded links, and identifiers recovered by an attacker who already holds another position on the platform, including the code execution in F-01. Both are outside what a source review can establish.

REMEDIATION DIRECTION

Add a tenant-scoped lookup by identifier to `ReportRepository`, along the lines of `findByIdAndTenantId`, and use it here, so that a report belonging to another tenant is indistinguishable from one that does not exist. The repository's two existing tenant-scoped methods return lists by tenant and by template, so neither substitutes for this. Return the same response for the missing and the foreign case, which also removes the existence oracle described below. Apply the same review to every other lookup by caller-supplied identifier. Adding a share revocation endpoint and surfacing shares to the report's owning tenant would both limit the consequences if a similar defect recurs.

EVIDENCE

LOCATION	SOURCE	WHAT IT ESTABLISHES
api/v2/ShareController.java:50	`ReportEntity report = reportRepository .findById(request.getReportId()) .orElseThrow(() -> new IllegalArgumentException( "Report not found: " + request.getReportId()));`	The lookup is by identifier alone. Existence is checked, ownership is not. The caller's tenant is never compared against the report's tenant.

LOCATION	SOURCE	WHAT IT ESTABLISHES

api/v2/ShareController.java:44–47	`@PreAuthorize("hasAuthority('ROLE_REPORT_SHARE')")`
	`public ResponseEntity<ShareResponse> createShare(`
	`  @Valid @RequestBody ShareRequest request,`
	`  @AuthenticationPrincipal VaultDocPrincipal principal)`
		The caller's tenant is available on the principal at the point of the lookup, so the comparison is omitted rather than unavailable.

repository/ReportRepository.java:14	`@Query("SELECT r FROM ReportEntity r WHERE " +`
	`  "r.tenantId = :tenantId AND r.templateId = :templateId")`
	`List<ReportEntity> findByTenantAndTemplate(...)`
		Tenant-scoped queries already exist on this repository, though no production caller invokes either of them. Both return lists by tenant, so neither is a substitute for a scoped lookup by identifier.

service/ShareService.java:38–39	`entity.setReportId(request.getReportId());` `entity.setCreatedByTenantId(principal.getTenantId());`	The share row binds a report the attacker does not own to the attacker's own tenant, completing the cross-tenant effect.

service/ShareService.java:78–82	`return Map.of(`
	`  "reportTitle", report.getTitle(),`
	`  "accessLevel", share.getAccessLevel().name(),`
	`  "format", report.getOutputFormat() != null`
	`  ? report.getOutputFormat() : "pdf"`
	`);`
		Redemption discloses title, access level and format of the victim tenant's report to an unauthenticated caller. It bounds the disclosure to exactly these fields.

config/jwt/VaultDocPrincipal.java:11–13	`private final String userId;`
	`private final String tenantId;`
	`private final List<String> roles;`
		Tenant identity originates in the verified token claims, so the attacker cannot simply assert the victim's tenant instead.

service/ReportService.java:89	`String reportId = UUID.randomUUID().toString();`	Report identifiers are random version 4 UUIDs carrying roughly 122 bits of entropy, which is the control that holds this finding at low severity. The entity declares the same generator at `ReportEntity.java:15–17`, and share tokens are generated the same way at `ShareService.java:103–104`.

ASSESSMENT BOUNDS

- **Identifier enumeration was considered and ruled out from the source.** The distinct error text on a missing report is an existence oracle, but against roughly 122 bits of entropy it offers no practical search-space reduction. The attacker must already hold the identifier.
- **Report content is not exposed.** The stored object path is never included in any response, and the assessment found no endpoint anywhere in the application that serves report bytes. The `DOWNLOAD` access level is therefore inert today. It stops being inert the moment a download endpoint is added.
- **No code path in this repository writes a report record.** Reports are stored to object storage, but the assessment found no path that persists the database row this lookup reads. Either the records arrive from a system outside this repository or the share path cannot currently resolve any report at all. This is stated as unresolved, not assumed either way.
- **Cross-tenant read of report bodies is not claimed.** Disclosure is limited to the three fields cited above.

Finding 5 of 7 · Low · CWE-770 · OWASP A06:2025 Insecure Design

F-05 · engine reference `f31341d2bdaf`

An unauthenticated endpoint submits work to a small shared thread pool with no throttle, denying notifications to other tenants

SEVERITY	STANDARD OF EVIDENCE
LOW	Fully traced: pool bounds, the unauthenticated trigger and the failure mode are each cited
ROOT CAUSE	ENTRY POINT
`src/main/java/com/vaultdoc/service/ShareService.java:72`, method `accessShare`	`GET /api/v2/shares/access/{token}`, which requires no credentials
WHO CAN EXPLOIT IT	SECURITY PROPERTIES LOST
Anyone holding a valid share token, including a recipient it was legitimately sent to	Availability of webhook notification across all tenants, and the accuracy of the response other tenants receive when creating or redeeming a share
INDEPENDENT OBSERVATIONS	FRAMEWORK REFERENCES
1	CWE-770 Allocation of Resources Without Limits or Throttling · OWASP A06:2025 · NIST SSDF PW.9.1 (added by this report)

PRIORITY REASONING

The degraded function is webhook notification. Report generation runs on a separate executor and is untouched, and a share that carries no webhook destination redeems normally throughout. The rating reflects a bounded loss of a secondary feature, with a spillover that does reach other tenants: once the pool starts rejecting, a share creation or redemption carrying a webhook destination returns a server error even though the database work has already been committed.

HOW THE DEFECT WORKS

Every redemption of a share that carries a webhook destination submits a notification task to a shared executor. Nothing bounds how often that happens: no per-share cap, no per-token rate limit, no access-count ceiling and no deduplication. The executor is sized at two core threads, four maximum and a fifty-slot queue, and every tenant's webhook deliveries run through that one pool. No custom rejection policy is configured, so the default applies and rejected submissions throw.

Two properties combine to make this reachable. The trigger route is anonymous by design, so no credential limits how often it can be hit. And the asynchronous calls are not wrapped, so when the executor rejects a submission the exception propagates on the calling request thread rather than being absorbed.

PATH FROM REQUEST TO EXHAUSTION

1. Attacker registers a share whose webhook destination accepts connections and never answers
   `service/ShareService.java:43`

2. Redemption route requires no credentials
   `config/SecurityConfig.java:30`
3. Each redemption submits a task unconditionally, with no cap or throttle
   `service/ShareService.java:72`
4. Shared pool: 2 core, 4 maximum, 50 queued, default rejection policy
   `config/AsyncConfig.java:28–30`
5. Rejection propagates into other tenants' requests as a server error
   `service/ShareService.java:54`

WHAT AN ATTACKER ACHIEVES

Fifty-four concurrent redemptions fill the four worker threads and the fifty-slot queue. A destination that accepts the connection and then never answers holds a worker for the ten-second read timeout. One that returns a byte at a time holds it for as long as it keeps trickling, because that timeout governs each individual read rather than the whole exchange. Beyond that, submissions are rejected.

The consequences reach past the attacker's own tenant. Other tenants' share creations throw after the row has been committed, producing a share that exists with no notification sent and a server error returned to a caller whose request in fact succeeded. Other tenants' share recipients receive an error instead of report metadata. Notifications lost during saturation are not retried or queued anywhere, so they are gone.

REMEDIATION DIRECTION

Set an explicit rejection policy on the sharing executor that logs and drops rather than throwing, so a saturated pool cannot fail a request whose database work has already committed. Webhook delivery already runs on its own executor, so what is missing is isolation between that pool's rejection behaviour and the request thread, not a second pool. Add a per-share access-count cap and a per-token rate limit on the anonymous route. Persisting undelivered notifications for retry would remove the silent loss.

EVIDENCE

LOCATION	SOURCE	WHAT IT ESTABLISHES

`service/ShareService.java:71–73`	`if (share.getWebhookUrl() != null) {`
	
	`webhookService.notifyShareAccessed(share);`
	`}`
		Every redemption of a share carrying a webhook destination submits a task, with no cap, throttle or deduplication.

`config/SecurityConfig.java:30`	`.antMatchers("/api/v2/shares/access/**").permitAll()`	The trigger requires no credentials, so no account limits the request rate.

`config/AsyncConfig.java:28–31`	`executor.setCorePoolSize(2);`
	`executor.setMaxPoolSize(4);`
	`executor.setQueueCapacity(50);`
	`executor.setThreadNamePrefix("share-webhook-");`
		One small pool carries every tenant's webhook deliveries, and no rejection policy is set, so the default throwing policy applies.

LOCATION	SOURCE	WHAT IT ESTABLISHES
config/WebhookRestTemplateConfig.java:20–23	`.setConnectTimeout(Duration.ofSeconds(5))` `.setReadTimeout(Duration.ofSeconds(10))`	A destination that stalls after accepting holds a worker for ten seconds, and one that trickles holds it for as long as it keeps trickling, because the read timeout applies to each read.

service/ShareService.java:53–54	`if (request.getWebhookUrl() != null`
	`&&`
	`!request.getWebhookUrl().isBlank())`
	`{`
	
	`webhookService.notifyShareCreated(entity);`
		The submission sits between the save at line 48 and the response at line 57, and nothing catches a rejection, so a rejected task turns another tenant's committed creation into a server error.

ASSESSMENT BOUNDS

- **The attacker needs a valid share token to start.** Tokens carry roughly 122 bits of entropy, so obtaining one means creating a share, which requires authentication, or receiving one legitimately. A recipient can do this without any account.
- **Report generation is unaffected.** It runs on a separate executor, and any share created without a webhook destination redeems normally throughout. This is what holds the severity at low.
- **The effect is transient.** The pool recovers once the stalled deliveries time out. Notifications lost during the window are not recoverable.

Finding 6 of 7 · Low · CWE-770 · OWASP A06:2025 Insecure Design

`F-06 · engine reference a453c355d704`

An unconstrained metadata field is stored verbatim and fully rewritten on every read of the share

SEVERITY	STANDARD OF EVIDENCE
LOW	Partially traced: the missing limits are certain; the volume required to cause harm was not established
ROOT CAUSE `src/main/java/com/vaultdoc/model/ShareRequest.java:41`, field `metadata`	ENTRY POINT `POST /api/v2/shares`, amplified at `GET /api/v2/shares/access/{token}`
WHO CAN EXPLOIT IT An authenticated user holding `ROLE_REPORT_SHARE`	SECURITY PROPERTIES LOST Availability, through storage growth and write amplification
INDEPENDENT OBSERVATIONS 1	FRAMEWORK REFERENCES CWE-770 Allocation of Resources Without Limits or Throttling · OWASP A06:2025 · NIST SSDF PW.5.1 (added by this report)

PRIORITY REASONING

Every element of the mechanism is present in the code and cited. What the assessment did not establish is the volume needed to matter, which depends on Excom's database sizing, connection limits and storage headroom. The finding is recorded so that the missing limits are on the record before the feature reaches general availability, not as an active exposure. The application's own source already tracks the related gap: a comment on the report controller notes that rate limiting is required before general availability and names the internal ticket.

HOW THE DEFECT WORKS

The metadata field is the only property on the share request with no validation annotation. It has no size constraint, no key-count limit, no nesting-depth limit and no per-value length limit, and no request body size cap is configured at the container. It is written to a JSON binary column declared with no length attribute, and the entity mapping adds none.

The read path is where the cost multiplies. The custom column handler parses the entire stored document into a map on every read, and the write path re-serializes the whole document. Because a redemption also increments an access counter on the same row, each anonymous redemption forces a full-row update that rewrites the whole stored value, whether or not anything in it changed.

PATH FROM REQUEST TO AMPLIFIED COST

1. Request validated, with no constraint covering this field
   `api/v2/ShareController.java:46`
2. The one DTO property with no size, depth or count limit
   `model/ShareRequest.java:41`
3. Persisted verbatim into a JSON binary column with no declared length
   `service/ShareService.java:46`

4. Full parse on every read and full re-serialize on every write
   repository/JsonbType.java:65–68

5. Access-counter increment forces a whole-row update per anonymous redemption
   service/ShareService.java:68–69

WHAT AN ATTACKER ACHIEVES

An authenticated caller stores an arbitrarily large document per share, then drives repeated parse-and-rewrite cycles over it through an endpoint that needs no credentials. Cost accrues in three places at once: stored volume, database write throughput and the heap used to hold each parsed copy. Because the rewrite is driven by the anonymous route, the caller who pays for the storage is not the party who drives the amplification.

REMEDIATION DIRECTION

Constrain the field: cap the serialized size, the key count and the nesting depth, and reject anything larger at the validation boundary. Set a request body size limit at the container as a backstop. Move the access counter out of the row that carries the document, or increment it with a targeted update that does not rewrite the JSON column. Close out the rate-limiting work the source comment already tracks before this feature reaches general availability.

EVIDENCE

LOCATION	SOURCE	WHAT IT ESTABLISHES
model/ShareRequest.java:41	`private Map<String, Object> metadata;`	The only request property carrying no validation annotation, so no size, depth or count is bounded at the entry point.

repository/ShareEntity.java:48–50	`@Column(name = "metadata", columnDefinition = "jsonb")`
	`@Type(type = "jsonb")`
	`private Map<String, Object> metadata;`
		The column is declared with no length attribute, so the database imposes no practical ceiling either.

repository/JsonbType.java:65–68	`String json = rs.getString(names[0]);`
	`if (json == null || json.isBlank()) return null;`
	`try { return objectMapper.readValue(json, Map.class);`
		The whole document is parsed into memory on every read, which is the amplification factor.

service/ShareService.java:68–69	`share.setAccessedCount( share.getAccessedCount() + 1);` `shareRepository.save(share);`	An anonymous redemption forces a full-row update, rewriting the entire stored document even though only a counter changed.

LOCATION	SOURCE	WHAT IT ESTABLISHES
`api/v2/ReportController.java:19`	`// TODO: add rate limiting before GA (CLEAR-1902)`	The absence of a request rate limit is a known, tracked gap in the application's own source, which is why this finding is held at low rather than speculated into a higher severity.

ASSESSMENT BOUNDS

- **No exploitation threshold was established.** The volume needed to degrade service depends on Excom's database sizing and storage headroom, which a source review cannot determine.
- **Authentication is required to store the payload.** Only the amplifying read is anonymous.
- **This is distinct from F-02.** Both concern the same field. F-02 is about which classes the document can name; this finding is about how large it can be and how often it is rewritten. Fixing one does not fix the other.

Finding 7 of 7 · Informational · CWE-502 · OWASP A08:2025 Software or Data Integrity Failures

`F-07 · engine reference 5abe3890b27f`

A process-wide mutable static is writable through a public setter on a class the deserializer is permitted to instantiate

SEVERITY	STANDARD OF EVIDENCE
INFORMATIONAL	Partially traced: the decisive step cannot be settled from this repository
ROOT CAUSE	ENTRY POINT
`src/main/java/com/vaultdoc/repository/JsonbType.java:36–39`, method `setObjectMapper`	Any request carrying a JSON body
WHO CAN EXPLOIT IT	SECURITY PROPERTIES LOST
Any authenticated user, if the unresolved step permits it	Availability, if reachable
INDEPENDENT OBSERVATIONS	FRAMEWORK REFERENCES
1	CWE-502 Deserialization of Untrusted Data · OWASP A08:2025 · NIST SSDF PW.5.1 (added by this report)

PRIORITY REASONING

The assessment could not settle whether this is exploitable, and it says so rather than choosing a side. It is included because the design property is a real one that Excom can act on regardless of the verdict, and because the unresolved step is itself worth recording: settling it needs a dependency's internals, which a source review of this repository cannot reach. Excom should read this as a code-quality item with a security consequence if the surrounding conditions change.

HOW THE DESIGN PROPERTY WORKS

The custom column handler holds its JSON mapper in a mutable process-wide static field. Spring performs the intended single assignment at startup, through an `@Autowired` instance setter. That setter is public and unguarded, so any code holding an instance of the type can write the static again. Every read and write of the stored metadata column then dereferences it with no null check. Setting it to null, or replacing it with a differently configured mapper, would affect every tenant in the process at once.

The connection to an attacker is the deserializer covered in F-02. Its permitted set includes the application's own package tree, and this class lives inside it, so the class name is admissible in a request document. The setter is an ordinary instance method rather than a static one, which is what makes an invocation by a deserializer conceivable at all. Whether that admissibility turns into an actual call is the step the assessment could not resolve.

PATH, WITH THE UNRESOLVED STEP MARKED

1. Authenticated request carrying a JSON body
   `config/jwt/JwtAuthenticationFilter.java:41–59`
2. The class name passes the package-prefix rule and is admissible as a type identifier
   `config/ShareJacksonConfig.java:37`
3. **Unresolved:** whether the deserializer builds a usable binding for this class, or aborts while resolving it
   `jackson-databind internals, not in this repository`

4. Public setter writes the process-wide static, affecting every tenant
   repository/JsonbType.java:36–39

5. All consumers dereference the static with no null check
   repository/JsonbType.java:68

WHY THE STEP COULD NOT BE SETTLED

The library that would decide it is not vendored into the repository, so its behaviour cannot be read. What the source does fix is the shape of the target: a public no-argument constructor at line 33, and exactly one public single-argument setter, whose parameter is Jackson's own `ObjectMapper`. Whether the deserializer builds a usable binding for that signature, and what it does with a value supplied for that parameter, is decided inside the library. The assessment recorded a lean toward not exploitable but no library evidence behind it, so this report states the step as open rather than carrying that lean forward as a conclusion.

REMEDIATION DIRECTION

Remove the public setter, so that no path reachable from a request can reassign the mapper after startup. Marking the field `final` is not a drop-in substitute, because it is static and written from an instance method: the assignment has to move to a static initialiser or to a Spring-managed holder the type reads from. Fixing F-02 removes the reachability argument, which is the stronger reason to treat F-02 as the priority of the two.

EVIDENCE

LOCATION	SOURCE	WHAT IT ESTABLISHES

repository/JsonbType.java:31, 36–39	`private static ObjectMapper objectMapper;`
	
	`@Autowired`
	`public void setObjectMapper(`
	
	`@Qualifier("shareObjectMapper")`
	`ObjectMapper mapper) {`
	`JsonbType.objectMapper = mapper;`
	`}`
		A process-wide mutable static is written by an ordinary public instance setter with no guard, so a second call replaces the mapper for every tenant in the process. Spring makes the intended single assignment at startup through the annotation, so nothing requires the setter to stay public at runtime.

repository/JsonbType.java:68	`return objectMapper.readValue(json, Map.class);`	Consumers dereference the static without a null check, so nulling it degrades all reads of the stored metadata column.
config/ShareJacksonConfig.java:37	`.allowIfSubType("com.vaultdoc")`	The prefix rule admits the application's own package tree, including this class, which is what connects the design property to a request-reachable path.

ASSESSMENT BOUNDS

- **The decisive step is unresolved.** The assessment leaned toward not exploitable and recorded no library evidence behind that lean, so the step is stated here as open. Settling it means reading how jackson-databind binds this setter signature.
- **Nothing here was executed.** No test was run and no payload was delivered. This finding rests entirely on source reading.
- **It is contingent on F-02.** Remove global default typing and the reachability argument disappears, leaving a code-quality item.

7. Vulnerability classes assessed without adverse finding

A findings list shows what an assessment found. It does not show what the assessment looked for. This section closes that gap. It documents fifteen weakness classes the assessment worked on and did not report, and for each one it records the code that prompted the investigation, the work performed, and the control or condition that closed it.

These are not near-misses and they are not deferred issues. Each is a line of enquiry that ran to a conclusion. Excom can read this section as the record of review depth behind the seven reported findings, and as a register of the controls the assessment confirmed to be working. Several entries here are the reason a finding in Section 6 is rated lower than its weakness class would suggest.

Two provenance tags appear on the entries below. Each entry is scoped to a specific path through the code rather than to a whole weakness class, so the same weakness identifier can appear both here and on a finding in Section 6: CWE-502 and CWE-917 each do.

EXAMINED AND CLEARED	Investigators formed and tested one or more attack hypotheses in this line of enquiry and closed every one against a control, an unreachable step or a missing precondition. Nothing in this line of enquiry was ever put forward as a candidate finding.
REJECTED AT VERIFICATION	An investigator did put a candidate finding forward in this class, and the verification stage rejected it on independent re-derivation from the source. Four of the fifteen entries fall here. They are the clearest evidence that verification is doing work rather than confirming what was handed to it.

7.1 Remote code execution through the polymorphic deserializer

CWE-502 · OWASP A08:2025 · EXAMINED AND CLEARED

Risk signal. The share endpoint's JSON mapper enables polymorphic default typing for every non-final type, which is the configuration behind most published remote-code-execution chains in this library. Finding it made code execution the working assumption rather than a possibility to be checked.

Agentic investigation. Investigators enumerated the known gadget families that this configuration normally unlocks, covering the database-driver, dependency-injection, connection-pool, logging and collections routes, and tested each against the mapper's type validator. They then abandoned library gadgets and searched the application's own package tree for a class whose construction or property assignment does something dangerous on its own.

Investigation outcome. Every gadget family the investigators enumerated lives under a package name the validator's three prefix rules exclude, and the rules are applied on the read path before instantiation. Inside the application's own package tree, one class does expose a setter with a process-wide side effect, and it is reported separately as F-07; no class was found whose construction or setters were shown to reach code execution. The result is not that the mapper is safe: it is that the traced ceiling is class instantiation and denial of service. That is why F-02 is rated high on traced impact rather than critical on the impact its weakness class implies.

7.2 JavaScript execution through the formula evaluator

CWE-94 · OWASP A05:2025 · EXAMINED AND CLEARED

Risk signal. A service in the application holds a JavaScript engine and calls its evaluation method on a string. A script engine evaluating a string is among the strongest signals available in a Java codebase, and this one drew more investigative effort than any other line of enquiry in the run: thirty-one separate hypotheses were recorded against it.

Agentic investigation. Investigators tried to construct a payload that satisfies the input filter while still reaching the engine, attempting Unicode escapes, numeric and octal encodings, whitespace and comment tricks, expressions built only from arithmetic characters, and prototype access through bracket notation. They then traced the method's callers through the whole repository, and checked which script engine the pinned Java version actually provides.

Investigation outcome. Four independent controls each close it on their own. The input filter is a full-string match against a pattern that admits only digits, arithmetic operators, parentheses, the decimal point and whitespace, so no identifier, quote, bracket or property access can survive it. A two-hundred-character cap applies before the filter. No production code calls the method at all: the only caller in the repository is its own unit test. And the build pins a Java version in which requesting this engine by name returns nothing, so the field is null and any call would fail before evaluating.

> The last two controls are worth separating. A method with no production caller and a null engine would not be exploitable even if the filter were removed. An assessment that reported this on the strength of the engine call alone would have produced a false positive with a convincing-looking code excerpt attached.

7.3 Expression injection on the version 1 report path

CWE-917 · OWASP A05:2025 · EXAMINED AND CLEARED

Risk signal. The version 1 report endpoint reaches the same expression evaluator that carries F-01, the critical finding. Two endpoints sharing one dangerous sink is the pattern that usually means both are exploitable, and treating them as one issue would have been the easy conclusion.

Agentic investigation. Investigators traced the version 1 path separately from the version 2 path, established where its expression text originates, and then tested whether caller-supplied data reaching the evaluator by a different route could still be evaluated as code.

Investigation outcome. On the version 1 path the expression text is a compile-time constant, drawn from a fixed enumeration in the source with no runtime way to add an entry. Caller data does reach the evaluator, but as named variables bound into the evaluation context rather than as expression text, and the expression language resolves a variable's value without re-parsing it. The two paths share a sink and differ in what feeds it. F-01 goes further and uses the version 1 path as its contrast, noting that the authority gating the vulnerable endpoint is the same ordinary permission that gates this benign one.

7.4 Server-side template injection through the rendered layout

CWE-1336 · OWASP A05:2025 · EXAMINED AND CLEARED

Risk signal. The template engine performs placeholder substitution into a layout string and then renders it. Substituting attacker-influenced values into a template that is subsequently interpreted is the standard shape of a template injection, and ten hypotheses were recorded against it.

Agentic investigation. Investigators worked on two routes. The first was second-order: place expression syntax inside a substituted value and get it interpreted on a later pass. The second was direct: find any way for a caller to influence the layout string itself, whether through a request field, a stored record or a file path.

Investigation outcome. The control that closes the second-order route is the single pass: the substitution loop scans only the original layout and writes its output to a separate buffer, so text that a substitution inserts is never re-examined by the scanner. The inserted value is passed through the platform's replacement-quoting helper first, which is narrower protection than it looks, since it prevents the value being read as a replacement-group reference and does not neutralise expression syntax. The direct route closes because the layout string has exactly one source: a hardcoded literal selected by output format, with no caller influence over which literal is chosen or what it contains. The engine installs no bean resolver either, so an injected expression could not reach application beans by name, although a bare evaluation context still permits type references.

7.5 Stored, second-order expression injection through the database

CWE-917 · OWASP A05:2025 · EXAMINED AND CLEARED

Risk signal. A database migration creates a table with a column that stores computed-field definitions, the same kind of data that F-01 shows is evaluated as code. A stored expression column suggests a persistent version of the critical finding, one that would survive a fix to the request path.

Agentic investigation. Investigators searched the entire repository for anything that reads that column: an entity mapping it to a field, a repository method selecting it, a native query, a direct database access call, or a migration that seeds it with data.

Investigation outcome. The column has no consumer anywhere in the application. No entity maps it, no query selects it, and nothing writes to it. The table exists in the schema without any code path that reads or populates it. A stored expression that nothing ever evaluates is not a vulnerability, so the enquiry closed on the absence of a consumer rather than on a control. Excom should note that this conclusion depends on that absence continuing: code that starts reading the column would create the second-order path this enquiry ruled out.

7.6 SQL and query-language injection

CWE-89 · OWASP A05:2025 · **EXAMINED AND CLEARED**

Risk signal. The application declares a custom repository query alongside caller-supplied identifiers and format strings, and it holds several tenant-scoped lookups. Any of those is a candidate for query injection, and the class is checked on every assessment regardless of what the survey stage surfaces.

Agentic investigation. Investigators read every declared query in the repository layer and then swept the whole source tree for the other ways a query can be built: query creation calls, entity-manager use, direct template access, native queries and string concatenation into query text.

Investigation outcome. The one custom query uses named bind parameters throughout, which the persistence layer sends as parameters rather than as text. The sweep for every other query-construction route returned nothing across the whole tree. All remaining data access goes through generated repository methods, which parameterise by construction. There is no place in this application where caller data becomes query text.

7.7 Token signature bypass and algorithm confusion

CWE-347 · OWASP A04:2025 · **EXAMINED AND CLEARED**

Risk signal. Authentication, tenant identity and every authority in this application come from claims inside a signed token. If the signature can be bypassed, every other control in the system follows, so the assessment treated this as the highest-value target after the report endpoints. Eighteen hypotheses were recorded against it.

Agentic investigation. Investigators worked through the standard bypass repertoire: a token declaring no algorithm, an unsigned token in place of a signed one, an asymmetric algorithm submitted against a symmetric verification key, a key too short to be secure, and a verification path that treats a parse failure as success.

Investigation outcome. The filter calls the parser method that requires a signed token, so an unsigned token is rejected on shape before any claim is read. The library version in use rejects a token declaring no algorithm and refuses to verify an asymmetric algorithm using a symmetric key. The key-construction helper throws if the configured secret is shorter than the algorithm requires, and the key is built inside the filter on each request, so a short secret fails verification rather than weakening it. And the filter's exception handler clears the security context on any failure, after which the configuration's final rule denies every unmatched request. The path fails closed at each step.

7.8 **Tokens and claims honoured after expiry** CWE-613 · OWASP A07:2025 · **REJECTED AT VERIFICATION**

Risk signal. The authentication filter validates a signature and reads claims, but never requires that an expiry claim be present, never checks the issuer or audience, and consults no revocation list. An investigator put this forward as a finding: a token with no expiry claim would be accepted indefinitely, and a compromised token could not be withdrawn.

Agentic investigation. The verification stage re-derived the claim from the source without reference to the investigator's reasoning, then attacked it. The re-derivation confirmed the source facts exactly: no expiry requirement, no issuer or audience check, no token identifier and no revocation mechanism exist in the code.

Investigation outcome. The attack step does not hold. Every token the filter accepts must carry a valid signature, and producing a token with no expiry claim means signing one, which requires the secret. An attacker with the secret has full impersonation of any user, tenant and role, so the missing expiry requirement adds nothing to their position. The revocation gap is a separate matter and it is not closed here: what a leaked token costs before it lapses depends on the lifetime the issuing system sets, and the issuer is outside this repository. The absent controls are real and they are defence-in-depth gaps, and Excom may still choose to add them. On the standard this report applies, which is whether an attacker gains something they do not already have, the claim failed. Engine group reference `c-11cb10e683`.

7. Vulnerability classes assessed without adverse finding (continued)

7.9 A hardcoded signing secret reaching a production runtime

CWE-798 · OWASP A07:2025 · REJECTED AT VERIFICATION

Risk signal. Three files in the repository carry a literal token signing secret: a development profile configuration, the continuous-integration workflow and the example environment file. Anyone with read access to the repository can see them, and a signing secret is the one value in this application that yields impersonation of any user in any tenant. An investigator put the development profile value forward as a hardcoded credential.

Agentic investigation. The verification stage asked the question the finding depends on: can that value ever be the secret a running instance uses? It traced every place the development profile is activated, and then read how the base configuration resolves the secret when no profile is active.

Investigation outcome. The only activation of that profile anywhere in the repository is a local development target in the build file. The base configuration resolves the secret from an environment variable with no fallback value, so an instance started without that variable set fails at startup instead of falling back to a committed literal. No production runtime in this repository resolves to one. The continuous-integration workflow does set that environment variable to a committed literal, so pipeline instances run on a known secret; the assessment treated that as a test runtime rather than a production credential. Whether to keep any literal secret in version control is a policy question this report does not settle, and Excom may want to confirm that the pipeline runtime never holds data worth impersonating a user for. Engine group reference `c-bd31752e27`.

7.10 Forged report provenance by overwriting trusted context values

CWE-915 · OWASP A08:2025 · REJECTED AT VERIFICATION

Risk signal. The report service copies the caller's entire parameter map into the same context that already holds trusted values: the generation timestamp, the tenant identifier and the identity of the user who generated the report. The trusted values are placed first and the caller's map is copied over them under matching key names, so a caller who supplies those keys replaces values the application intends to control. An investigator put this forward as provenance forgery, with the tenant identifier as the concerning case.

Agentic investigation. The verification stage accepted that the overwrite happens, and then looked for a consumer that would act on the polluted values. It traced every read of the context after the copy: what the renderer uses, what the storage layer uses to compute a location, and whether any authorization or tenant decision reads from it.

Investigation outcome. The polluted context has no consumer that crosses an authority boundary. Rendering does read it: the layout's one placeholder resolves the generation timestamp from the context, so a caller who supplies that key changes the timestamp printed on their own report and nothing else. The storage location is computed from the tenant on the verified principal, not from the context, so a forged tenant value does not move the stored object or reach another tenant's prefix. The overwrite is real and it is a code-quality problem, since caller data and application data share one namespace. It produces no security consequence in this codebase. It would become one if a future change made any decision read the tenant from the context. Engine group reference `c-d1060dc57e`.

7.11 Class instantiation at the stored-data boundary

CWE-502 · OWASP A08:2025 · **REJECTED AT VERIFICATION**

Risk signal. The same polymorphic mapper that carries F-02 also reads the stored metadata column. Three separate candidate findings were put forward on one premise: if an attacker can write a class name into that column, the deserializer will instantiate it every time the row is read, which turns a request-time weakness into a stored one that persists and fires repeatedly.

Agentic investigation. The verification stage tested the premise rather than the consequence. It identified every writer to that column, checked whether any of them puts caller-supplied text into a position the deserializer reads as a class name, and looked for alternative write paths: native queries, direct column updates and seeded migration data.

Investigation outcome. The premise fails. The only writer serializes an in-memory object graph, so the class identifiers it emits come from the runtime types of objects already constructed in the process, not from attacker text. Nothing else in the repository writes the column, no native query touches it and no migration seeds it. Attacker-controlled data does reach the column, but not the type-identifier position. All three candidate findings were rejected on the same ground. The related effect that *is* reachable, storing a value that cannot be read back, is reported inside F-02. Engine group references `c–6695aa8132`, `c–b06ea777d6`, `c–5157fbf54b`.

7.12 Path traversal into the object-storage key

CWE-22 · OWASP A01:2025 · **EXAMINED AND CLEARED**

Risk signal. The storage layer builds an object key by string formatting, inserting a tenant identifier, a report identifier and an output format directly into the path. String-formatted paths are a standard traversal candidate, and the tenant segment is the only thing separating one customer's stored reports from another's.

Agentic investigation. Investigators examined each of the three inserted values for attacker control, and specifically tested whether the format value could carry path separators or parent-directory sequences on either the version 1 or the version 2 request path.

Investigation outcome. The format field carries a strict pattern constraint that permits exactly two literal values, and the constraint is declared on both request types and enforced on both controllers. No traversal character can pass it. The tenant segment comes from the verified token claim rather than from the request, so the caller cannot assert another tenant's prefix. The report identifier is generated server-side. All three positions are closed, and they are closed by validation that is actually enforced, which is the distinction that matters here: F-01 exists precisely because a constraint on a neighbouring field was declared in a position where it does not apply.

7.13 Unauthenticated reach beyond the public share route

CWE-862 · OWASP A01:2025 · EXAMINED AND CLEARED

Risk signal. The security configuration grants anonymous access to a path prefix. An open prefix in a configuration that otherwise requires authentication is worth testing on its own, and here it matters more than usual, because if anonymous access extended even slightly further it would reach the endpoints that carry F-01 and F-02. Six hypotheses were recorded against it.

Agentic investigation. Investigators checked what the pattern actually matches, whether path variations such as encoded separators or trailing segments could widen it, whether any request handler outside the intended one falls under it, whether method-level authorization is switched on, and where the authentication filter sits relative to the framework's context-handling filters.

Investigation outcome. The anonymous pattern resolves to a single request handler. Everything else under the interface prefix requires authentication, and the configuration's final rule denies every request that matches nothing, so an unmapped path is refused rather than allowed. Method-level authorization is enabled and both sharing operations declare a required authority. The authentication filter is registered after the framework's context-handling filter, which is the correct order for the security context to be populated when the handler runs. The anonymous route is a deliberate capability-link design in which the share token is the credential, and that token's strength is the subject of the next entry.

7.14 Share-token prediction and enumeration

CWE-340 · OWASP A04:2025 · EXAMINED AND CLEARED

Risk signal. The share token is the only credential protecting an anonymous endpoint that returns another party's report metadata. If it can be predicted or enumerated, the anonymous route in the previous entry stops being a deliberate design and becomes an open door. Eleven hypotheses were recorded against it, and the token's strength is load-bearing for the severity of F-04 as well.

Agentic investigation. Investigators identified the generator and its underlying source of randomness, calculated the effective entropy after the post-processing the code applies, examined the lookup for any partial or prefix matching, and tested whether the endpoint's distinct error messages leak enough to narrow a search.

Investigation outcome. Tokens are generated by the platform's random identifier facility, which draws on a cryptographically secure source, giving roughly one hundred and twenty-two bits of unpredictability. The code strips separator characters for formatting, which changes the token's appearance without removing any entropy. The column is unique and the lookup is an exact match, so no prefix or partial search is possible. The endpoint does return different messages for an unknown token and an expired one, which is an existence oracle, but an oracle that confirms one guess at a time offers no meaningful progress against a space that size.

7.15 An exploitable vulnerable third-party component

CWE-1395 · OWASP A03:2025 · **EXAMINED AND CLEARED**

Risk signal. Two declared dependencies were taken up on the strength of published advisories against their declared versions: a text-processing library with a known interpolation vulnerability, and a spreadsheet library. A version match against an advisory is what most scanning tools report as a finding on its own, so the assessment treated it as a hypothesis to be tested rather than a result.

Agentic investigation. Investigators searched the whole source tree for the specific classes and methods each advisory requires, probing the text-processing library's interpolation entry points four separate times under different naming patterns, and searching for any call into the spreadsheet library's document-writing interface.

Investigation outcome. Neither vulnerable code path is reached from this application. Not one of the interpolation entry points appears anywhere in the repository, and no call to the spreadsheet library's writing interface exists either, despite the application declaring a spreadsheet output format. Both are unreachable vulnerable dependencies. That is a real supply-chain item for Excom's dependency management to resolve on its own schedule, and it is not an application defect, so this report does not raise it as a finding. Section 9 records that no software bill of materials was supplied to the assessment, which bounds this conclusion to the two components examined rather than to the dependency tree as a whole.

WHAT THIS SECTION ESTABLISHES

Fifteen lines of enquiry ran to a conclusion and were not reported. Eleven closed without any candidate finding being put forward: nine against controls the assessment traced and found effective, which Section 8 collects, and two on the absence of a consumer for the code in question rather than on a control. Four were candidate findings that verification rejected on re-derivation from the source, covering six rejected root-cause groups, which is the part of the process a findings list alone does not show.

The rejected four also show where the assessment draws its line. In each case the underlying source observation was accurate: the missing expiry requirement, the committed development secret, the shared context namespace and the storage-boundary deserializer all exist as described. What failed was the step connecting the observation to something an attacker gains. Excom may reasonably decide to address some of them anyway as defence in depth. This report separates that decision from the question of whether they are exploitable today.

8. Security controls confirmed present and working

Section 7 closed nine lines of enquiry against controls that are present in the application and working. This section collects those controls in one place, so that Excom has a register of what is currently protecting the application and what would be lost if any of it were removed or refactored.

Each entry names the control, where it sits, and the attack path it closes. These are not assertions that the control is well designed in general, and no control was exercised against a running instance. Each is a record that the assessment traced an attack path to this control in the source and could not get past it.

8.1 Controls traced and found effective

CONTROL	LOCATION	ATTACK PATH MITIGATED
Signed-token parsing with algorithm and key-length enforcement	`JwtAuthenticationFilter.java:41–42`	Unsigned tokens, tokens declaring no algorithm, asymmetric algorithms submitted against the symmetric key, and secrets too short for the algorithm. A secret too short for the algorithm is rejected where the key is built, on each request, so verification fails rather than weakening. See 7.7.
Fail-closed authentication error handling	`JwtAuthenticationFilter.java:63`	A parse or verification failure clears the security context instead of leaving a partly populated one behind, so a malformed token cannot arrive at a handler as an authenticated request.
Deny by default on unmatched requests	`SecurityConfig.java:32`	Reach to any path the configuration does not explicitly permit, including paths added later without a matching rule. See 7.13.
Method-level authorization on both sharing operations	`ShareController.java:44` , `:65`	Share creation and listing by an authenticated user who lacks the sharing authority.
Secret resolution with no fallback value	`application.yml:39`	An instance starting with no signing secret configured and silently falling back to a committed development literal. There is no fallback to fall back to, so an unset variable fails startup. Read this with 7.9, which records that the continuous-integration workflow sets the variable to a committed literal: pipeline instances run on a known secret by explicit configuration, not by fallback.
Tenant identity taken only from verified claims	`VaultDocPrincipal.java:11–13`	A caller asserting another tenant in a request field. This is also what stops the storage-key traversal in 7.12 and bounds the impact of F-04.
Named bind parameters on all data access	`ReportRepository.java:14` and generated repository methods	SQL and query-language injection across the whole application. The sweep for other query-construction routes across the tree returned none. See 7.6.
Output-format allow-list, enforced on both request types	`ReportRequest.java:18` , `CustomReportRequest.java:24`	Path traversal and separator injection into the object-storage key. See 7.12.
Cryptographically strong share tokens with exact-match lookup	`ShareService.java:104` , `ShareEntity.java:42`	Prediction and enumeration of share links. This control is load-bearing: it is what makes the anonymous route in 7.13 a deliberate design rather than an open door. What holds F-04 at low severity is the identifier on the other side of that finding, the report identifier generated by the same means at `ReportService.java:89` . See 7.14.

CONTROL	LOCATION	ATTACK PATH MITIGATED
Deserialization type validator	`ShareJacksonConfig.java:34–39`	Every gadget family the investigators enumerated for this library. Recorded with the qualification below, because the configuration it guards is itself F-02. See 7.1.
Anchored input filter and length cap on the formula evaluator	`FormulaService.java:43–49`	Script injection into a JavaScript engine, backed by two further conditions: no production caller and no engine present under the pinned platform version. See 7.2.
Single-pass substitution over the template body	`TemplateEngine.java:50–58`	Second-order template injection. The scanner walks the body once and appends each resolved value to a separate buffer, so a value that itself contains placeholder syntax is never scanned again. The escaping applied to that value is regex-replacement escaping, which is a narrower guarantee than it appears: see 7.4.
Compile-time expression definitions on the version 1 path	`TemplateDefinition.java:20–41`	Expression injection through the version 1 report endpoint, which shares a sink with F-01. See 7.3.
Generic error responses	`GlobalExceptionHandler.java:50–56`	Disclosure of stack traces, internal paths and dependency detail. Every message the assessment traced is a fixed string or an echo of the caller's own input.
Tenant-scoped share listing	`ShareService.java:85–86`	One tenant reading another tenant's shares. Note that this is also the query through which F-02's stored availability effect propagates.
Connect and read timeouts on outbound requests	`WebhookRestTemplateConfig.java:20–23`	Indefinite retention of a worker thread by an unresponsive destination. It bounds the cost of F-03 and F-05 without preventing either.

8.2 Controls to track that hold under current conditions

Four protections in this application work because of something that is currently true rather than because of a check that enforces it. Each one is recorded here with the condition it depends on, so that a future change does not remove it silently.

WHAT HOLDS TODAY	THE CONDITION IT DEPENDS ON
The `DOWNLOAD` access level on a share discloses nothing beyond the `VIEW` level.	No endpoint in the application serves report content. The distinction between the two access levels is enforced by the absence of a feature, not by a check. Adding a download endpoint gives the access level meaning it does not currently have, and at that point F-04 stops being a metadata disclosure.
The stored computed-field definitions cannot become a second-order injection.	Nothing reads that column. The safety is the absence of a consumer. Code that begins reading it would create the persistent version of F-01 that 7.5 ruled out.
Overwriting trusted values in the report context has no security consequence.	No authorization or tenant decision reads from that context. If a future change makes any decision read the tenant from the context rather than from the principal, the shared namespace described in 7.10 becomes exploitable.
The deserialization validator blocks the gadget families that matter.	The allowed set is three package-name prefixes, matched by prefix rather than by exact type. A new dependency contributing classes under one of those prefixes, or a new application class with a side-effecting constructor or setter, changes the answer without anyone editing the validator. This is the strongest argument for fixing F-02 at the configuration rather than relying on the validator.

MONITORING ENDPOINT EXPOSURE AS CONFIGURED

Four Actuator endpoints are enabled at `application.yml:34`: `health`, `info`, `prometheus` and `metrics`. The security configuration names only the first two. `SecurityConfig.java:29` permits `/actuator/health` and `/actuator/info` anonymously, so both answer without a token. `/actuator/prometheus` and `/actuator/metrics` match no rule ahead of `anyRequest().denyAll()` at `SecurityConfig.java:32`, so both are refused to every caller, authenticated or not. Neither half is raised as a finding: the two anonymous endpoints return liveness and build data only, and the two denied endpoints return nothing to anyone. The consequence Excom may not intend is operational, in that a Prometheus scraper pointed at this service receives 403.

9. Limitations of this assessment

This section states what the assessment did not cover and where its own execution fell short. It is written to be read alongside the coverage statement in Section 5, and it is deliberately specific: a limitation that names the file, the class or the step is one Excom can act on.

9.1 Agentic whitebox discovery

Source review, not testing	Every conclusion in this report comes from reading the application's source and configuration. No instance of the application was deployed, no request was sent, and no payload was executed. Where this report says a finding is **Fully traced**, it means every step is established from cited source with no unresolved gap. It does not mean an exploit was run.
Point-in-time	The assessment covers the state of the repository as reviewed on 2026-08-27. It says nothing about any commit after that point.
No runtime or deployment context	Network segmentation, egress filtering, web application firewall rules, container privileges, database permissions and secret management were not observed. Several findings note where a deployment control would change their impact. None assumes one exists.
Application code only	Third-party library internals were not read except where a dependency was vendored. Two findings, F-06 and F-07, record a step that could not be settled for this reason.

9.2 Weakness classes not examined

The coverage map in section 5.4 lists every class the assessment worked on. The following classes do not appear there because no hypothesis was recorded against them. **Absence from the coverage map is not clearance.** Excom should not read this report as evidence that the application is free of these weaknesses.

Cross-site scripting	Open redirect	XML external entity processing
Operating-system command execution reached directly from request data	Directory-service query injection	Log injection and log forging
Regular-expression denial of service	Insecure file upload	Cross-origin resource sharing misconfiguration
Missing or weak security response headers	Web and proxy cache poisoning	HTTP request smuggling
Archive extraction path traversal	Prototype pollution	Session fixation
Mail and header injection	Cross-site request forgery	

On the last item: the survey did record that the security configuration disables the framework's cross-site request forgery protection at `SecurityConfig.java:25` and that the session policy is stateless. Nobody carried that observation forward into a hypothesis, so the class sits here rather than in the coverage map and this report offers no view on whether the combination suits Excom's interface. On operating-system command execution: the code-execution impact recorded in F-01 arrives through expression evaluation. No search was made for a process-execution call site reached directly from request data. One class that might be expected in this list is absent from it on purpose: over-binding of a request body was examined, and F-06 is the result, though only on the one property where the binding is unconstrained.

9.3 Artefacts inside the repository but outside coverage

ARTEFACT	CONSEQUENCE FOR THIS REPORT
The migration set, measured against the entities the application maps	Both Flyway migrations were read. Neither creates the `document_shares` table that `ShareEntity.java:14` maps, and no code in the reviewed tree persists a report record. With `ddl-auto: validate` configured at `application.yml:13`, an instance built from this tree alone would not start. The consequence for this report is specific: F-03, F-04, F-05 and F-06 are established from the source as written, and each depends on the sharing tables and report rows being supplied by something outside this artefact. Excom should read those four findings as applying to the deployed system that does have them.
Continuous integration workflow definition	Opened only for the secret values it sets, which is how 7.9 reached the committed continuous-integration literal. How the pipeline is triggered, what it publishes, and what other credentials it holds are outside this assessment.
Container and local-orchestration definitions, and the environment-variable example file	Read in part. The container definition and the environment-variable example were opened and cited for the variables they name, and neither was assessed as a hardening question. Container privileges, base image, exposed ports and the handling of credentials at deployment time are outside this assessment. F-01's impact analysis assumes only what the application source shows about the process it runs in.
Shell scripts under the repository's script directory	Not examined for embedded credentials or unsafe operations.
Test sources	Read only to establish whether a production caller exists for a given method, which is how 7.2 reached its conclusion. Not assessed as an attack surface and not assessed for embedded credentials.
`VaultDocApplication.java`	The one Java file the investigation record never names. It is the framework bootstrap class and contains no security logic, but this report cannot claim it was reviewed.

9.4 Guidance for interpreting results

Where this report reports no finding in a class, the claim is narrow and specific: investigators formed attack hypotheses in that class, tested them against the source, and closed each one against a control, an unreachable step or a missing precondition, with the closing reason recorded. Section 7 states those reasons so that Excom and its reviewers can judge them rather than accept them.

Two things that claim does not cover. It does not extend to the classes named in section 9.2, where no hypothesis was formed at all. And it does not survive a change to the code: nine of the fifteen clearances in Section 7 rest on a control remaining in place, and two more rest on the absence of a consumer for the code in question rather than on any control at all. Section 8.2 collects four conditions of that second kind, drawn from both Section 7 and the findings, because those are the ones a routine change is most likely to remove without anyone noticing.

Appendix A. Severity and path-evidence definitions

Two independent judgements are recorded on every finding in this report. Severity states how much harm the established impact represents. Path evidence states how completely the path to that impact was traced in the source. Reading them together is what makes the ratings in Section 6 interpretable.

A.1 Severity

RATING	DEFINITION AS APPLIED IN THIS REPORT	ASSIGNED TO
CRITICAL	An attacker holding no more than ordinary authenticated access takes control of the application process, or reaches the data or identity of parties other than themselves, with every step established.	F-01
HIGH	Substantial loss of integrity or availability affecting parties other than the attacker, established from entry point to effect, without control of the process.	F-02
MEDIUM	The attacker gains a capability they should not have, with a real consequence, where either the reach of that capability or the information returned by it is constrained.	F-03
LOW	The defect is established in the code, but exploitation depends on something the attacker is not shown to possess, or the function it degrades is secondary and recovers.	F-04, F-05, F-06
INFORMATIONAL	A design property with a security consequence that the assessment could not establish as reachable, recorded so that it is on the register if the surrounding conditions change.	F-07

A.2 Path evidence

Fully traced	Every step from an entry point to the stated effect is established from cited source, with no unresolved gap in the path. Applied to F-01, F-02, F-03 and F-05. A fully traced path can still carry a precondition on the shape of the request that the assessment did not construct; where that is so, the precondition is stated in the finding's **Assessment bounds** block. This is a statement about the completeness of the source analysis. No instance was deployed and no payload was executed, as stated in section 9.1.
Partially traced	At least one step depends on a precondition the assessment could not establish. Applied to F-04, F-06 and F-07. In each case the unresolved step is named inside the finding, in the **Assessment bounds** block.

A.3 The rule that governs both

Severity is assigned on what the analysis traced in the source, not on the typical impact of the weakness class.

No instance of the application was deployed and no payload was delivered, so nothing in this report is a demonstrated exploit. Each rating states how far the reasoning got from cited source and stops there. The rule has visible consequences in Section 6. F-02 is a deserialization defect rated high rather than critical, because the analysis looked for a path to code execution and found none it could trace. F-04 is a tenant-boundary failure rated low, because the step that would reach another tenant needs an identifier with roughly 122 bits of entropy that no route in the source discloses. F-07 is recorded at informational because its decisive step cannot be settled from this repository.

The tradeoff is that a rating describes what is traceable today, not what is reachable tomorrow. Excom should read the assessment bounds on each finding alongside its rating, and re-rate a finding if any bound stops holding. Section 8.2 lists the four cases where the bound is a condition rather than a check, which are the ones most likely to move.

Appendix B. Framework coverage mapping

> Every mapping in this appendix is supplied by this report, not by the assessment engine. The engine records a CWE identifier on each finding; the OWASP category assignments, the NIST SSDF practice references and the coverage judgements in table B.2 are editorial additions made when this report was prepared. They are offered as a reading aid for readers working from those frameworks, and Excom should treat them as such.

B.1 CWE references

REF	CWE	WEAKNESS NAME
F-01	CWE-917	Improper Neutralization of Special Elements used in an Expression Language Statement (Expression Language Injection)
F-02	CWE-502	Deserialization of Untrusted Data
F-03	CWE-918	Server-Side Request Forgery (SSRF)
F-04	CWE-639	Authorization Bypass Through User-Controlled Key
F-05	CWE-770	Allocation of Resources Without Limits or Throttling
F-06	CWE-770	Allocation of Resources Without Limits or Throttling
F-07	CWE-502	Deserialization of Untrusted Data

Section 5.4 lists a further nineteen CWE identifiers that the assessment worked on without issuing a finding. Section 7 accounts for eleven of them, one to an entry. Its four remaining entries revisit an identifier a finding already carries, which is why the same identifier can appear in both places. The eight identifiers Section 7 does not reach are disposed of in the outcome column of section 5.4 itself.

B.2 OWASP Top 10 2025 coverage

This table states what the assessment covered in each category, not whether the application passes it. A category marked partially covered means some of what the category describes was examined and some was not, with the uncovered part named.

CATEGORY	COVERAGE	WHAT WAS EXAMINED, AND WHAT WAS NOT
A01 Broken Access Control	**Examined**	Tenant-boundary enforcement on every data-access path, request-level and method-level authorization rules, the anonymous route, traversal into storage keys, and every outbound request construction together with the destination validation applied to each. Produced F-04 and F-03; cleared 7.12 and 7.13. Server-side request forgery had its own category in the 2021 edition and sits inside this one in 2025, which is why F-03 is recorded here.
A02 Security Misconfiguration	Partially covered	Examined: the security filter chain, application and profile configuration, serialization configuration, executor sizing and management-endpoint exposure. Read only for the variables they set: the continuous integration workflow, the container definition and the environment-variable example. Not examined: security response headers, cross-origin resource sharing policy and external entity processing. See sections 9.2 and 9.3.

CATEGORY	COVERAGE	WHAT WAS EXAMINED, AND WHAT WAS NOT
A03 Software Supply Chain Failures	Partially covered	Two declared dependencies carrying published advisories were tested for reachability from application code, and both were found unreachable. See 7.15. No software bill of materials was supplied and the dependency tree was not inventoried, so this covers those two components only. The 2025 edition widens this category beyond the vulnerable-components question it replaces, to the integrity of the build, publication and distribution chain. None of that was examined.
A04 Cryptographic Failures	Partially covered	Examined: signing algorithm and key-length enforcement, signature verification, signing-key sourcing, and the randomness and entropy of the identifiers the application generates. Cleared 7.7 and 7.14. Not examined: transport security configuration, cipher selection, and encryption of stored reports, all of which are deployment properties not visible in this source.
A05 Injection	**Examined**	The most heavily worked category in the run. Expression language, script engine, template, stored second-order and query-language injection were all covered. Produced F-01; cleared 7.2 through 7.6. Not examined: cross-site scripting, and operating-system command execution reached directly from request data.
A06 Insecure Design	**Examined**	Resource limits and throttling on the sharing paths, the capability-link sharing design, and a time-of-check to time-of-use question on the access counter. Produced F-05 and F-06. Two lines of enquiry that began as design questions in this category appear elsewhere in this table, under the category their weakness identifier belongs to: token strength at A04 and the shared context namespace at A08.
A07 Authentication Failures	**Examined**	Signature verification, algorithm confusion, key strength, expiry and revocation handling, and signing-secret sourcing. Rejected 7.8 and 7.9 at verification. The signature-forgery work is recorded at A04, where CWE-347 sits. Not examined: session fixation.
A08 Software or Data Integrity Failures	**Examined**	Polymorphic deserialization at both the request boundary and the stored-data boundary, mutable process-wide state, and overwriting of trusted values in a shared context. Produced F-02 and F-07; cleared 7.1; rejected 7.10 and 7.11 at verification.
A09 Security Logging and Alerting Failures	NOT EXAMINED	No hypothesis was recorded in this category, and neither logging nor alerting adequacy was assessed. Two incidental observations are on the record and are not conclusions: failed webhook deliveries and swallowed expression-evaluation errors are logged at warning level, and F-01 notes that probing the expression sink produces no error visible to the caller. Whether Excom's logging would detect any of the findings in this report, and whether anything raises an alert on them, was not assessed. Section 9.2 lists log injection and log forging among the classes with no hypothesis recorded against them.

CATEGORY	COVERAGE	WHAT WAS EXAMINED, AND WHAT WAS NOT
A10 Mishandling of Exceptional Conditions	Partially covered	New in the 2025 edition. Examined: the content of every reflected error message, the generic catch and swallowed per-entry exception on the expression path, the fail-open behaviour that follows from it, and parameter handling at the request boundary. Every reflected message was found to be a static string or an echo of the caller's own input, so nothing follows on that route; the swallowed exception is reported inside F-01 as an aggravating factor rather than as a defect in its own right. Coverage here is a by-product of other lines of enquiry rather than a category the assessment set out to work through, so it should not be read as a sweep of every error path in the application.

Two of the weakness identifiers this report tags to a category, CWE-770 on F-05 and F-06 and CWE-1336 on 7.4, do not appear on any category's mapped identifier list in the 2025 edition, and neither does CWE-208 or CWE-1236 in section 5.4. The category shown against each is this report's judgement of where the weakness belongs, not an OWASP mapping. Two structural changes in the 2025 edition also move work that a 2021 reading would place elsewhere: server-side request forgery is now inside A01, and the former vulnerable-and-outdated-components category has been absorbed into A03 Software Supply Chain Failures, so F-03 and 7.15 appear under different headings than before.

B.3 NIST Secure Software Development Framework references

Practice identifiers follow NIST Special Publication 800-218. The practice text is paraphrased.

PRACTICE	PRACTICE, IN BRIEF	WHAT THIS ASSESSMENT OBSERVED
PW.1.1	Use risk modelling to assess the security risk of the software.	F-04 shows the multi-tenant boundary enforced on some data-access paths and omitted on another in the same repository. The inconsistency, rather than the omission alone, is what suggests the boundary is not modelled as a requirement that every path must satisfy.
PW.1.3	Build in support for standardised security features and services rather than bespoke handling.	F-01: the framework provides a restricted evaluation context that removes exactly the capability being abused, and the application does not use it. F-03: no standardised destination policy or egress control mediates outbound requests.
PW.4.4	Verify that acquired third-party components meet security requirements.	Two declared components carry published advisories. Both were found unreachable from application code, so no defect follows. The absence of a software bill of materials means the assessment cannot speak to the rest of the dependency tree, which is itself relevant to this practice.
PW.5.1	Follow secure coding practices appropriate to the language and environment.	The practice most frequently implicated. F-01 evaluates untrusted strings as code; F-02 enables polymorphic typing globally; F-03 issues requests to unvalidated destinations; F-06 accepts unbounded input; F-07 exposes mutable process-wide state through a public setter.
PW.7.2	Review code against the organisation's secure coding standards.	Two defects behind F-01 are the kind a review catches by reading: a validation constraint declared on a map key where the value is the sensitive position, and a code comment asserting that expression syntax is validated on a path where it is not. Both survived into the reviewed state of the repository.

PRACTICE	PRACTICE, IN BRIEF	WHAT THIS ASSESSMENT OBSERVED
PW.9.1, PW.9.2	Define a secure baseline for every setting that has an effect on security, and implement those defaults.	F-02: polymorphic deserialization is enabled by configuration, guarded by three package-name prefixes described in the source as known-safe packages, which is broader than the description implies. F-05: the shared executor is sized at two core threads with the framework's default throwing rejection policy, and is reachable from an anonymous endpoint. No request body size limit is set anywhere.

Appendix C. Evidence manifest

Figures, citations and conclusions in this report trace to one of the artefacts below. They were written during the assessment and retained unmodified. A reader asking how a specific number in Section 5 was arrived at, or whether a specific hypothesis in Section 7 was really recorded, can be shown the artefact that carries it. Code excerpts are the one exception: the artefacts record the file and line, and the excerpt text in this report is quoted from the repository at that location.

ARTEFACT	WHAT IT CONTAINS	WHAT IN THIS REPORT DEPENDS ON IT
Survey record	Every code region selected for investigation, the target it was given, its completion status, and the candidate weaknesses it produced.	The 60-region figure in section 5.3. The observation that regions dispatched at the formula evaluator and the two advisory-bearing dependencies produced no candidate in their assigned class.
Investigation record	All 338 recorded hypotheses. For each of the 140 carried forward, the input, path and operation reached. For each of the 198 ruled out, the hypothesis, the code location and the written closing reason.	All of Section 7. The per-class effort figures in section 5.4. The hypothesis counts quoted in individual Section 7 entries.
Consolidation record	The 109 distinct root causes, the candidates absorbed into each, and each one's location and weakness class.	The grouping figures in section 5.3. The reconciliation that identified the two root-cause groups the reporting stage did not place. The observation counts on each finding in Section 6.
Findings output	The seven issued findings with severity, weakness identifier, path-evidence standard, mechanism, path, impact analysis, cited evidence and recorded bounds.	All of Section 6: the mechanism, the traced path, the impact analysis, every assessment bound, and the file-and-line citation carried by each excerpt. The engine reference on each finding heading is its identifier in this file.
Execution log	Per-region investigation outcomes, per-root-cause verification verdicts, the output of the severity-assignment stage, and its warnings.	The verification outcome split of 99 upheld, 6 rejected and 4 unresolved in section 5.3. The provenance tags in Section 7 and the group references on the four rejected entries. The truncation warning recorded against the severity-assignment stage.
Full step trace	The step-by-step record of every investigation and verification unit, retained in a local database.	The source-coverage figures in section 5.1, including the identification of the one Java file the record never names. The five rejected source sweeps recorded during verification.
Configuration snapshot	Every setting the assessment resolved at start, with the source of each value.	The record that reachability analysis did not run and that no software bill of materials was supplied. The verification capacity figure in section 5.3.

C.1 Run identification

Assessment reference	`vda-ae7e35b47769`
Target	`vault-doc-api`, Java with Spring Boot, as at 2026-08-27
Executed	2026-08-27 21:10 to 2026-08-28 00:21, elapsed 3 hours 11 minutes
Completion state	Completed with no unrecovered errors, with two qualifications that belong alongside that. The severity-assignment stage reached its ceiling on how much it can emit in a single pass before it had placed two of the 109 verified root-cause groups, and recorded a truncation warning that does not surface as an error. Both dropped groups were checked afterwards against the consolidation records: one is the same expression-evaluation root cause as F-01, the other is the same missing ownership check as F-04, so no distinct weakness was lost on this occasion. Separately, five source sweeps during verification used a pattern the search tool would not compile and returned errors; each was re-run in working form within the same unit of work, but the pipeline registered every rejected call as a success, so the execution record reports no tool errors for a stage that had five.
Stages executed	Survey, Investigate, Consolidate, Verify, Report. Two optional stages did not run. Reachability analysis, which independently confirms whether a sink is reachable from an entry point, was not enabled, so the reachability conclusions in this report come from investigators tracing call paths by hand rather than from a second independent pass. The ingest path that accepts a supplied bill of materials or repository context had nothing configured, so dependency work was limited to reading the manifest and testing two specific advisories for reachability; 7.15 covers those two components and is not a dependency inventory.

Prepared for Excom from assessment `vda-ae7e35b47769`. Findings, severities, path-evidence standards, evidence citations, recorded bounds and remediation direction are reproduced from the assessment output. Framework mappings and the organisation of this document were added when the report was prepared, as stated in Section 1 and Appendix B.