# Penetration Testing Report

Fictional sample environment

**CONFIDENTIAL - SAMPLE ONLY**

Assessment type: Grey-box web application and API penetration test

Report version: 1.0 | Date: 10 July 2026

Prepared by: VAPT Experts

This document is an illustrative marketing sample. It contains no client information.

# Document Control

| Field | Value |
|---|---|
| Client | Fictional SaaS Company Ltd. |
| Assessment | Grey-box web application and API penetration test |
| Environment | Fictional staging environment |
| Testing window | 01-05 July 2026 |
| Report date | 10 July 2026 |
| Version | 1.0 - sample |
| Classification | Confidential - illustrative sample only |

## Important sample notice

### Fictional content

All organizations, users, domains, identifiers, requests, responses, screenshots and findings in this report are fictional or sanitized. The document demonstrates report structure and writing quality only.

## Distribution

- Client executive sponsor
- Engineering and security teams
- Authorized auditor or customer reviewer
- VAPT Experts engagement team

## Version history

| Version | Date | Author | Change |
|---|---|---|---|
| 0.9 | 08 July 2026 | VAPT Experts | Draft for quality review |
| 1.0 | 10 July 2026 | VAPT Experts | Final fictional sample |

# Contents


| | | |
|---|---|---|
| 1 | Executive Summary | 4 |
| 2 | Scope and Rules of Engagement | 5 |
| 3 | Methodology and Risk Rating | 6 |
| 4 | Findings Summary | 7 |
| 5 | Detailed Finding: Cross-Tenant Access | 8 |
| 6 | Detailed Finding: Session Lifecycle | 10 |
| 7 | Detailed Finding: Content Security Policy | 12 |
| 8 | Remediation Roadmap | 14 |
| 9 | Retest Summary | 15 |
| 10 | Positive Security Observations | 16 |
| 11 | Limitations and Conclusion | 17 |


### How to use this report

Executives should begin with the Executive Summary and Remediation Roadmap. Engineering teams should use the detailed findings and root-cause guidance. Auditors and customers should review scope, methodology, limitations and retest status.

# 1. Executive Summary

## Assessment objective

VAPT Experts performed a fictional grey-box security assessment of a multi-tenant SaaS web application and its supporting REST API. The objective was to identify exploitable weaknesses affecting customer data, account security and browser-side protection before an enterprise customer review.

## Overall conclusion

### Material risk identified

The sample assessment identified one high-severity authorization weakness, one medium-severity session-management weakness and one low-severity browser-policy gap. The high-severity issue could allow an authenticated user to access records belonging to another fictional tenant. Immediate remediation and focused retesting were recommended.

| Severity | Count | Interpretation |
|---|---|---|
| HIGH | 1 | Prompt remediation required due to cross-tenant data exposure risk. |
| MEDIUM | 1 | Control weakness increases account persistence risk after security-sensitive changes. |
| LOW | 1 | Defense-in-depth improvement to reduce browser-side exploitation impact. |

## Priority actions

- Enforce tenant ownership checks on every server-side object lookup, independent of client-supplied identifiers.
- Invalidate all active sessions following password reset, credential rotation and administrator-forced logout.
- Deploy a restrictive, tested Content Security Policy using nonces or hashes and remove unsafe inline dependencies.
- Add automated authorization regression tests for cross-tenant and cross-role access patterns.

# 2. Scope and Rules of Engagement

## In-scope assets

| Asset | Type | Access | Notes |
|---|---|---|---|
| https://app.example.invalid | Web application | Grey-box | Three fictional user roles |
| https://api.example.invalid/v1 | REST API | Grey-box | OpenAPI documentation supplied |
| Fictional tenant A and B | Test data | Authorized | Synthetic records only |

## Agreed access

- Standard user accounts in two separate fictional tenants
- One tenant administrator account
- API documentation and representative test data
- Staging environment approved for controlled validation

## Excluded activities

- Denial-of-service, stress testing and destructive data modification
- Social engineering, physical access and testing of third-party services
- Production systems, employee accounts and real customer information
- Persistence beyond the minimum needed to validate a finding

## Rules of engagement

Testing was performed only against the listed fictional assets during the agreed window. Potentially disruptive actions required explicit approval. Emergency contacts and stop conditions were defined before testing. This sample does not represent actual execution against a live system.

# 3. Methodology and Risk Rating

## Assessment workflow

| | | |
|---|---|---|
| 1 | **Scope** | Confirm assets, roles, constraints and evidence requirements. |
| 2 | **Map** | Identify routes, APIs, roles, tenants, objects and trust boundaries. |
| 3 | **Test** | Perform safe automated discovery and expert-led manual validation. |
| 4 | **Validate** | Reproduce, de-duplicate and assess context and impact. |
| 5 | **Report** | Document evidence, root cause, remediation and limitations. |
| 6 | **Retest** | Verify eligible fixes and update status. |

## Reference guidance

Coverage was informed by OWASP web and API testing guidance, common authorization and session attack patterns, and the agreed application context. Frameworks guide consistency but do not replace manual tester judgment.

## Severity model

| Rating | Meaning |
|---|---|
| CRITICAL | Direct, severe compromise with broad impact and limited prerequisites. |
| HIGH | Material confidentiality, integrity or privilege impact with realistic exploitation. |
| MEDIUM | Meaningful control weakness with additional prerequisites or narrower impact. |
| LOW | Limited impact or defense-in-depth weakness that should be improved. |
| INFO | Observation or improvement opportunity without a confirmed vulnerability. |

# 4. Findings Summary

| ID | Finding | Severity | Status |
|---|---|---|---|
| WEB-01 | Cross-tenant record access through missing object authorization | HIGH | Closed after retest |
| WEB-02 | Existing sessions remain valid after password reset | MEDIUM | Open |
| WEB-03 | Content Security Policy permits unsafe inline script execution | LOW | Partially remediated |

## Risk themes

- Server-side authorization must be applied to each requested object rather than inferred from the user interface.
- Security-sensitive account events should terminate or re-evaluate existing sessions.
- Browser security policy should reduce the impact of future injection defects and third-party script compromise.

## Finding status definitions

| | |
|---|---|
| **Open** | The issue remains reproducible or remediation has not been validated. |
| **Partially remediated** | Some risk was reduced, but the finding or a material component remains. |
| **Closed after retest** | The supplied remediation was tested and the original issue was no longer reproducible within retest scope. |

WEB-01 HIGH STATUS: Closed after retest

# Cross-Tenant Record Access Through Missing Object Authorization

**CVSS:** 8.1 **Affected:** Invoice API

## Executive impact

An authenticated user in one fictional customer tenant could retrieve an invoice record belonging to another tenant by changing a predictable object identifier. This could expose sensitive customer and transaction information across tenant boundaries.

## Technical description

The API accepted an invoice identifier and returned the associated record after confirming that the requester was authenticated. It did not consistently verify that the invoice belonged to the requester's tenant. The web interface hid other-tenant identifiers, but server-side access control could not rely on the interface.

## Prerequisites

- Valid low-privilege user account in any tenant
- Knowledge or discovery of another invoice identifier
- Network access to the documented API endpoint

## Evidence summary

The tester authenticated as a standard user in fictional tenant A, requested a known invoice identifier from tenant B and received a successful response containing tenant B metadata. The response below is intentionally sanitized.

```
GET /api/v1/invoices/inv_tenant_b_1042 HTTP/1.1 Host: api.example.invalid Authorization: Bearer
[REDACTED] HTTP/1.1 200 OK Content-Type: application/json { "invoice_id": "inv_tenant_b_1042",
"tenant_id": "tenant-b", "customer": "[REDACTED]", "amount": "[REDACTED]" }
```

The same request returned HTTP 403 after the remediation was deployed and retested.

# WEB-01 - Root Cause and Remediation

## Root cause

Object ownership was enforced in the user-interface query path but not in the shared API repository method. The API loaded the invoice by identifier, then serialized it without applying the authenticated tenant identifier to the database query or authorization policy.

## Recommended remediation

- Apply centralized object-level authorization to every invoice read, update and delete operation.
- Bind database queries to both the requested object identifier and the authenticated tenant identifier.
- Use deny-by-default authorization policies for all multi-tenant resources.
- Avoid exposing sequential or guessable identifiers as a substitute for authorization.
- Add automated tests that attempt cross-tenant access for every privileged object operation.
- Log and alert on repeated authorization failures without recording sensitive response data.

## Example secure pattern

```
invoice = repository.find_one( invoice_id=request.invoice_id, tenant_id=current_user.tenant_id ) if
invoice is None: return forbidden_or_not_found() authorize(current_user, "invoice:read", invoice)
```

## Retest result

### Closed after retest

The retest confirmed that tenant A could no longer retrieve tenant B invoice records through the original endpoint. Equivalent read and update operations in the agreed retest subset returned authorization failures. This result does not guarantee that every application object uses the corrected policy.

## Risk ownership

**Primary owner:** API engineering
**Supporting owners:** Application security, QA automation, product engineering
**Suggested priority:** Immediate

WEB-02 MEDIUM **STATUS: Open**

# Existing Sessions Remain Valid After Password Reset

**CVSS:** 6.4 **Affected:** Web and API sessions

## Executive impact

A user who changed a password or used the password-reset workflow did not invalidate previously issued sessions on other devices. If a session token had already been stolen, the attacker could remain authenticated after the legitimate user attempted to secure the account.

## Technical description

The application rotated the account password but did not increment the session version, revoke refresh tokens or terminate active server-side sessions. The observed session remained valid until its normal expiration time.

## Reproduction summary

- Sign in to the fictional account in browser A and capture a representative authenticated request.
- Complete the password-reset flow in browser B.
- Repeat the original authenticated request from browser A.
- Observe that the server continues to accept the pre-reset session.

## Evidence

```
POST /api/v1/auth/reset-password HTTP/1.1 Host: api.example.invalid Content-Type: application/json { "reset_token": "[REDACTED]", "new_password": "[REDACTED]" } HTTP/1.1 204 No Content --- existing session --- GET /api/v1/profile HTTP/1.1 Authorization: Bearer [PRE-RESET TOKEN] HTTP/1.1 200 OK
```

## Recommended remediation

- Invalidate all active sessions and refresh tokens after password reset, password change, account recovery or administrator-forced logout.
- Maintain a server-side session version or token revocation mechanism.
- Offer users a visible device/session management page and a global sign-out option.
- Require re-authentication for security-sensitive operations.
- Define short access-token lifetimes with secure refresh-token rotation where stateless tokens are necessary.

## Validation guidance

Retest should confirm invalidation across browser sessions, mobile sessions, refresh tokens, API keys where relevant and active sessions created before the security-sensitive event.

# WEB-02 - Engineering Notes

## Why password rotation alone is insufficient

Changing the password protects future password-based logins, but it does not automatically affect already-issued session identifiers or signed tokens. Session revocation must be part of the account-security event.

## Implementation options

| Pattern | Strengths | Considerations |
|---|---|---|
| Server-side session store | Immediate revocation and central visibility | Requires resilient state and cleanup |
| Session version on user record | Simple global invalidation check | Must be checked on each protected request |
| Short access token + rotating refresh token | Scales for APIs and mobile clients | Refresh-token storage and reuse detection are critical |
| Revocation list | Targeted token invalidation | Operational complexity and storage growth |

## Suggested acceptance criteria

- A session issued before password reset is rejected immediately after reset.
- Refresh tokens issued before reset cannot obtain new access tokens.
- The account owner receives a security notification for the reset event.
- Security logs record session invalidation without storing raw tokens.
- The same behavior applies to administrator-forced password resets.

## Current status

### Open in this fictional sample

The client acknowledged the issue and scheduled remediation for the next authentication release. No retest evidence was available at report finalization.

WEB-03

LOW

**STATUS: Partially remediated**

# Content Security Policy Permits Unsafe Inline Script Execution

**CVSS:** 3.7

**Affected:** Web application responses

## Executive impact

The browser policy allowed unsafe inline JavaScript. This does not create cross-site scripting by itself, but it weakens a defense that can reduce the impact of future injection defects or compromised third-party content.

## Observed policy

```
Content-Security-Policy: default-src 'self'; script-src 'self' 'unsafe-inline'
https://cdn.example.invalid; object-src 'none'; base-uri 'self'
```

## Technical description

The script-src directive permitted unsafe inline script execution and trusted a broad fictional CDN origin. The application also included legacy inline event handlers, preventing a stronger policy from being deployed without refactoring.

## Recommended remediation

- Remove unsafe-inline from script-src after migrating inline code to nonce- or hash-authorized scripts.
- Use unique cryptographically random nonces per response for required inline script blocks.
- Restrict third-party script origins to the minimum required hosts and paths where feasible.
- Add object-src none, base-uri self and frame-ancestors as appropriate to the deployment.
- Test the policy in report-only mode, collect violations safely and promote to enforcement after validation.

## Example target policy

```
Content-Security-Policy: default-src 'self'; script-src 'self' 'nonce-{RANDOM_PER_RESPONSE}'; object-src
'none'; base-uri 'self'; frame-ancestors 'none'; form-action 'self'; upgrade-insecure-requests
```

## Partial remediation

### Partially remediated

The broad CDN origin was removed and frame-ancestors was added. Unsafe inline script remained because legacy application components still required refactoring.

# WEB-03 - Deployment Plan

## Recommended staged approach

| Phase | Action | Success measure |
|---|---|---|
| 1. Inventory | Identify inline scripts, event handlers and third-party sources. | Complete script-source inventory |
| 2. Report only | Deploy nonce-based candidate policy in report-only mode. | Violations reviewed without user impact |
| 3. Refactor | Remove inline handlers and unnecessary third-party script dependencies. | No unsafe-inline requirement |
| 4. Enforce | Enable restrictive policy and monitor violations. | Policy enforced across priority routes |

## Operational considerations

- Avoid sending full sensitive URLs or request bodies to third-party CSP reporting services.
- Version and test the policy as application dependencies change.
- Coordinate CSP with Subresource Integrity for static third-party scripts where feasible.
- Use automated browser tests to catch broken functionality before enforcement changes.

## Residual risk

Even a strong Content Security Policy is a defense-in-depth control. It does not remove the need to prevent output encoding, template injection, DOM-based injection and unsafe third-party dependencies.

# 8. Remediation Roadmap

## Recommended sequence

| Priority | Timeframe | Action | Owner |
|---|---|---|---|
| 1 | Immediate | Deploy centralized tenant ownership checks and review equivalent object operations. | API engineering |
| 2 | Near term | Invalidate all sessions after password reset and add authentication regression tests. | Identity team |
| 3 | Planned | Refactor inline scripts and enforce a nonce-based Content Security Policy. | Frontend engineering |
| 4 | Ongoing | Add recurring authorization tests, secure-code review and release security checks. | Engineering + security |

## Root-cause themes

- Authorization logic was implemented inconsistently across application layers.
- Security-sensitive account events were not linked to session lifecycle controls.
- Legacy frontend patterns limited the ability to deploy stronger browser security policies.

## Program recommendations

- Create a reusable, deny-by-default authorization service and test matrix for all multi-tenant objects.
- Define a session-security standard covering password reset, MFA changes, account recovery and administrative actions.
- Add security acceptance criteria to product stories involving identity, roles, records and integrations.
- Track remediation through to retest rather than closing findings on implementation alone.

# 9. Retest Summary

## Retest scope

The fictional retest assessed the specific endpoints, workflows and fixes supplied by the client. It was not a full repeat of the original assessment and does not prove that equivalent issues are absent elsewhere.

| Finding | Original | Retest result | Status |
|---|---|---|---|
| WEB-01 Cross-tenant access | HIGH | Original and representative equivalent requests returned authorization failures. | CLOSED |
| WEB-02 Session lifecycle | MEDIUM | No fix supplied during the retest window. | OPEN |
| WEB-03 Content Security Policy | LOW | Third-party source reduced; unsafe-inline remains. | PARTIAL |

### Closure statement

WEB-01 was no longer reproducible using the original and agreed representative test cases. Closure applies only to the retested scope and environment.

## Evidence retained

The fictional sample assumes that retest evidence, timestamps and tester notes are retained according to the engagement's agreed retention period. Real engagements should define evidence retention and deletion before testing begins.

# 10. Positive Security Observations

Security assessments should record effective controls as well as weaknesses. The following fictional observations influenced risk and reduced unnecessary recommendations.

## Multi-factor authentication available

Privileged fictional users could enroll in multi-factor authentication.

## Secure cookie attributes

Observed session cookies used Secure, HttpOnly and an appropriate SameSite setting.

## Rate limiting on authentication

Repeated failed login attempts triggered increasing delay and temporary controls.

## Separate staging data

The assessment used synthetic records rather than real customer information.

## Centralized logging

Authentication and authorization events were recorded in a central platform, although alert tuning was outside scope.

### Why positive observations matter

Recording effective controls provides a balanced view, supports remediation prioritization and helps teams preserve working protections during change.

# 11. Limitations and Conclusion

## Limitations

- This is a fictional sample report and does not reflect a real assessment.
- A penetration test is a point-in-time evaluation of the agreed scope and available access.
- Absence of a reported finding does not prove absence of vulnerability.
- Third-party services, source code, production infrastructure, social engineering and denial-of-service were outside the fictional scope.
- Retesting was limited to supplied fixes and representative test cases.

## Conclusion

The fictional assessment identified a material tenant-authorization weakness that required immediate remediation, a session-lifecycle weakness that remained open and a browser-policy improvement that was partially addressed. The highest-risk issue was closed after focused retesting. The remaining work should be tracked through implementation and validation.

## Recommended next review

Repeat focused authorization testing after material changes to tenant, role, billing or record-management workflows. Reassess session security after the authentication fix and include browser-policy validation in the next application release review.

### Contact

For a scoped assessment, contact sales@vaptexperts.com. Do not send credentials, private reports or sensitive evidence until NDA and secure transfer method are agreed.

End of fictional sample report