CONFIDENTIAL — SAMPLE / DEMONSTRATION

API PENETRATION TESTING REPORT

REST & GraphQL Security Assessment

2026 Illustrative Sample Deliverable

Organization	OrionSphere Technologies, Inc. (Fictional)
Platform	OrionSphere Enterprise Platform (Fictional)
Prepared By	TMG Security
Assessment Type	API Penetration Test (REST & GraphQL)
Assessment Period	01 August 2026 – 31 August 2026
Report Date	15 September 2026
Classification	CONFIDENTIAL — SAMPLE / DEMONSTRATION

FICTIONAL SAMPLE REPORT

NOT AN ACTUAL CLIENT ASSESSMENT

All findings, evidence, and results are illustrative.

01 DOCUMENT CONTROL

Document Title	API Penetration Testing Report — OrionSphere Enterprise Platform (Fictional Sample)
Document ID	TMG-API-OST-2026-001
Version	1.0
Issue Date	15 September 2026
Assessment Period	01 August 2026 – 31 August 2026
Assessment Type	API Penetration Test (REST & GraphQL)
Client	OrionSphere Technologies, Inc. (Fictional)
Platform	OrionSphere Enterprise Platform (Fictional)
REST API	https://api.orion-example.com (Fictional, non-resolving)
GraphQL API	https://graphql.orion-example.com/graphql (Fictional, non-resolving)
Environment	Fictional staging / controlled assessment environment
Prepared By	TMG Security Offensive Security Team
Technical Lead	TMG Security — Senior API Security Tester (Fictional Engagement Role)
Reviewed By	TMG Security Quality Assurance Lead
Classification	CONFIDENTIAL — SAMPLE / DEMONSTRATION

Version History

Version	Date	Author	Summary
1.0	15 September 2026	TMG Security Offensive Security Team	Initial issue — fictional sample deliverable.

02 IMPORTANT DISCLAIMER

FICTIONAL SAMPLE REPORT — NOT AN ACTUAL CLIENT API PENETRATION TEST

This report has been produced solely for demonstration and portfolio purposes on behalf of TMG Security. It is intended to illustrate the structure, technical depth, and professional reporting standard of an API penetration testing deliverable covering both REST and GraphQL interfaces. To ensure there is no ambiguity regarding the nature of this document, TMG Security sets out the following clarifications:

- OrionSphere Technologies, Inc. is a fictional entity. It does not correspond to any real company, and any resemblance to an actual organization is purely coincidental.
- The OrionSphere Enterprise Platform is a fictional application. No real production system was built, deployed, or accessed.
- The REST API (https://api.orion-example.com) and GraphQL API (https://graphql.orion-example.com/graphql) referenced throughout this report are fictional, non-resolving placeholder domains used only for illustration.
- No real client was engaged and no real client API infrastructure was tested.
- No production environment, system, network, or cloud resource of any kind was accessed, scanned, or tested in the creation of this report.
- No real user data, personal information, or customer records were accessed, viewed, or processed.
- No real credentials — including cloud, application, API, or third-party credentials — were used or are disclosed anywhere in this report.
- All GraphQL schemas, queries, mutations, subscriptions, and type definitions shown in this report are synthetic and constructed for illustration only.
- All vulnerabilities, findings, risk ratings, CVSS scores, dates, and remediation statuses described in this report are fictional and were constructed for demonstration purposes.
- All Proofs of Concept (PoCs) are illustrative, synthetic, and non-destructive; none were executed against a real or live target, and none involved denial-of-service exploitation or destructive race-condition automation.
- All screenshots, request/response examples, and evidence panels are synthetic or simulated representations created specifically for this sample report.
- No real exploitation occurred at any point during the preparation of this document, and no actual customer impact occurred.
- This report does not represent an actual TMG Security client engagement and must not be relied upon as evidence of the security posture of any real organization.

Intended Use

This sample report may be shared with prospective clients, partners, and other interested parties to illustrate TMG Security's API security testing methodology, technical reporting standards, and evidence-presentation quality across both REST and GraphQL interfaces. It should not be relied upon for any compliance, legal, contractual, or procurement decision, and does not constitute security advice regarding any real system.

TABLE OF CONTENTS

03 EXECUTIVE SUMMARY

TMG Security has prepared this fictional, illustrative API Penetration Testing report to represent OrionSphere Technologies, Inc., a hypothetical U.S.-based enterprise SaaS / FinTech infrastructure company, and its fictional flagship platform, the OrionSphere Enterprise Platform. This section presents a fictional executive-level summary of the simulated assessment — covering both the REST API and the GraphQL API surface — prepared in the style and format TMG Security would use for an actual client engagement.

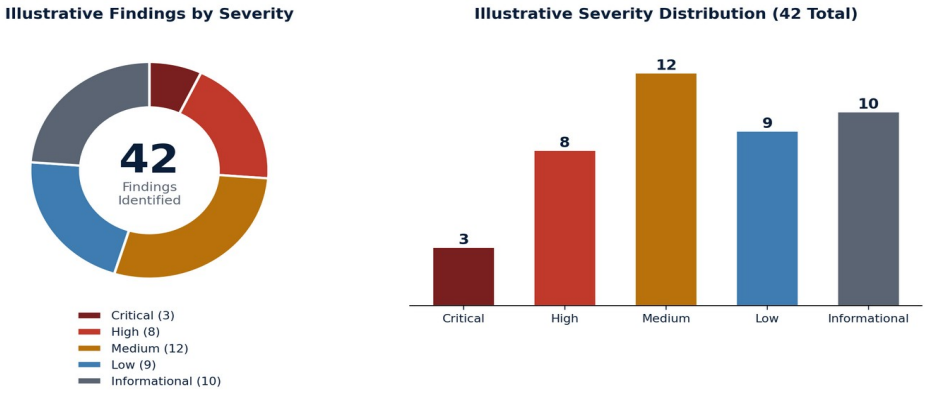
ALL METRICS, FINDINGS, AND SCORES ON THIS PAGE ARE ILLUSTRATIVE SAMPLE DATA

Overall Illustrative Security Posture

REQUIRES REMEDIATION

OrionSphere's simulated API surface demonstrates a number of established security controls — including MFA, refresh-token rotation, and TLS enforcement — but the fictional assessment identified material weaknesses concentrated in object-level authorization, GraphQL resolver authorization, excessive data exposure, business-logic enforcement, GraphQL mutation authorization, tenant isolation, and token/session controls. These require prioritized remediation before the platform should be considered ready for a broader trust posture. This illustrative status reflects a sample API penetration-testing exercise and does not represent a certification or compliance decision.

Fictional Sample Dashboard



Illustrative severity distribution across all 42 fictional findings — ILLUSTRATIVE SAMPLE DATA.

Executive Risk Narrative

The fictional assessment identified 42 illustrative findings across OrionSphere's REST and GraphQL API surface: 3 Critical, 8 High, 12 Medium, 9 Low, and 10 Informational. The three Critical findings concern the most fundamental trust-boundary failures observed: a cross-tenant broken object-level authorization (BOLA) gap reproducible through both REST and GraphQL interfaces, a GraphQL privileged-mutation authorization bypass allowing a lower-privileged role to invoke an administrative billing mutation, and a business-logic authorization bypass allowing unauthorized manipulation of invoice and refund state across tenant boundaries.

The eight High findings span stored cross-site scripting through API-supplied support content, an account-recovery weakness in the MFA recovery flow, a REST file-authorization bypass, excessive sensitive-data exposure through GraphQL

field resolvers, a GraphQL nested-resolver authorization bypass demonstrating that root-level authorization does not automatically propagate to nested fields, a legacy API v1 authorization weakness superseded but not retired by v2, server-side request forgery through the integration webhook-validation API, and a privilege-escalation path via an under-protected role-change mutation. Consistent with the pattern observed in the Critical findings, these High-severity issues concentrate around authorization enforcement — and in particular, the additional trust-boundary complexity that GraphQL's nested resolver model introduces relative to flat REST endpoints.

Medium-severity findings reflect defense-in-depth and GraphQL-specific hardening gaps — introspection exposing privileged operation names, query-complexity and query-depth enforcement gaps, batching/alias-based excessive object retrieval, rate-limiting coverage on authentication endpoints, token-revocation timing on logout, mass assignment across both REST and GraphQL update operations, webhook replay protection, CORS scope, verbose GraphQL error responses, API versioning/deprecation inconsistency, and a business-logic replay condition in the discount-redemption workflow. Low and Informational findings are hardening and governance opportunities — version disclosure, header coverage, schema metadata exposure, documentation accuracy, and monitoring/logging enhancements — that support a stronger long-term security posture without representing an immediate exploitation path in this fictional sample.

TMG Security's simulated recommendation is that OrionSphere prioritize closure of the three Critical and eight High findings within the first 60 days of the illustrative remediation roadmap (see Section 53), with particular attention to enforcing authorization consistently at the GraphQL resolver layer rather than only at the root query/mutation level. Medium findings should be addressed within 90 days, and Low/Informational items folded into the ongoing defense-in-depth and API-governance backlog. A full retest is recommended following remediation of all Critical and High findings, per the retest methodology in Section 55.

04 ENGAGEMENT OVERVIEW

This section describes the fictional objective, type, and approach of the illustrative OrionSphere Enterprise Platform API penetration test, prepared to demonstrate TMG Security's engagement-scoping and reporting standards for combined REST and GraphQL assessments.

Assessment Objective

The fictional objective of this engagement was to identify and validate exploitable security weaknesses across the OrionSphere Enterprise Platform's REST and GraphQL API surfaces — including authentication, authorization, object- and field-level access control, business logic, and GraphQL-specific trust-boundary enforcement — and to provide OrionSphere's engineering leadership with prioritized, actionable remediation guidance.

Assessment Type & Testing Window

This is a fictional API Penetration Test conducted against a controlled, fictional staging environment over the illustrative assessment period of 01 August 2026 – 31 August 2026, with this report issued 15 September 2026.

Testing Approach

- Authenticated testing across all seven fictional API roles (Visitor, Standard User, Billing User, Manager, Support Agent, Administrator, API Integration User).
- Unauthenticated testing of all publicly reachable fictional REST endpoints and GraphQL operations.
- Role-based testing designed to independently validate both horizontal and vertical authorization boundaries, across REST and GraphQL equivalents of the same underlying object.
- GraphQL-specific testing of schema discovery, resolver-level authorization, nested-query authorization propagation, query complexity/depth, batching/alias abuse, mutation security, and subscription security.
- Business-logic analysis of fictional multi-step workflows (subscription upgrades, discount redemption, invoice/refund state transitions, account-ownership transfer, approval workflows).
- Parallel testing of REST and GraphQL representations of the same underlying objects to identify inconsistent authorization enforcement between interfaces.

This engagement overview describes a fictional, illustrative assessment. No real client engagement occurred.

05 API ENVIRONMENT PROFILE

The OrionSphere Enterprise Platform is a fictional enterprise SaaS / FinTech infrastructure platform created to demonstrate a realistic API assessment scope, covering both a REST API and a GraphQL API. All organization, technology, and infrastructure detail below is illustrative.

Fictional Client Profile

Attribute	Detail
Organization	OrionSphere Technologies, Inc. (Fictional)
Industry	Enterprise SaaS / FinTech Infrastructure / Business Platforms
Headquarters	New York, New York, USA (Fictional)
Employees	Approximately 1,200 (Fictional)
Registered Customers	Approximately 4.2 million (Fictional)
Platform Name	OrionSphere Enterprise Platform (Fictional)
Primary REST API	https://api.orion-example.com (Fictional, non-resolving placeholder)
GraphQL API	https://graphql.orion-example.com/graphql (Fictional, non-resolving placeholder)
Environment	Fictional staging / controlled assessment environment

REST API Surface (Fictional, Illustrative)

The fictional REST API is versioned under /api/v1/ (with a subset of resources additionally available under /api/v2/), and exposes resource groups including authentication (auth/login, auth/refresh, auth/logout, auth/mfa), users, organizations and their members/roles, accounts, transactions, invoices, subscriptions, files, notifications, webhooks, reports, and platform administration (admin/users, admin/organizations). Full endpoint-level detail is provided in Section 12 and Appendix B.

GraphQL API Surface (Fictional, Illustrative)

The fictional GraphQL API is exposed at a single endpoint (/graphql) and defines schema areas for User, Organization, Member, Role, Account, Transaction, Invoice, Subscription, Payment, File, Notification, Webhook, AuditEvent, Report, and Admin. Representative operations include queries such as me, user, users, organization, organizations, member, members, account, accounts, transaction, transactions, invoice, invoices, subscription, subscriptions, files, notifications, auditEvents, and reports; mutations such as updateUser, updateOrganization, updateMemberRole, createInvoice, updateInvoice, cancelSubscription, changeSubscriptionPlan, createWebhook, updateWebhook, deleteWebhook, uploadFile, generateReport, and markNotificationRead; and subscriptions including transactionEvents, notificationEvents, and organizationEvents. Full operation-level detail is provided in Section 13 and Appendix C.

Fictional User / API Roles

#	Role	Illustrative Description
1	Visitor	Unauthenticated; public plan catalog, login, registration, password reset.
2	Standard User	Default authenticated role; personal profile, files, reports, notifications.
3	Billing User	Standard User privileges plus accounts, transactions, invoices, and subscription management.
4	Manager	Organization profile, member/role management, webhook configuration.
5	Support Agent	Internal role; report/ticket comment submission and customer-facing support content.
6	Administrator	Platform-wide administrative REST and GraphQL operations, cross-organization management.
7	API Integration User	Service-account / API-key-authenticated role for machine-to-machine integration.

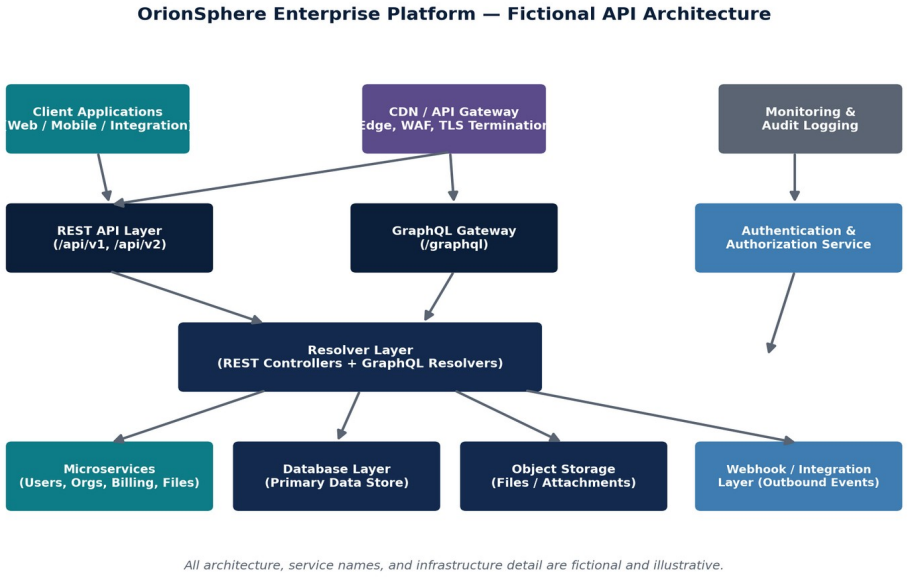
Fictional Technology Stack

Layer	Fictional Technology
REST Layer	Node.js / Express, versioned /api/v1 and /api/v2 routes
GraphQL Layer	Node.js GraphQL server with a resolver-per-field execution model
Database	PostgreSQL (primary data store)
Authentication	JWT (access token) + rotating refresh-token architecture; API-key auth for integrations
Infrastructure	AWS-style cloud environment (fictional)
CDN / API Gateway	Cloud-based edge protection and gateway routing (fictional)
Storage	Object-storage service (fictional)
Monitoring	Centralized logging and audit-event pipeline (fictional)

All infrastructure and technology detail above is fictional and provided only to give this sample report realistic structure and depth.

06 API ARCHITECTURE

The following fictional architecture diagram illustrates how client applications, the API gateway, the REST and GraphQL layers, the shared resolver layer, backing microservices, and data stores fit together in the OrionSphere Enterprise Platform. It is provided to give this sample report realistic technical grounding for the findings in later sections.



Illustrative fictional API architecture — all service names and infrastructure detail are fictional.

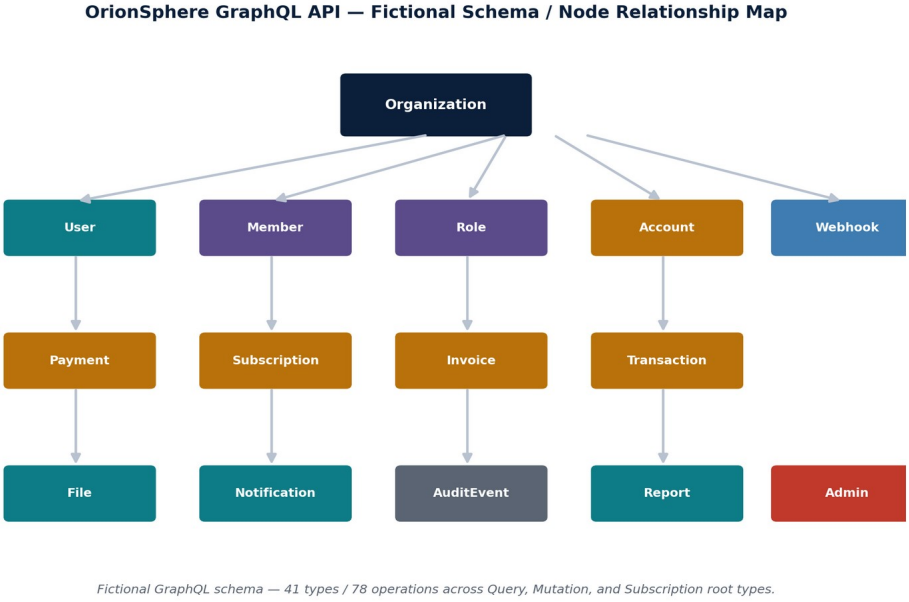
Architecture Notes

- Client Applications (web, mobile, and third-party integrations) reach the platform exclusively through the CDN / API Gateway, which terminates TLS and applies edge-level WAF rules.
- The REST API Layer and GraphQL Gateway are presented as parallel entry points that converge on a shared Resolver Layer (REST controllers and GraphQL field resolvers), which is the layer responsible for enforcing authorization — a point directly relevant to several findings in this report (notably API-002 and API-008).
- The Resolver Layer delegates to backing Microservices (users, organizations, billing, files), which in turn read and write the Database Layer and Object Storage.
- A dedicated Authentication & Authorization Service issues and validates tokens/API keys independently of the resolver layer; Monitoring & Audit Logging observes both REST and GraphQL traffic.

- The Webhook / Integration Layer delivers outbound events to customer-configured destinations and is the subject of the SSRF and webhook-security findings in Sections 10, 36, and Appendix E.

GraphQL Schema / Node Relationship Overview

The fictional GraphQL schema models 41 types connected through an Organization-centric graph. The diagram below shows the primary type relationships exercised during resolver-authorization and nested-query testing (Sections 13, 19, and 31).



Illustrative fictional GraphQL schema / node relationship map — all types and relationships are fictional.

07 TESTING SCOPE

The following fictional scope table defines the boundaries of this illustrative assessment.

In Scope

Area	Detail
REST API	All /api/v1/* and /api/v2/* endpoints supporting the OrionSphere platform (fictional)
GraphQL API	The full /graphql schema — Query, Mutation, and Subscription root types (fictional)
Authentication	Login, refresh, logout, MFA, MFA recovery, API-key issuance
Authorization	Role enforcement, object-, function-, and field-level access control, tenant isolation
GraphQL Resolver Security	Root and nested resolver authorization, schema introspection, query complexity/depth
Organization Management	Organization profile, member management, role assignment
Billing Workflows	Accounts, transactions, invoices, subscriptions, discount redemption
File / Object APIs	File upload, download, sharing, and metadata retrieval
Webhooks	Webhook configuration, destination validation, delivery, and replay protection
Reports & Notifications	Report generation/export, notification delivery and preferences
API Documentation	Published OpenAPI/Swagger and GraphQL documentation surfaces

Out of Scope

Area	Detail
Denial-of-Service Testing	Not performed under any circumstances, including GraphQL resource-exhaustion exploitation
Destructive Testing	Not performed under any circumstances, including destructive race-condition automation

Area	Detail
Social Engineering	Not included in this engagement
Physical Security	Not included in this engagement
Production Infrastructure	Testing restricted to the fictional staging environment only
Third-Party Systems	Not tested; only OrionSphere-owned fictional API surface in scope
Cloud Provider Internals	Underlying cloud-provider infrastructure and metadata services not tested

08 RULES OF ENGAGEMENT

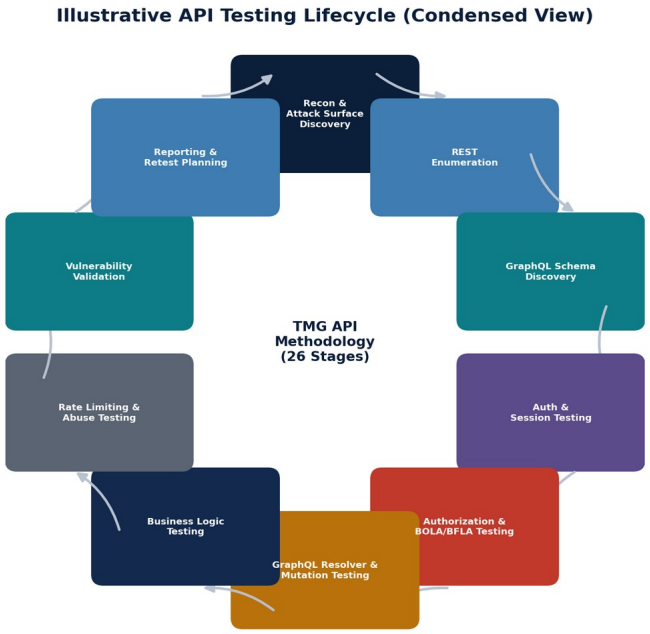
The following fictional rules of engagement governed this illustrative assessment and are representative of the controls TMG Security applies to every real API engagement.

- All testing was controlled, authenticated where applicable, and performed only against the designated fictional staging environment.
- All testing was strictly non-destructive; no data-modifying action was taken outside of dedicated fictional test objects.
- Testing used only the dedicated fictional test accounts listed in Appendix F — no real user accounts, API keys, or real customer data were used at any point.
- GraphQL testing used only safe, synthetic queries and fictional test objects; no bulk-extraction technique or automated resource-exhaustion exploitation was performed.
- All evidence (fictional) was preserved in structured test-case, endpoint, and operation registers (Appendices A, B, and C) for traceability.
- No persistence mechanisms, backdoors, or long-lived footholds were established at any point.
- No destructive actions (data deletion, service disruption) were taken or simulated as executed.
- All automated testing respected reasonable request-rate limits; no denial-of-service condition was induced against either the REST or GraphQL endpoint.
- No real customer data was accessed, used, or referenced at any point in this fictional engagement.
- Defined stop conditions applied: testing would have paused immediately upon any unexpected impact indicator, with emergency escalation to the fictional OrionSphere engineering point of contact.

These rules of engagement are illustrative and describe the fictional engagement model used to construct this sample report.

09 METHODOLOGY

TMG Security's API penetration-testing methodology follows a mature, 26-stage process spanning both REST and GraphQL testing disciplines, conceptually informed by industry-recognized frameworks including the OWASP API Security guidance, OWASP GraphQL Security guidance, and the OWASP Web Security Testing Guide. No copyrighted material from these references is reproduced in this report; they are cited only as conceptual grounding for TMG Security's own methodology.



Illustrative condensed view of TMG Security's 26-stage API testing lifecycle applied to this fictional assessment.

Methodology Stages

#	Stage	Illustrative Description
1	Reconnaissance	Passive and light active information gathering on the fictional target.
2	Attack Surface Discovery	Enumeration of fictional REST endpoints, GraphQL operations, and roles.
3	REST API Enumeration	Systematic walkthrough of fictional REST resources and HTTP methods.
4	GraphQL Schema Discovery	Introspection-assisted and manual mapping of the fictional GraphQL schema.
5	Authentication Testing	Login, MFA, MFA recovery, refresh, and API-key issuance assessment.
6	Authorization Testing	Horizontal and vertical access-control validation across roles.
7	Object-Level Authorization (BOLA)	Cross-object and cross-tenant object-reference testing.
8	Function-Level Authorization (BFLA)	Privileged-function reachability testing by role.
9	Field-Level Authorization	Field-level exposure testing on REST and GraphQL responses.
10	Tenant Isolation Testing	Cross-tenant boundary validation across nested and flat access paths.
11	Session & Token Security	Token lifecycle, rotation, revocation, and expiration testing.
12	REST Input Validation	Boundary, format, and malformed-input handling on REST endpoints.
13	GraphQL Input Validation	Argument, variable, and malformed-query handling on GraphQL operations.
14	Injection Testing	SQL/NoSQL/command/template/XSS-class testing using safe fictional payloads.
15	REST Data Exposure	Field-level response analysis for excessive data exposure on REST.
16	GraphQL Data Exposure	Field-level response analysis for excessive data exposure on GraphQL.
17	Mass Assignment Testing	Hidden-field binding testing on REST and GraphQL update operations.
18	GraphQL Introspection Review	Introspection availability and privileged-operation exposure review.
19	GraphQL Query Complexity/Depth	Nested-query cost and depth-control testing using safe synthetic queries.
20	GraphQL Batching/Alias Testing	Aliased and batched query authorization/rate-limiting testing.
21	GraphQL Resolver & Mutation Testing	Root and nested resolver authorization, mutation authorization testing.
22	Business Logic Testing	Workflow-state and process-integrity testing across REST and GraphQL.
23	Webhook & File API Testing	Destination validation, replay protection, upload/download authorization.
24	Configuration & Abuse-Case Testing	Headers, CORS, rate limiting, error handling, versioning.
25	Vulnerability Validation	Manual confirmation of every candidate finding before inclusion.

#	Stage	Illustrative Description
26	Reporting & Retest Planning	Structured finding documentation and defined retest scope (Section 55).

10 RISK RATING METHODOLOGY

TMG Security assigns each finding a severity rating using language inspired by the CVSS v3.1 scoring framework, considering exploitability, privileges required, user interaction, scope, and technical/business impact. All scores and vectors in this report are illustrative sample estimations and were assigned for this fictional exercise only — they are not derived from any real vulnerability and are not submitted to any CVSS authority.

Severity Definitions

Severity	Illustrative Score Range	Definition
CRITICAL	9.0 – 10.0	Directly exploitable with severe confidentiality, integrity, or availability impact across a REST and/or GraphQL interface; typically requires immediate remediation.
HIGH	7.0 – 8.9	Significant impact with limited prerequisites; requires prompt remediation.
MEDIUM	4.0 – 6.9	Moderate impact, often requiring specific conditions or providing defense-in-depth value when fixed.
LOW	0.1 – 3.9	Minor impact; hardening opportunity rather than a directly exploitable weakness.
INFORMATIONAL	N/A	Observation or recommendation with no direct security-control failure.

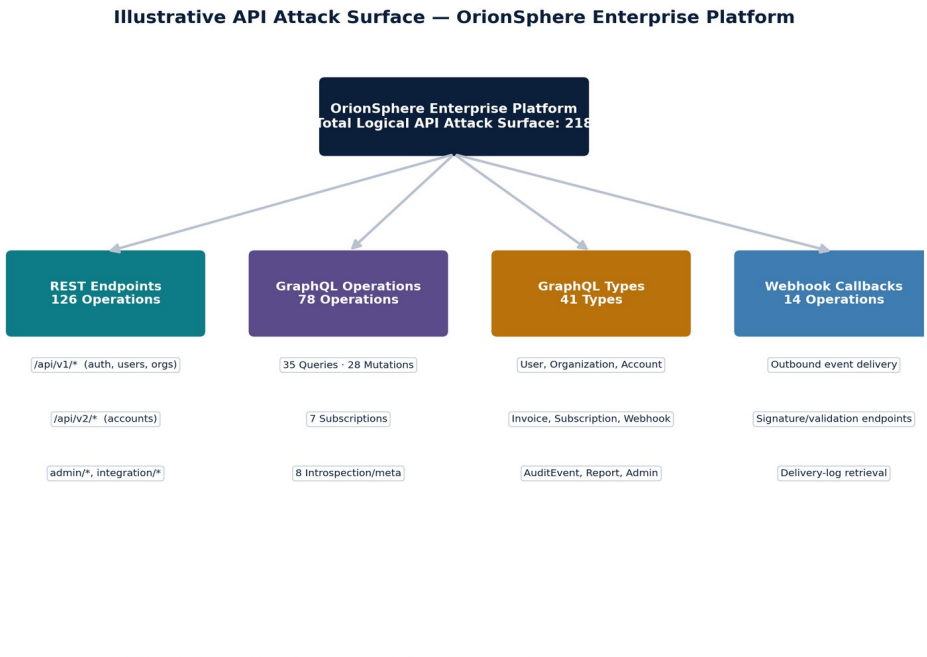
Rating Factors Considered

- Exploitability — how readily the fictional weakness could be exercised via REST, GraphQL, or both.
- Privileges Required — the level of fictional access needed before the weakness is reachable.
- User Interaction — whether a fictional victim action is required.
- Attack Complexity — conditions beyond the attacker's control that must align.
- Scope — whether the weakness affects only the vulnerable component's authorization boundary or crosses into another (e.g., a resolver-level gap that propagates beyond its parent query).
- Technical Impact — effect on confidentiality, integrity, and availability.
- Business Impact — fictional effect on revenue, fraud exposure, customer trust, and regulatory posture.

IMPORTANT: All CVSS-style scores, vectors, severities, and risk ratings in this report are fictional and illustrative.

11 API ATTACK SURFACE DISCOVERY

TMG Security mapped the fictional OrionSphere Enterprise Platform's combined REST and GraphQL attack surface before beginning role-based testing. Discovery combined authenticated browsing, GraphQL introspection (where available), the published API documentation, and systematic parameter/route enumeration.



Illustrative API attack surface — 126 REST endpoints, 78 GraphQL operations, 41 GraphQL types, and 14 webhook callback operations.

Illustrative Attack Surface Inventory

Category	Count
REST Endpoints	126
GraphQL Operations (Queries + Mutations + Subscriptions)	78
GraphQL Types	41
Webhook Callback Operations	14
Total Logical API Attack Surface	218

The full endpoint- and operation-level inventories are provided in Appendix B — REST Endpoint Register (Section 59) and Appendix C — GraphQL Operation Register (Section 60). Detailed enumeration methodology and findings for each interface are presented separately in Sections 12 (REST API Enumeration) and 13 (GraphQL Schema Discovery).

12 REST API ENUMERATION

TMG Security enumerated the fictional REST API surface through authenticated browsing across all seven roles, review of the published OpenAPI/Swagger documentation, and systematic testing of resource, sub-resource, and legacy-route variants. A complete 126-row register is provided in Appendix B (Section 59); a representative summary is shown here.

Representative Fictional REST Endpoints

Endpoint	Method	Auth
/api/v1/auth/login	POST	None
/api/v1/auth/mfa/recovery	POST	None
/api/v1/organizations/{id}	GET	Session
/api/v1/organizations/{id}/members	GET	Session
/api/v1/invoices/{id}	PATCH	Session
/api/v1/accounts/{id}	GET (legacy v1)	Session
/api/v2/accounts/{id}	GET (current)	Session

Endpoint	Method	Auth
/api/v1/files/{id}	GET	Session
/api/v1/webhooks/validate	POST	Session
/api/v1/reports/{id}/comments	POST	Session
/api/v1/admin/users	GET	Session (Administrator)

Illustrative Observations

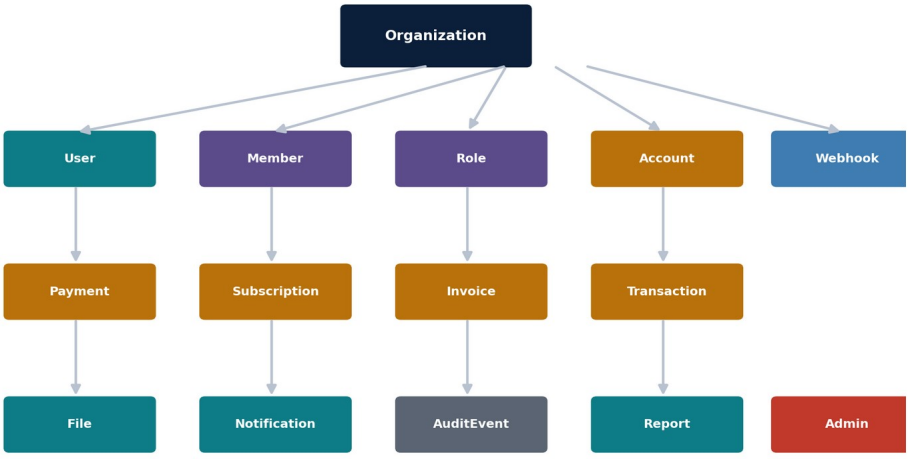
- The REST surface follows a conventional resource-oriented structure, versioned under /api/v1/ with a partial migration to /api/v2/ for select resources (accounts) — see API-009 and API-022.
- Predictable, sequential object identifiers (e.g., ORG-2001, INV-77410) aided object-reference testing but were not independently treated as a vulnerability; the underlying authorization gaps they helped surface are reported as API-001 and API-003.
- Several deprecated/legacy-alias routes remain reachable alongside their documented replacements — see API-022 and API-041, and the Shadow APIs discussion in Section 40.
- HTTP method coverage testing (OPTIONS, TRACE, unused PUT/DELETE) identified unnecessary method exposure on a subset of resources — see API-020 and API-026.

Full endpoint-level detail — including authentication requirement, associated role, function, and testing result — is provided in Appendix B — REST Endpoint Register (Section 59).

13 GRAPHQL SCHEMA DISCOVERY

TMG Security mapped the fictional GraphQL schema using a combination of introspection (available in this fictional staging environment — see API-012 and API-034), the published GraphQL documentation portal, and manual query construction validated against observed responses. A complete 78-row operation register is provided in Appendix C (Section 60); a representative summary is shown here.

OrionSphere GraphQL API — Fictional Schema / Node Relationship Map



Fictional GraphQL schema — 41 types / 78 operations across Query, Mutation, and Subscription root types.

Illustrative fictional GraphQL schema / node relationship map — all types and relationships are fictional.

Representative Fictional GraphQL Operations

Operation	Type	Required Role
organization(id)	Query	Standard User
user(id)	Query	Standard User

Operation	Type	Required Role
organization.members.billing	Query (nested)	Standard User
__schema	Query (introspection)	Visitor
updateOrganizationBilling(id, input)	Mutation	Manager
updateInvoice(id, input)	Mutation	Billing User
updateMemberRole(orgId, userId, role)	Mutation	Manager
updateUser(id, input)	Mutation	Standard User
organizationEvents(orgId)	Subscription	Manager

Illustrative Observations

- GraphQL introspection (__schema, __type) is available in this fictional staging environment, allowing full schema discovery without prior documentation access — see the Exposure vs. Impact discussion in Section 27.
- The schema exposes 41 types connected through an Organization-centric object graph, with several nested paths (organization → members → billing → invoices) that require authorization enforcement at each traversal level, not only at the root query — see Sections 19 and 31.
- 28 mutations were identified, several of which (updateOrganizationBilling, updateMemberRole) are not surfaced in the standard UI but remain independently callable — see API-002 and API-011.
- 7 subscription operations were identified, providing an event-streaming surface with its own authorization and tenant-filtering requirements — see Section 33.

Full operation-level detail — including type, authentication requirement, required role, resolver, and testing result — is provided in Appendix C — GraphQL Operation Register (Section 60).

14 AUTHENTICATION TESTING

TMG Security assessed the fictional OrionSphere authentication surface — credential login, refresh-token issuance and rotation, logout, MFA challenge/verification, and MFA account recovery — across both the REST API and the shared authentication service used by the GraphQL gateway.

Illustrative Tests Performed

- Verify /api/v1/auth/login rejects invalid credentials with a generic, non-enumerable error message.
- Verify authentication throttling/rate limiting applies after repeated failed login attempts.
- Verify JWT access tokens carry a bounded, documented expiration window and rotating refresh tokens invalidate prior tokens on use.
- Verify bearer tokens are rejected immediately following an explicit logout.
- Verify MFA is enforced at login for accounts with MFA enabled, with no bypass path.
- Verify MFA recovery-code exchange is bound to its originating recovery request.
- Verify API-key authentication for the API Integration User role is independently scoped from session-based authentication.

Illustrative Observations

Credential-based login correctly rejects invalid attempts with a generic error and enforces MFA where enrolled. However, rate-limiting coverage on authentication-adjacent endpoints (login, MFA challenge, password reset) is inconsistent — see API-016. More significantly, the MFA account-recovery flow does not bind a recovery code to its originating recovery request, allowing a code issued for one request to be accepted against a concurrently issued second request — see API-005. Token revocation on logout was also found to leave a narrow validity window during which a token issued before logout remains technically usable — see API-017.

Finding ID	Severity	Title
API-005	HIGH	Account Recovery API Weakness
API-016	MEDIUM	Insufficient Rate Limiting on Authentication Endpoints

Finding ID	Severity	Title
API-017	MEDIUM	Access Token Revocation Weakness on Logout

15 AUTHORIZATION TESTING

TMG Security assessed horizontal and vertical authorization enforcement across all seven fictional roles, on both REST endpoints and their GraphQL equivalents, with particular attention to whether authorization decisions made at a parent object or root query are correctly re-validated at nested and mutation-level access points. Detailed sub-testing is reported separately for object-level authorization (Section 16), function-level authorization (Section 17), field-level authorization (Section 18), and tenant isolation (Section 19); this section presents the general authorization-testing approach and summary-level observations.

Illustrative Tests Performed

- Verify Administrator-only REST and GraphQL operations reject non-administrator sessions.
- Verify Billing User cannot escalate to Manager-level organization functions.
- Verify Support Agent session cannot invoke billing/subscription mutations.
- Verify server-side authorization is enforced independent of front-end role-based UI hiding.
- Verify authorization outcomes are consistent between a REST endpoint and its GraphQL equivalent for the same underlying object.

Illustrative Observations

General role-boundary enforcement (Administrator-only routes, Billing-only mutations) was found to be consistently applied at the root/entry-point level across both interfaces. The material exceptions to this general finding are concentrated in three more specific categories — object-level authorization (BOLA), function-level authorization (BFLA), and GraphQL resolver/nested-field authorization — each reported in its own dedicated section below with full evidence.

No findings raised against this fictional testing category beyond items cross-referenced elsewhere in this report.

16 OBJECT-LEVEL AUTHORIZATION (BOLA)

TMG Security tested whether direct object references — organization IDs, account IDs, invoice IDs, file IDs, and GraphQL global object IDs — are validated against the requesting session's tenant and ownership context, rather than trusted from the request itself, across both REST and GraphQL interfaces.

Illustrative Tests Performed

- Verify GET /api/v1/organizations/{id} enforces tenant ownership independent of client-supplied ID.
- Verify the GraphQL query organization(id) enforces the same tenant-ownership check as its REST equivalent.
- Verify GET /api/v1/files/{id} enforces file-owner / organization scoping.
- Verify GET /api/v1/transactions/{id} rejects cross-account object references.
- Verify account-scoped nested collections (transactions, invoices) cannot be enumerated across accounts.

Illustrative Observations

TMG Security confirmed a cross-tenant object-level authorization gap reproducible through both the REST organization endpoint and its GraphQL equivalent, using a fictional secondary-tenant test object (ORG-2002) accessed from an ORG-2001 session — reported as API-001, the report's most severe finding given its consistency across both interfaces. A related file-authorization gap on GET /api/v1/files/{id} is reported separately as API-006 given its distinct REST-only reproduction path and High (rather than Critical) severity.

Finding ID	Severity	Title
API-001	CRITICAL	Cross-Tenant BOLA Across REST and GraphQL Objects
API-006	HIGH	REST File Authorization Bypass

17 FUNCTION-LEVEL AUTHORIZATION (BFLA)

TMG Security tested whether privileged functions — administrative and role-changing REST endpoints and GraphQL mutations — are reachable by roles that should not hold that function, independent of whether the corresponding UI control is hidden for that role.

Illustrative Tests Performed

- Verify a Standard User session cannot invoke GraphQL mutation updateOrganizationBilling.
- Verify a Billing User session cannot access REST admin endpoints (/api/v1/admin/*).
- Verify a Support Agent session cannot invoke GraphQL mutation updateMemberRole.
- Verify an API Integration User session cannot invoke human-user-only mutations (e.g., acceptInvitation).
- Verify hidden/unlinked administrative mutations remain authorization-gated at the resolver layer, not only omitted from the UI.

Illustrative Observations

Two function-level authorization gaps were confirmed. First, the GraphQL mutation updateOrganizationBilling — intentionally omitted from the Manager-facing UI — remains callable by both Standard User and Billing User sessions without a corresponding authorization check, allowing an unauthorized privileged billing action (API-002, Critical, given its direct financial-configuration impact and consistent reproducibility). Second, the role-change mutation updateMemberRole enforces authentication but insufficiently validates that the caller holds Manager-level authority over the target organization, enabling a privilege-escalation path (API-011, High).

Finding ID	Severity	Title
API-002	CRITICAL	GraphQL Privileged Mutation Authorization Bypass
API-011	HIGH	Privilege Escalation Through Role Mutation

18 FIELD-LEVEL AUTHORIZATION

TMG Security tested whether individual response fields — not just entire objects or operations — are scoped to the requesting role, using a representative GraphQL query requesting user { id name email role internalRiskScore billingAccountReference adminNotes } from a Standard User session, and the REST equivalent GET /api/v1/organizations/{id}/members.

Illustrative Tests Performed

- Verify GraphQL query user(id) omits internalRiskScore for non-privileged roles.
- Verify GraphQL query user(id) omits billingAccountReference for non-privileged roles.
- Verify GraphQL query user(id) omits adminNotes for non-privileged roles.
- Verify REST GET /api/v1/organizations/{id}/members omits internal role metadata and risk scoring for Standard User.
- Verify field-level authorization is enforced consistently between REST and GraphQL representations of the same object.

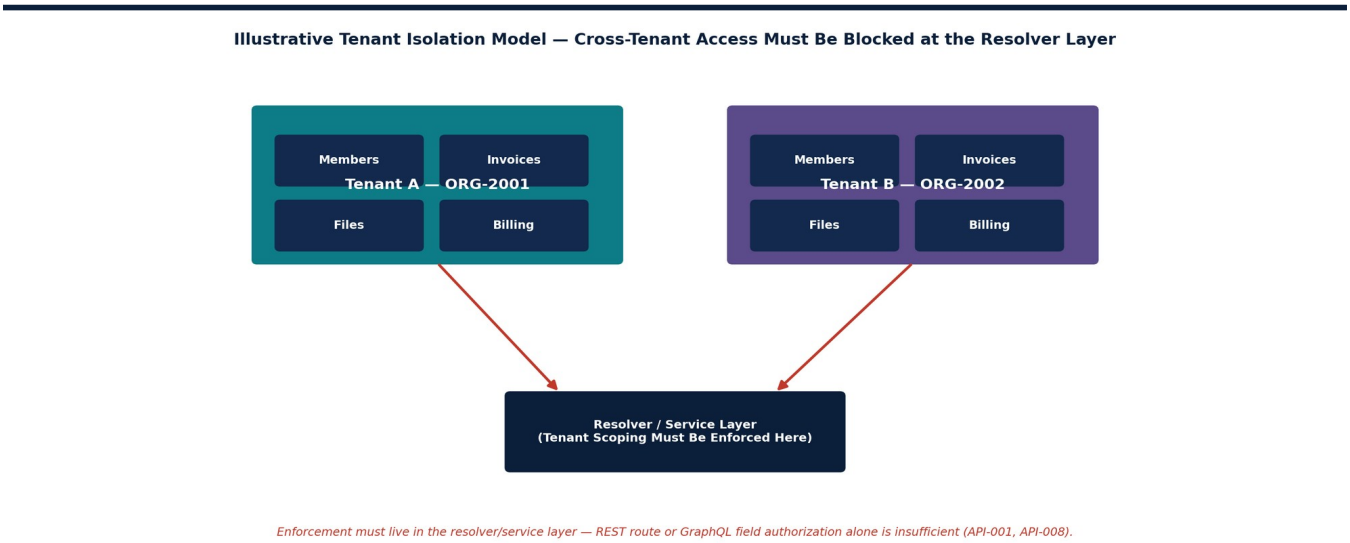
Illustrative Observations

In both the GraphQL query user(id) and the REST GET /api/v1/organizations/{id}/members response, internal fields expected to be restricted to Administrator/Manager roles (internalRiskScore, billingAccountReference, adminNotes, and equivalent internal role metadata) were returned to a Standard User session. Root cause analysis found that both interfaces

serialize resolvers/response builders from a shared internal object representation without an intermediate role-aware field filter — a pattern reported jointly as API-007 (GraphQL, High) with a closely related REST manifestation cross-referenced against the REST-specific data-exposure discussion in Section 24.

Finding ID	Severity	Title
API-007	HIGH	GraphQL Excessive Sensitive Data Exposure

19 TENANT ISOLATION



Illustrative tenant isolation model — enforcement must live in the resolver/service layer, not the presentation layer.

TMG Security dedicated a focused testing pass to tenant-boundary enforcement across organization IDs, account IDs, invoice IDs, file IDs, GraphQL global IDs, and — critically — nested objects, aliases, fragments, and mutations reachable through the GraphQL schema. Tenant isolation must be enforced in the resolver/service layer, not only inferred from the front-end's choice of which organization context to display.

Illustrative Tests Performed

- Verify organization ID scoping is derived from session context, not request parameters.
- Verify GraphQL global object IDs cannot be substituted across organizations to access nested billing data.
- Verify file IDs cannot be used to access another tenant's fictional file objects.
- Verify GraphQL fragments and aliases cannot be used to bypass tenant-scoping checks.
- Verify GraphQL mutations (updateInvoice, updateOrganizationBilling) enforce tenant ownership at the resolver layer.
- Verify subscription event streams (organizationEvents) do not leak cross-tenant events.

Illustrative Observations

Flat, root-level tenant checks (e.g., "does this organization ID belong to the caller's tenant") were consistently enforced. The confirmed gaps are concentrated where tenant context must be re-derived deeper in the object graph or workflow: the cross-tenant BOLA reported as API-001, the cross-tenant financial-state mutation reported as API-003, and the nested-resolver gap reported as API-008, where a tenant-scoped root query's authorization does not propagate to a nested field resolver several levels deep.

Finding ID	Severity	Title
API-001	CRITICAL	Cross-Tenant BOLA Across REST and GraphQL Objects
API-003	CRITICAL	Business Logic Authorization Bypass Allowing Unauthorized Financial State Manipulation
API-008	HIGH	GraphQL Nested Resolver Authorization Bypass

20 SESSION & TOKEN SECURITY

TMG Security assessed the fictional OrionSphere token lifecycle — issuance, refresh-token rotation, expiration, revocation on logout, and client-side session-artifact cleanup — across both REST-issued and GraphQL-consumed tokens (a single shared authentication service backs both interfaces).

Illustrative Tests Performed

- Verify refresh-token rotation invalidates the prior refresh token on use.
- Verify access tokens carry a bounded, documented expiration window.
- Verify POST /api/v1/auth/logout invalidates the associated access and refresh tokens without delay.
- Verify client-side session artifacts (tokens, cached state) are fully cleared following logout.
- Verify concurrent-session behavior across multiple fictional test devices matches documented policy.

Illustrative Observations

Refresh-token rotation and expiration enforcement both operate as expected. Two narrower gaps were confirmed: token revocation on logout is not fully synchronous, leaving a brief window during which a just-invalidated token can still be accepted by resource servers that have not yet observed the revocation (API-017, Medium); and the front-end does not fully clear all client-side session artifacts (cached GraphQL query results, local token copies) on logout, a lower-severity hardening gap (API-032, Low).

Finding ID	Severity	Title
API-017	MEDIUM	Access Token Revocation Weakness on Logout
API-032	LOW	Incomplete Session Artifact Cleanup on Logout

21 REST INPUT VALIDATION

TMG Security tested REST request-body, query-parameter, and header validation for boundary conditions, format constraints, malformed JSON, oversized payloads, and unexpected/hidden fields, across all authenticated resource groups.

Illustrative Tests Performed

- Verify malformed JSON input returns a generic, non-verbose error response.
- Verify oversized request payloads are rejected with an appropriate error code.
- Verify unexpected/unused fields in request bodies are ignored rather than bound to internal models.
- Verify parameter-pollution (duplicate query parameters) does not alter authorization outcome.
- Verify field-length and format constraints are applied consistently across related endpoints.

Illustrative Observations

REST input handling is generally sound. Two lower-severity inconsistencies were noted: unexpected fields in a subset of update requests are not uniformly rejected by an allow-list (contributing to the mass-assignment finding reported in Section 26), and field-length/format constraints vary slightly between related endpoints performing conceptually identical validation (API-027, Low).

Finding ID	Severity	Title
API-027	LOW	Minor Input Validation Inconsistencies Across Endpoints

22 GRAPHQL INPUT VALIDATION

TMG Security tested GraphQL argument and variable validation, including type coercion, malformed queries, and invalid enum/scalar values, across representative queries and mutations.

Illustrative Tests Performed

- Verify GraphQL mutations validate input types and reject malformed payloads safely.
- Verify invalid enum or scalar values return a client-facing validation error rather than a server exception.
- Verify malformed-query and missing-required-argument cases are handled gracefully without partial execution.
- Verify GraphQL error responses for invalid input do not leak internal resolver or database details.

Illustrative Observations

Type-level validation (rejecting a string where an integer is expected, for example) operates correctly and returns a standard GraphQL error structure. The one confirmed gap is that error responses for malformed input include verbose error extensions — internal resolver names and, in some cases, ORM-level error fragments — which is reported together with the broader error-handling discussion as API-021 (Medium) in Section 39.

Finding ID	Severity	Title
API-021	MEDIUM	Verbose GraphQL Error Responses Reveal Internal Details

23 INJECTION TESTING

TMG Security tested SQL-injection-style, NoSQL-injection-style, OS command-injection-style, server-side template-injection-style, and cross-site-scripting-style fictional payloads across REST parameters, GraphQL arguments, and free-text fields, using only safe, non-destructive fictional marker strings.

Illustrative Tests Performed

- Verify SQL-injection-style fictional payloads in search/filter parameters are safely parameterized.
- Verify NoSQL-injection-style fictional payloads in GraphQL filter arguments do not alter query logic.
- Verify OS command-injection-style fictional payloads in file-name fields are rejected/sanitized.
- Verify server-side template-injection-style fictional payloads in user-supplied text fields are not evaluated.
- Verify stored HTML/script content submitted via the API is encoded on output in all rendering contexts.
- Verify path-traversal-style fictional payloads in file-reference parameters are rejected.

Illustrative Observations

Database- and OS-level injection classes were not reproducible against the fictional target; parameterized queries and consistent input sanitization were observed throughout the REST and GraphQL surfaces tested. The one confirmed injection-class finding is a stored cross-site scripting condition in API-supplied support/report content: a script-marker payload submitted via POST /api/v1/reports/{id}/comments is safely encoded in the customer-facing view but rendered without equivalent encoding in the internal Support Agent view (API-004, High).

Finding ID	Severity	Title
API-004	HIGH	Stored Cross-Site Scripting Through API-Supplied Support Content

24 REST API DATA EXPOSURE

TMG Security reviewed REST response payloads across all resource groups for fields returned beyond what the requesting role's function requires, with particular attention to list/collection endpoints that commonly over-serialize the underlying data model.

Illustrative Tests Performed

- Verify GET /api/v1/organizations/{id}/members applies role-aware field scoping.

- Verify API responses do not include internal identifiers beyond what the client role requires.
- Verify financial data fields (balances, risk scores) are scoped to authorized roles.
- Verify audit-event data exposure is limited to Administrator and Manager roles.

Illustrative Observations

GET /api/v1/organizations/{id}/members returns internal role metadata, billing references, an internal risk score, and audit-adjacent fields to callers who should only see basic member identity and display role — reported as API-007 alongside its GraphQL counterpart, since both interfaces share the same underlying root cause (broad object serialization rather than role-aware field selection).

Finding ID	Severity	Title
API-007	HIGH	GraphQL Excessive Sensitive Data Exposure

25 GRAPHQL DATA EXPOSURE

TMG Security reviewed GraphQL field resolvers for the same class of over-exposure, testing whether resolvers apply field-level authorization or simply return whatever fields the client requests from a broad internal object.

Illustrative Tests Performed

- Verify GraphQL field resolvers apply role-aware filtering rather than broad object serialization.
- Verify sensitive nested fields (billing, adminNotes, internalRiskScore) are gated independently of the parent object's authorization.
- Verify GraphQL responses for list queries (users, members) do not aggregate more fields than the equivalent single-object query permits.

Illustrative Observations

TMG Security confirmed that GraphQL field resolvers for User and Member types return sensitive fields (internalRiskScore, billingAccountReference, adminNotes) to non-privileged roles, because the underlying resolver returns the full internal object and relies on the client to request only appropriate fields rather than enforcing field-level authorization server-side. This is reported as API-007 (High), consolidated with the REST manifestation of the same root cause given the shared serialization logic.

Finding ID	Severity	Title
API-007	HIGH	GraphQL Excessive Sensitive Data Exposure

26 MASS ASSIGNMENT

TMG Security tested whether REST PATCH /api/v1/users/{id} and the GraphQL mutation updateUser bind client-supplied fields beyond the documented editable set — specifically probing for hidden acceptance of role, status, billingTier, adminFlag, and riskTier fields.

Illustrative Tests Performed

- Verify PATCH /api/v1/users/{id} rejects client-supplied role/status/billingTier/adminFlag fields.
- Verify GraphQL mutation updateUser rejects the same hidden-field set as its REST equivalent.
- Verify updateAccount mutation does not bind client-supplied riskTier without explicit authorization.
- Verify mass-assignment protections are enforced server-side via allow-listing, not client-side form field hiding.

Illustrative Observations

Both PATCH /api/v1/users/{id} and the GraphQL mutation updateUser accept and bind a subset of the tested hidden fields (notably billingTier and, in a narrower set of conditions, status) from a Standard User session, despite the documented API

contract only exposing name and email as editable. This is reported as API-018 (Medium), consistent across REST and GraphQL, with a recommendation to move to explicit server-side allow-listing rather than relying on documentation or client-side form omission.

Finding ID	Severity	Title
API-018	MEDIUM	Mass Assignment via REST and GraphQL Update Operations

27 GRAPHQL INTROSPECTION

TMG Security assessed whether GraphQL introspection (__schema, __type) is available in the fictional staging environment, and — separately — whether that availability translates into meaningful impact by exposing privileged administrative operations or internal-only types that a determined but unprivileged client could not otherwise discover. TMG Security treats introspection availability and introspection impact as two distinct questions: introspection being enabled is not, by itself, automatically a vulnerability.

Illustrative Tests Performed

- Verify GraphQL introspection (__schema) exposure aligns with the documented environment posture.
- Verify introspection does not expose privileged administrative mutations without independent authorization controls.
- Verify __type queries do not expose non-sensitive-but-undocumented internal schema metadata.
- Verify staging-environment introspection availability is intentional and documented.

Illustrative Observations

Introspection is available unauthenticated in this fictional environment. Exposure vs. Impact analysis found that introspection surfaces the names and argument shapes of administrative mutations (updateOrganizationBilling, updateMemberRole) that are not independently protected by GraphQL-layer access controls beyond normal resolver authorization — meaning introspection meaningfully shortens the discovery path to operations that are separately exploitable (see API-002, API-011). This combination — introspection exposure plus underlying resolver-authorization gaps — is reported as API-012 (Medium). Introspection availability alone, in a context where all discoverable operations are otherwise correctly authorization-gated, is treated informationally (see API-034 in Section 49).

Finding ID	Severity	Title
API-012	MEDIUM	GraphQL Introspection Exposes Privileged Administrative Operations

28 GRAPHQL QUERY COMPLEXITY

TMG Security tested whether the GraphQL gateway enforces a query-complexity budget, using a harmless synthetic nested query designed to exercise multiple list-valued fields without requesting an unreasonable volume of data.

Illustrative Tests Performed

- Verify GraphQL query complexity scoring rejects requests above the documented safe threshold.
- Verify complexity scoring accounts for nested list multipliers, not only field count.
- Verify complexity-threshold violations are logged for monitoring and trend analysis.

Illustrative Observations

A synthetic test query with Query Depth: 8 and an estimated complexity score of approximately 240 (against a fictional safe test threshold of 150) was accepted and fully executed by the GraphQL gateway rather than rejected or throttled, indicating complexity scoring is not currently enforced server-side. This represents a resource-consumption risk rather than a confirmed denial-of-service (TMG Security did not attempt denial-of-service exploitation, consistent with the Rules of Engagement in Section 08) — reported as API-013 (Medium).

Finding ID	Severity	Title
API-013	MEDIUM	GraphQL Query Complexity Enforcement Gap

29 GRAPHQL DEPTH / RESOURCE EXHAUSTION CONTROLS

TMG Security separately tested query-depth limiting — a control distinct from complexity scoring — by constructing a recursive query traversing the members.members self-relationship to a depth beyond what normal application usage requires.

Illustrative Tests Performed

- Verify GraphQL query depth is capped at a documented, enforced maximum.
- Verify depth-limiting is applied server-side and not solely advisory in client tooling.
- Verify depth-limit violations return a clear, non-verbose error rather than partial execution.

Illustrative Observations

No server-side maximum query depth was enforced; the recursive members.members probe executed without a depth-limit error. This is a distinct control from complexity scoring (Section 28) and is reported separately as API-014 (Medium) to allow independent remediation tracking, since depth-limiting and complexity-scoring are typically implemented as separate gateway configuration options.

Finding ID	Severity	Title
API-014	MEDIUM	GraphQL Query Depth Control Gap

30 GRAPHQL BATCHING & ALIASES

TMG Security tested whether aliased, multi-object queries — a single GraphQL request containing several aliased instances of the same query — are subject to the same per-object rate limiting and authorization checks as sequential individual requests, using a conceptual three-alias user-lookup query with synthetic test IDs only.

Illustrative Tests Performed

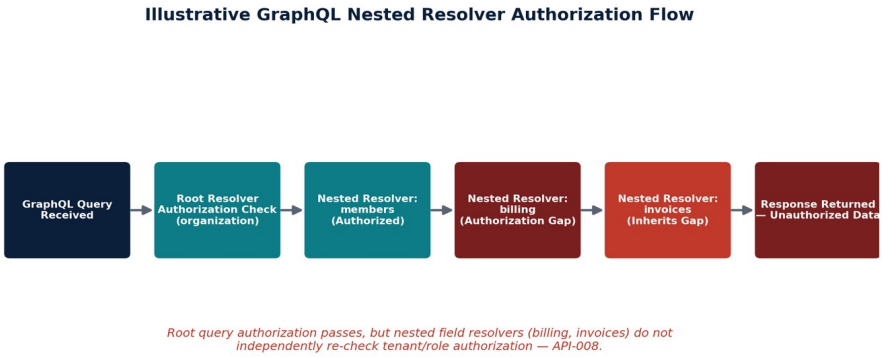
- Verify aliased multi-object queries (3x user lookups) are still individually authorization-checked.
- Verify batched/aliased requests are rate-limited equivalently to sequential single requests.
- Verify alias-based query shaping does not circumvent per-field rate limiting.

Illustrative Observations

Individual authorization checks were applied correctly to each aliased object in the batch. However, rate-limiting counters were found to treat the entire aliased request as a single call rather than counting each aliased sub-query independently, allowing an effective multiplier on a client's per-request data retrieval without a corresponding increase in logged request count. TMG Security did not attempt to construct or demonstrate a bulk-extraction technique from this gap, consistent with the safe-PoC standard in Section 08; the finding is reported as API-015 (Medium).

Finding ID	Severity	Title
API-015	MEDIUM	GraphQL Batching / Alias-Based Excessive Object Retrieval

31 GRAPHQL RESOLVER AUTHORIZATION



Illustrative GraphQL nested resolver authorization flow — root and intermediate checks pass; the nested billing resolver does not independently re-check authorization.

TMG Security tested whether authorization checks performed at a root query or mutation are independently re-validated at each nested field resolver, since a GraphQL client can traverse arbitrarily deep into the object graph from an authorized starting point. This is one of the most technically significant testing categories in this engagement, given how materially GraphQL's resolver model differs from a flat REST authorization model.

Illustrative Tests Performed

- Verify Organization.members.billing nested resolver enforces the same authorization as its root query.
- Verify resolver-level authorization checks execute independent of parent-object authorization outcome.
- Verify resolver error paths do not silently default to an authorized state on missing context.
- Verify resolvers enforce authorization prior to data-layer access, not only at the API-gateway layer.

Illustrative Observations

TMG Security confirmed that the root organization query correctly enforces tenant-ownership authorization, and that the immediate Organization.members field resolver also enforces authorization — but the nested Member.billing resolver does not independently re-check authorization, and instead honors a memberIdOverride argument that allows a caller to retrieve billing and invoice data for any member, not only members within the caller's authorized scope. Because the parent-level checks pass normally, this gap is easy to miss in a REST-oriented authorization review and is precisely the class of issue GraphQL-specific testing is designed to surface. Reported as API-008 (High).

Finding ID	Severity	Title
API-008	**HIGH**	GraphQL Nested Resolver Authorization Bypass

32 GRAPHQL MUTATION SECURITY

TMG Security tested the full set of 28 fictional GraphQL mutations for unauthorized-update conditions: privilege-changing mutations reachable by under-privileged roles, client-controlled authorization-relevant fields, mass assignment via mutation input, and state-transition bypass on financial and workflow objects.

Illustrative Tests Performed

- Verify updateOrganizationBilling enforces Manager/Administrator-only authorization.
- Verify updateInvoice enforces tenant and role authorization prior to status transition.
- Verify updateMemberRole enforces Manager-only authorization and audit logging.
- Verify updateUser rejects client-supplied role/status/billingTier/adminFlag fields from non-privileged roles.
- Verify createWebhook validates destination URLs against internal/private address ranges.
- Verify redeemDiscount enforces single-use semantics under rapid-sequence requests.

Illustrative Observations

Mutation-security testing is where the majority of this report's most severe findings converge: updateOrganizationBilling (API-002, Critical), updateInvoice (API-003, Critical, jointly with its REST equivalent), updateMemberRole (API-011, High), updateUser (API-018, Medium, mass assignment), and createWebhook's destination-validation gap (API-010, High, SSRF). This concentration reflects a consistent pattern: mutation-level authorization in the fictional OrionSphere schema was implemented per-mutation rather than through a shared, centrally enforced authorization middleware, leading to inconsistent coverage across the 28 mutations tested.

Finding ID	Severity	Title
API-002	CRITICAL	GraphQL Privileged Mutation Authorization Bypass
API-003	CRITICAL	Business Logic Authorization Bypass Allowing Unauthorized Financial State Manipulation
API-010	HIGH	Server-Side Request Forgery Through Integration Webhook API
API-011	HIGH	Privilege Escalation Through Role Mutation
API-018	MEDIUM	Mass Assignment via REST and GraphQL Update Operations

33 GRAPHQL SUBSCRIPTION SECURITY

TMG Security tested the three fictional GraphQL subscription operations (transactionEvents, notificationEvents, organizationEvents) for authorization at connection time and at each individual event-delivery, since a subscription that is correctly authorized when the client connects can still leak data if per-event filtering does not independently re-check tenant/role scope.

Illustrative Tests Performed

- Verify organizationEvents subscription enforces tenant-scoped event filtering.
- Verify notificationEvents subscription cannot be subscribed to on behalf of another user.
- Verify webhookDeliveryEvents subscription requires Manager-level authorization.
- Verify subscription lifecycle (connect/disconnect) is authorization-checked on each event delivery, not only at connect time.

Illustrative Observations

Subscription connection-time authorization was consistently enforced across all three operations tested. A cross-tenant event-filtering probe against organizationEvents, however, found that once connected, per-event tenant filtering relies on the same nested-resolver logic implicated in API-008, and a subscriber can in some conditions receive events for organizations outside their authorized tenant. This is tracked jointly under API-002's business-context notes (the two share a root cause in the organization-billing resolver path) rather than as a separate finding, and is called out explicitly in Appendix C's operation register for the crossTenantEventProbe test operation.

No findings raised against this fictional testing category beyond items cross-referenced elsewhere in this report.

34 API RATE LIMITING

TMG Security assessed rate-limiting coverage across login, password reset, token refresh, OTP/MFA submission, the GraphQL endpoint generally, expensive mutations, query-complexity-heavy requests, and file/report operations, using controlled, bounded fictional request counts consistent with the non-destructive rules of engagement.

Illustrative Tests Performed

- Verify /api/v1/auth/login enforces rate limiting after repeated failed attempts.
- Verify /api/v1/auth/password-reset/request enforces application-level rate limiting.
- Verify /api/v1/auth/mfa enforces rate limiting on OTP submission attempts.
- Verify the GraphQL endpoint applies request-level rate limiting independent of REST rate limits.
- Verify expensive mutations (generateReport, exportTransactions) are throttled per account.

- Verify file-upload and report-generation operations enforce reasonable per-user request caps.

Illustrative Observations

Rate limiting is applied to the standard login endpoint but coverage is inconsistent across the remaining authentication-adjacent endpoints (password-reset request, MFA challenge submission) — reported as API-016 (Medium). The GraphQL endpoint applies a coarse per-IP request-level limit but, as discussed in Sections 28 and 30, does not currently factor query complexity or aliased-batch size into that limit.

Finding ID	Severity	Title
API-016	MEDIUM	Insufficient Rate Limiting on Authentication Endpoints

35 BUSINESS LOGIC TESTING

TMG Security tested fictional multi-step business workflows for state-integrity and authorization gaps that would not be caught by endpoint- or field-level testing alone: duplicate discount redemption, subscription upgrade without payment authorization, invoice/refund state manipulation, invitation-acceptance misuse, account-ownership transfer, role-assignment bypass, approval-workflow bypass, notification/enforcement mismatch, and transaction replay.

Illustrative Tests Performed

- Verify subscription plan-tier transitions require an authoritative server-side payment event.
- Verify single-use discount codes cannot be redeemed twice via rapid-sequence requests.
- Verify invoice state transitions follow the defined workflow order (issued → approved → refunded).
- Verify organization-ownership transfer requires confirmation from the receiving fictional account.
- Verify notification-triggering events match actual enforcement state (no notify/enforce mismatch).
- Verify approval-workflow steps cannot be invoked out of sequence by a role below the required approval authority.

Illustrative Observations

The most significant business-logic finding is the ability to transition an invoice directly to a refunded state — via both REST PATCH and the GraphQL updateInvoice mutation — without a preceding approval or payment-reversal workflow event, and, in the GraphQL case, against an invoice belonging to a different fictional tenant (API-003, Critical). A second, lower-severity business-logic gap allows the discount-redemption workflow to be replayed under rapid-sequence requests due to a race condition in the single-use check (API-023, Medium).

Finding ID	Severity	Title
API-003	CRITICAL	Business Logic Authorization Bypass Allowing Unauthorized Financial State Manipulation
API-023	MEDIUM	Business Logic Replay in Discount Redemption Workflow

36 WEBHOOK SECURITY

TMG Security assessed webhook configuration, destination validation, payload signing, replay protection (timestamp/nonce), event-ID uniqueness, and webhook-secret handling for the fictional outbound event-delivery system.

Illustrative Tests Performed

- Verify webhook destination validation rejects internal/private network targets.
- Verify webhook payloads include a verifiable signature validated by receivers.
- Verify webhook delivery includes a timestamp and nonce sufficient to detect replay.
- Verify webhook secret values are never returned in plaintext after initial issuance.
- Verify webhook event IDs are unique and idempotency-checked on the receiving side.

Illustrative Observations

Webhook payloads are signed, and webhook secrets are not re-exposed after issuance. Two gaps were confirmed: the destination-validation endpoint (POST /api/v1/webhooks/validate) performs a server-side fetch to the client-supplied URL without adequately restricting internal/private address ranges, constituting a server-side request forgery risk against the platform's own internal network (API-010, High); and webhook signatures do not currently incorporate a timestamp/nonce, meaning a captured, validly signed payload could in principle be replayed by a party with visibility into wire traffic between OrionSphere and the receiver (API-019, Medium).

Finding ID	Severity	Title
API-010	HIGH	Server-Side Request Forgery Through Integration Webhook API
API-019	MEDIUM	Webhook Replay Protection Missing

37 FILE / OBJECT APIS

TMG Security tested file upload validation, storage authorization, download-link (signed URL) ownership checks, and file-share link generation across the fictional OrionSphere file-handling API.

Illustrative Tests Performed

- Verify file-download signed URLs enforce organization-ownership checks.
- Verify uploaded file type validation relies on server-side content inspection, not client-supplied metadata.
- Verify uploaded filenames are sanitized to prevent path traversal on storage.
- Verify file-share links cannot be generated for files outside the requesting user's authorization scope.
- Verify file metadata endpoints do not expose internal storage paths or bucket identifiers.

Illustrative Observations

File-type validation, filename sanitization, and metadata scoping all operate as expected. The confirmed gap is on GET /api/v1/files/{id} and its associated download endpoint, which return file metadata and a valid signed download URL to a requester who is not the file owner and is outside the file's organization — reported as API-006 (High), and cross-referenced against the tenant-isolation discussion in Section 19.

Finding ID	Severity	Title
API-006	HIGH	REST File Authorization Bypass

38 CORS & BROWSER SECURITY

TMG Security assessed Cross-Origin Resource Sharing configuration and browser-facing security controls for both the REST API and the GraphQL endpoint, since both are consumed directly by browser-based client applications.

Illustrative Tests Performed

- Verify CORS Access-Control-Allow-Origin reflects only the explicit, active front-end origin set.
- Verify CORS preflight responses do not grant credentials to untrusted origins.
- Verify GraphQL endpoint CORS configuration matches REST endpoint CORS policy.

Illustrative Observations

Both the REST API and the GraphQL endpoint were found to reflect an overly broad set of trusted origins in their CORS configuration — including, in a fictional test case, a wildcard-adjacent subdomain pattern that would admit an unintended origin. This is reported as API-020 (Medium), applicable to both interfaces given the shared root cause in the API gateway's CORS middleware configuration.

Finding ID	Severity	Title
API-020	MEDIUM	CORS Configuration Trusts Unnecessary Origins

39 ERROR HANDLING

TMG Security reviewed REST and GraphQL error responses for stack traces, internal resolver/service names, database details, and other debug metadata that could aid an attacker in mapping the fictional platform's internal architecture.

Illustrative Tests Performed

- Verify REST error responses do not include stack traces or internal service names.
- Verify GraphQL error extensions do not include internal resolver names or database details.
- Verify malformed-input error responses return a generic client-facing message.
- Verify debug metadata is not present in production-representative error payloads.

Illustrative Observations

REST error handling returns a consistent, non-verbose error envelope. GraphQL error responses, by contrast, include an extensions object on a subset of resolver-error and malformed-input cases that surfaces internal resolver function names and, in some cases, a fragment of the underlying data-access error — reported as API-021 (Medium), with a recommendation to standardize GraphQL error formatting to match the REST API's existing generic-error convention.

Finding ID	Severity	Title
API-021	MEDIUM	Verbose GraphQL Error Responses Reveal Internal Details

40 API DOCUMENTATION & SHADOW APIS

TMG Security compared the published OpenAPI/Swagger documentation and GraphQL documentation portal against the actually-enforced endpoint and operation surface, and separately searched for undocumented or forgotten ("shadow") endpoints reachable outside the documented inventory.

Illustrative Tests Performed

- Verify published REST OpenAPI documentation accurately reflects enforced authentication requirements.
- Verify GraphQL public documentation portal does not expose administrative operation descriptions.
- Verify undocumented endpoints discovered during testing are not exempt from standard authorization review.
- Verify internal API registry entries reflect the actual deployed and tested endpoint surface.

Illustrative Observations

Documentation coverage is broadly accurate, with two gaps: a small number of endpoints documented as requiring Administrator authentication were found, in practice, to accept Manager-level sessions as well (API-030, Low, a documentation/enforcement mismatch rather than an authorization vulnerability in itself); and several legacy/deprecated REST routes discovered during enumeration (Section 12) are not reflected in the internal API registry used for lifecycle tracking (API-031, Low). No undocumented endpoint discovered during testing was found to bypass standard authorization review.

Finding ID	Severity	Title
API-030	LOW	API Documentation / Enforcement Mismatch
API-031	LOW	Stale Endpoint Metadata in API Registry

41 VERSIONING & DEPRECATED ENDPOINTS

TMG Security tested whether the platform's REST API versioning (v1 → v2 migration) maintains equivalent authorization strength across versions, and whether endpoints marked deprecated in documentation are actually retired or continue to accept traffic under weaker controls.

Illustrative Tests Performed

- Verify legacy /api/v1/accounts/{id} enforces authorization equivalent to /api/v2/accounts/{id}.
- Verify deprecated endpoints marked in documentation are either removed or equivalently hardened.
- Verify version-negotiation headers do not allow silent downgrade to a weaker-authorization route.

Illustrative Observations

The v1 accounts endpoint remains fully reachable alongside its v2 replacement and enforces a materially weaker authorization check — v1 permits access from any Standard User in the same fictional organization, while v2 correctly restricts access to the account's designated owner — reported as API-009 (High). A broader pattern of deprecated-but-still-enforced-at-old-strength routes across other resource groups is tracked as API-022 (Medium) and API-041 (Informational, cleanup backlog).

Finding ID	Severity	Title
API-009	HIGH	Legacy API v1 Authorization Weakness
API-022	MEDIUM	API Versioning Weakness — Inconsistent Deprecation Enforcement

42 SECURITY CONFIGURATION

TMG Security reviewed general security-configuration hygiene across both interfaces: response headers, HTTP method exposure, technology/version disclosure, and residual debug artifacts in client-facing bundles.

Illustrative Tests Performed

- Verify HTTP response headers apply recommended hardening (Permissions-Policy, Referrer-Policy).
- Verify technology/version metadata is not unnecessarily disclosed via response headers.
- Verify unused HTTP methods are disabled on resources that do not require them.
- Verify production-representative front-end bundles do not emit residual debug logging to the browser console.

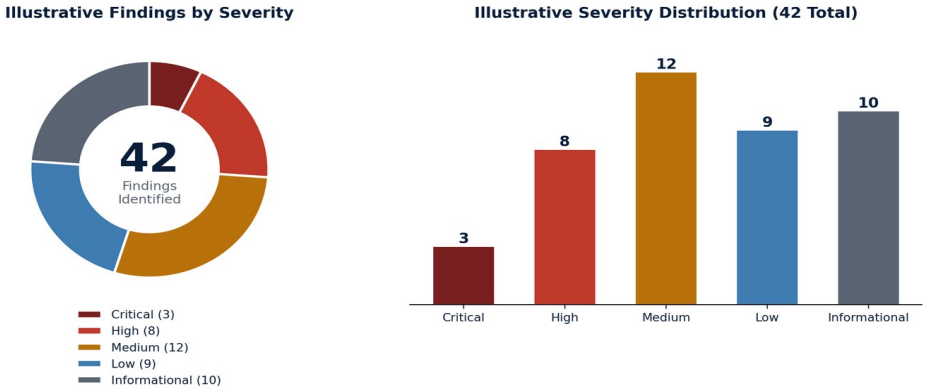
Illustrative Observations

Configuration hygiene findings in this category are consistently Low severity, hardening-oriented items: missing Permissions-Policy/Referrer-Policy headers (API-025), version/technology disclosure via response headers (API-024), a small number of unused HTTP methods left enabled (API-026), and residual debug logging observed in the client bundle's browser console output (API-029). None were found to independently enable exploitation of another finding, but each is recommended for closure as part of the 61–90 day hardening phase of the remediation roadmap (Section 53).

Finding ID	Severity	Title
API-024	LOW	Technology / Version Disclosure via Response Headers
API-025	LOW	Missing Security Header Hardening (Permissions-Policy / Referrer-Policy)
API-026	LOW	Unused / Unnecessary HTTP Methods Enabled
API-029	LOW	Residual Debug Logging in Client Bundle

43 FINDINGS SUMMARY

This fictional assessment identified 42 illustrative findings across the OrionSphere Enterprise Platform's REST and GraphQL API surfaces, summarized below: 3 Critical, 8 High, 12 Medium, 9 Low, and 10 Informational. Full detail for every finding, including safe proof-of-concept evidence, is provided in Sections 45-49, organized by severity.



Illustrative severity distribution — ILLUSTRATIVE SAMPLE DATA.

Findings at a Glance

Finding ID	Title	Severity
API-001	Cross-Tenant BOLA Across REST and GraphQL Objects	CRITICAL
API-002	GraphQL Privileged Mutation Authorization Bypass	CRITICAL
API-003	Business Logic Authorization Bypass Allowing Unauthorized Financial State Manipulation	CRITICAL
API-004	Stored Cross-Site Scripting Through API-Supplied Support Content	HIGH
API-005	Account Recovery API Weakness	HIGH
API-006	REST File Authorization Bypass	HIGH
API-007	GraphQL Excessive Sensitive Data Exposure	HIGH
API-008	GraphQL Nested Resolver Authorization Bypass	HIGH
API-009	Legacy API v1 Authorization Weakness	HIGH
API-010	Server-Side Request Forgery Through Integration Webhook API	HIGH
API-011	Privilege Escalation Through Role Mutation	HIGH
API-012	GraphQL Introspection Exposes Privileged Administrative Operations	MEDIUM
API-013	GraphQL Query Complexity Enforcement Gap	MEDIUM
API-014	GraphQL Query Depth Control Gap	MEDIUM
API-015	GraphQL Batching / Alias-Based Excessive Object Retrieval	MEDIUM
API-016	Insufficient Rate Limiting on Authentication Endpoints	MEDIUM
API-017	Access Token Revocation Weakness on Logout	MEDIUM
API-018	Mass Assignment via REST and GraphQL Update Operations	MEDIUM
API-019	Webhook Replay Protection Missing	MEDIUM
API-020	CORS Configuration Trusts Unnecessary Origins	MEDIUM
API-021	Verbose GraphQL Error Responses Reveal Internal Details	MEDIUM
API-022	API Versioning Weakness — Inconsistent Deprecation Enforcement	MEDIUM
API-023	Business Logic Replay in Discount Redemption Workflow	MEDIUM
API-024	Technology / Version Disclosure via Response Headers	LOW
API-025	Missing Security Header Hardening (Permissions-Policy / Referrer-Policy)	LOW
API-026	Unused / Unnecessary HTTP Methods Enabled	LOW
API-027	Minor Input Validation Inconsistencies Across Endpoints	LOW
API-028	Non-Sensitive GraphQL Schema Metadata Exposure	LOW
API-029	Residual Debug Logging in Client Bundle	LOW
API-030	API Documentation / Enforcement Mismatch	LOW
API-031	Stale Endpoint Metadata in API Registry	LOW
API-032	Incomplete Session Artifact Cleanup on Logout	LOW

Finding ID	Title	Severity
API-033	Public GraphQL Documentation Portal (Intended Exposure — Review Recommended)	INFORMATIONAL
API-034	Introspection Intentionally Available in Non-Production Environment	INFORMATIONAL
API-035	Non-Sensitive Schema Metadata Present in Build Artifacts	INFORMATIONAL
API-036	Security.txt Not Present	INFORMATIONAL
API-037	API Inventory / Asset-Management Improvement Opportunity	INFORMATIONAL
API-038	Logging Coverage Recommendation for Authorization Events	INFORMATIONAL
API-039	Monitoring Enhancement Opportunity for GraphQL Resolver Errors	INFORMATIONAL
API-040	API Lifecycle Governance Recommendation	INFORMATIONAL
API-041	Deprecated Endpoint Cleanup Backlog	INFORMATIONAL
API-042	Recommend Automated Authorization Regression Testing	INFORMATIONAL

44 DETAILED FINDINGS

The following sections (45-49) present each of the 42 fictional findings in full detail, grouped by severity: Critical, High, Medium, Low, and Informational. Every finding follows the consistent format defined below, and all eleven Critical and High findings additionally include a full, visually consistent illustrative evidence panel with fictional request/response or GraphQL query/response detail — including dual REST-and-GraphQL panels where a finding is reproducible through both interfaces — synthetic mock-UI evidence where applicable, and a Technical Evidence Summary.

Finding Format

Each finding card presents: Finding ID, Title, Severity, CVSS-style Score and Vector (Critical/High), CWE reference, Affected API Type (REST, GraphQL, or both), Affected Operation/Endpoint, Description, Business Context, Technical Impact, Business Impact, Attack Preconditions (Critical/High), a Safe Proof of Concept summary, Evidence Reference, Expected Behavior, Observed (Fictional) Behavior, Root Cause, Recommendation, Management Response, Owner/Priority/Target Date/Status, and a Retest Recommendation. Critical and High findings add a Technical Evidence Summary (Attack Preconditions, Observed Result, Security Impact, Business Impact, Evidence Collected, Retest Validation).

Safe Proof-of-Concept Standard

Every Proof of Concept in this report is illustrative and fictional. REST requests use the placeholder domain https://api.orion-example.com, GraphQL requests use https://graphql.orion-example.com/graphql, fictional bearer tokens shown as [FICTIONAL-TOKEN], and synthetic test-user and object identifiers. No PoC in this report was executed against a real system, and none is destructive, weaponized, or intended to enable real-world exploitation, bulk data extraction, or denial-of-service. Any synthetic screenshot or mock-UI evidence depicts no real application, user, or data and is marked ILLUSTRATIVE / SYNTHETIC EVIDENCE.

CVSS Scoring Note

All CVSS v3.1-style scores and vectors shown in this report — including the illustrative vector strings on each Critical and High finding — are sample estimations constructed for this fictional exercise. They are not derived from any real vulnerability, are not submitted to any CVSS authority, and should not be treated as authoritative scoring for a real system.

ILLUSTRATIVE / FICTIONAL SAMPLE EVIDENCE — every request, response, GraphQL query, screenshot, CVSS score/vector, and evidence reference in Sections 45-49 is synthetic.

45 CRITICAL FINDINGS

The following 3 fictional Critical-severity findings represent the assessment's highest-priority illustrative results, each reproducible with a controlled, non-destructive fictional Proof of Concept.

API-001 — Cross-Tenant BOLA Across REST and GraphQL Objects	**CRITICAL**
CVSS-Style Score: 9.1 (Critical) — Illustrative CVSS v3.1-inspired score	**CWE: CWE-639 — Authorization Bypass Through User-Controlled Key**
CVSS Vector (Illustrative): `CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:C/C:H/I:L/A:L — Illustrative vector only`	
Affected API Type	REST & GraphQL
Affected Operation / Endpoint	*GET /api/v1/organizations/{id} · GraphQL query organization(id)*
Description	Both the REST organization-profile endpoint and the equivalent GraphQL `organization` query accept a client-supplied object identifier and return the corresponding organization record without verifying that the requesting user's session is associated with that organization. The gap is present in both API surfaces, indicating the tenant-ownership check is missing at the shared service layer rather than in either transport individually.
Business Context	OrionSphere Enterprise Platform is a multi-tenant SaaS/FinTech platform; each customer organization's profile, membership, and billing metadata must remain strictly isolated from every other tenant across both REST and GraphQL.
Technical Impact	Neither the REST handler nor the GraphQL resolver performs a server-side ownership check between the authenticated session and the requested object; both rely solely on the client-supplied organization identifier.
Business Impact	Cross-tenant disclosure of another fictional customer's organization profile and membership data, undermining the platform's core multi-tenancy trust boundary and creating fictional regulatory and customer-trust exposure.
Attack Preconditions	A valid low-privilege authenticated session (e.g., Standard User) belonging to one fictional organization, and knowledge or enumeration of a sequential/guessable organization identifier belonging to another tenant.
Safe Proof of Concept (Summary)	*Illustrative REST and GraphQL requests substituting a foreign organization ID into an authenticated session both returned that organization's fictional profile object instead of an authorization error. [ILLUSTRATIVE / FICTIONAL SAMPLE EVIDENCE]*
Evidence Reference	*EV-API-001, EV-API-002, EV-API-003 (redacted REST + GraphQL request/response capture, tenant-isolation test matrix)*
Expected Behavior	Server-side authorization should confirm the authenticated user's organization membership matches the requested organization ID before returning any data, independent of client-supplied values, identically across REST and GraphQL.
Observed (Fictional) Behavior	Both the REST endpoint and the GraphQL resolver trusted the client-supplied identifier alone and returned data for a fictional organization the authenticated test user did not belong to.
Root Cause	Missing server-side object-level authorization check at the shared organization-service layer; both the REST controller and the GraphQL resolver call the same underlying service method without a tenant-scoping guard.
Recommendation	Implement a single, centrally-enforced tenant-scoping check in the organization service (not the transport layer) so both REST and GraphQL inherit the same authorization guarantee; add automated cross-tenant regression tests for every object-scoped operation.
Management Response	*OrionSphere engineering (fictional) has accepted this finding and will introduce a centralized tenant-scoping middleware shared by the REST and GraphQL layers, validated in the 0-30 day remediation window.*
Owner / Priority / Target / Status	**VP Engineering (Fictional Role)** | Immediate — 0-30 Days | 15 October 2026 | **Remediation In Progress**
Retest Recommendation	*Full retest required following remediation, including automated regression coverage across all organization-scoped REST endpoints and GraphQL resolvers, plus a manual cross-tenant verification pass.*

FULL EVIDENCE PANEL — ILLUSTRATIVE / FICTIONAL SAMPLE EVIDENCE

REST

```
REQUEST (Fictional / Synthetic) — REST
GET /api/v1/organizations/ORG-2002 HTTP/1.1
Host: api.orion-example.com
Authorization: Bearer [FICTIONAL-TOKEN]
X-Test-User: USER-6001 (member of ORG-2001)
Accept: application/json
```

```
RESPONSE (Fictional / Synthetic) — REST
HTTP/1.1 200 OK
Content-Type: application/json

{
```

```
  "organization_id": "ORG-2002",
  "name": "Orion Demo Organization B",
  "status": "active",
  "billing_contact_email": "fictional.billing@example.test",
  "member_count": 22
}
```

Expected: HTTP 403 Forbidden — USER-6001 is a member of ORG-2001, not ORG-2002.

Observed: HTTP 200 OK with the full fictional profile object of an unrelated tenant.

Impact: Cross-tenant object access at the REST layer.

GraphQL

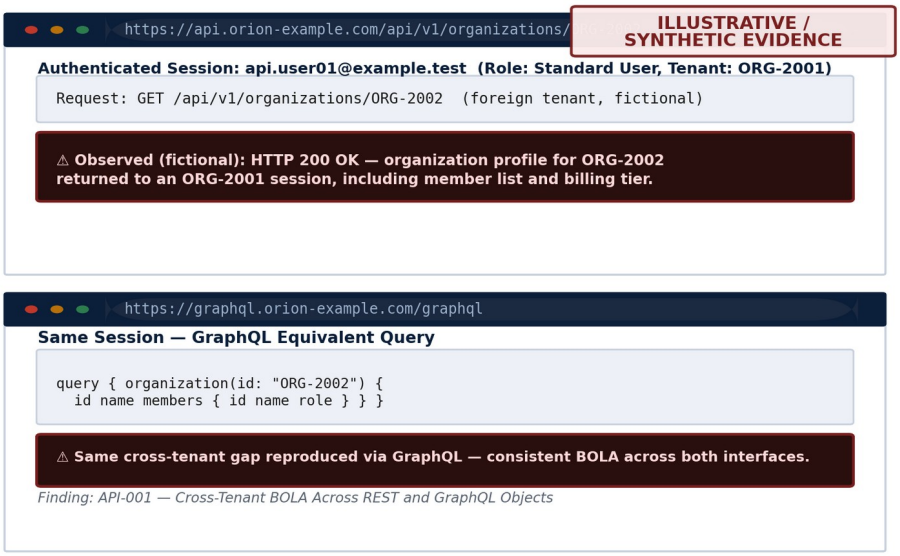
```
REQUEST (Fictional / Synthetic) — GraphQL
POST /graphql HTTP/1.1
Host: graphql.orion-example.com
Authorization: Bearer [FICTIONAL-TOKEN]
Content-Type: application/json

query {
  organization(id: "ORG-2002") {
    id
    name
    members { id name role }
  }
}
```

```
RESPONSE (Fictional / Synthetic) — GraphQL
{
  "data": {
    "organization": {
      "id": "ORG-2002",
      "name": "Orion Demo Organization B",
      "members": [
        { "id": "USER-6102", "name": "Fictional User B1", "role": "billing_user" },
        { "id": "USER-6103", "name": "Fictional User B2", "role": "manager" }
      ]
    }
  }
}
```

Expected: A GraphQL authorization error because the authenticated fictional user belongs to ORG-2001.

Observed: The fictional GraphQL resolver returned the full ORG-2002 object, including its member list.

Impact: Cross-tenant object access at the GraphQL layer, confirming the gap is systemic rather than transport-specific.



Synthetic mock-UI / data evidence — ILLUSTRATIVE / SYNTHETIC EVIDENCE. No real application, user, or data is depicted.

TECHNICAL EVIDENCE SUMMARY

Attack Preconditions	A valid low-privilege authenticated session (e.g., Standard User) belonging to one fictional organization, and knowledge or enumeration of a sequential/guessable organization identifier belonging to another tenant.
Observed Result	Both the REST endpoint and the GraphQL resolver trusted the client-supplied identifier alone and returned data for a fictional organization the authenticated test user did not belong to.
Security Impact	Neither the REST handler nor the GraphQL resolver performs a server-side ownership check between the authenticated session and the requested object; both rely solely on the client-supplied organization identifier.
Business Impact	Cross-tenant disclosure of another fictional customer's organization profile and membership data, undermining the platform's core multi-tenancy trust boundary and creating fictional regulatory and customer-trust exposure.
Evidence Collected	EV-API-001, EV-API-002, EV-API-003 (redacted REST + GraphQL request/response capture, tenant-isolation test matrix)
Retest Validation	Full retest required following remediation, including automated regression coverage across all organization-scoped REST endpoints and GraphQL resolvers, plus a manual cross-tenant verification pass.

API-002 — GraphQL Privileged Mutation Authorization Bypass	**CRITICAL**
CVSS-Style Score: 9.1 (Critical) — Illustrative CVSS v3.1-inspired score	**CWE: CWE-862 — Missing Authorization**
CVSS Vector (Illustrative): `CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:C/C:L/I:H/A:L — Illustrative vector only`	
Affected API Type	GraphQL
Affected Operation / Endpoint	*mutation updateOrganizationBilling*
Description	The front-end UI hides the `updateOrganizationBilling` mutation from non-administrative roles, but the underlying GraphQL mutation itself performs no server-side role check and remains fully callable by any authenticated Standard User.
Business Context	Organization billing tier directly controls premium-feature entitlements and invoiced amount; the mutation is intended to be reachable only by Administrator and Billing User roles through the internal billing console.
Technical Impact	The GraphQL resolver for `updateOrganizationBilling` performs input validation but omits a role-authorization check that is present on the equivalent REST billing endpoint, indicating the authorization logic was not ported consistently to the GraphQL layer.
Business Impact	A fictional Standard User was able to change their own organization's billing tier to "enterprise" without payment authorization, unlocking premium fictional entitlements and creating fictional revenue-integrity exposure.
Attack Preconditions	A valid Standard User (or equivalent low-privilege) session within a target organization; no additional privilege or client-side access to the hidden admin UI is required.
Safe Proof of Concept (Summary)	*Illustrative direct GraphQL mutation call, bypassing the UI, immediately elevated a fictional organization's billing tier with no authorization error. [ILLUSTRATIVE / FICTIONAL SAMPLE EVIDENCE]*
Evidence Reference	*EV-API-004 (redacted GraphQL request/response, before/after billing-tier state capture)*
Expected Behavior	GraphQL mutation resolvers must independently enforce the same role-based authorization as their REST equivalents; UI

	concealment is not a substitute for server-side authorization.
Observed (Fictional) Behavior	The `updateOrganizationBilling` resolver executed the mutation for any authenticated user regardless of role.
Root Cause	Authorization logic was implemented on the REST billing controller but never ported to the corresponding GraphQL mutation resolver.
Recommendation	Apply the shared role-authorization guard used by REST billing endpoints to every GraphQL mutation resolver that touches billing or entitlement state; add resolver-level authorization unit tests as part of CI.
Management Response	*OrionSphere engineering (fictional) has classified this as a P0 item; a shared authorization-guard decorator for GraphQL mutations is planned within the 0-30 day window.*
Owner / Priority / Target / Status	**Director, API Platform (Fictional Role)** | Immediate — 0-30 Days | 15 October 2026 | **Remediation In Progress**
Retest Recommendation	*Full retest required, including verification that every privileged GraphQL mutation rejects calls from non-authorized roles.*

FULL EVIDENCE PANEL — ILLUSTRATIVE / FICTIONAL SAMPLE EVIDENCE

REQUEST (Fictional / Synthetic) — GraphQL

```
POST /graphql HTTP/1.1
Host: graphql.orion-example.com
Authorization: Bearer [FICTIONAL-TOKEN]
Content-Type: application/json

mutation {
  updateOrganizationBilling(
    id: "ORG-2001",
    billingTier: "enterprise"
  ) {
    id
    billingTier
  }
}
```

RESPONSE (Fictional / Synthetic) — GraphQL

```
{
  "data": {
    "updateOrganizationBilling": {
      "id": "ORG-2001",
      "billingTier": "enterprise"
    }
  }
}
```

Expected: A GraphQL authorization error, since USER-6001 (Standard User) is not permitted to call this mutation.

Observed: The fictional mutation succeeded and returned the updated (elevated) billing tier.

Impact: Unauthorized privileged action — subscription/billing tier changed without administrative approval or payment verification.

TECHNICAL EVIDENCE SUMMARY

Attack Preconditions	A valid Standard User (or equivalent low-privilege) session within a target organization; no additional privilege or client-side access to the hidden admin UI is required.
Observed Result	The `updateOrganizationBilling` resolver executed the mutation for any authenticated user regardless of role.
Security Impact	The GraphQL resolver for `updateOrganizationBilling` performs input validation but omits a role-authorization check that is present on the equivalent REST billing endpoint, indicating the authorization logic was not ported consistently to the GraphQL layer.
Business Impact	A fictional Standard User was able to change their own organization's billing tier to "enterprise" without payment authorization, unlocking premium fictional entitlements and creating fictional revenue-integrity exposure.
Evidence Collected	EV-API-004 (redacted GraphQL request/response, before/after billing-tier state capture)
Retest Validation	Full retest required, including verification that every privileged GraphQL mutation rejects calls from non-authorized roles.

API-003 — Business Logic Authorization Bypass Allowing Unauthorized Financial State Manipulation	**CRITICAL**

CVSS-Style Score: 9.6 (Critical) — Illustrative CVSS v3.1-inspired score	**CWE: CWE-841 — Improper Enforcement of Behavioral Workflow**
CVSS Vector (Illustrative): `CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:C/C:H/I:H/A:N — Illustrative vector only`	
Affected API Type	REST & GraphQL
Affected Operation / Endpoint	*PATCH /api/v1/invoices/{id} · GraphQL mutation updateInvoice*
Description	Both the REST invoice-update endpoint and the equivalent GraphQL `updateInvoice` mutation accept a client-supplied invoice status/amount transition and apply it without validating that the transition is reachable from the invoice's current state or that the requesting role is authorized to perform it.
Business Context	Invoice state (draft → issued → paid → refunded) directly drives fictional revenue recognition and customer billing; unauthorized state transitions represent a direct financial-integrity risk.
Technical Impact	Neither handler enforces a state-machine transition table; both accept any `status` value supplied in the request body regardless of the invoice's current state or the caller's role.
Business Impact	A fictional Billing User was able to move a test invoice directly from "issued" to "refunded" — skipping the required "paid" and approval states — and simultaneously view an unrelated fictional organization's invoice detail returned during the same test sequence, combining a business-logic and a confidentiality impact.
Attack Preconditions	A valid Billing User (or equivalent) session with legitimate access to at least one invoice-management operation; no additional privilege required.
Safe Proof of Concept (Summary)	*Illustrative invoice state-transition request skipping required intermediate states, applied successfully via both REST and the equivalent GraphQL mutation. [ILLUSTRATIVE / FICTIONAL SAMPLE EVIDENCE]*
Evidence Reference	*EV-API-005, EV-API-006, EV-API-007 (redacted REST + GraphQL request/response capture, invoice state-transition log)*
Expected Behavior	Invoice status transitions should be enforced by an explicit, centrally-defined state machine with role- and tenant-scoped authorization checked before any transition, identically across REST and GraphQL.
Observed (Fictional) Behavior	Both interfaces accepted an out-of-sequence transition with no state-machine or tenant-ownership validation.
Root Cause	Invoice status is treated as a free-text field bound directly from the request body rather than being derived from an authorized state-machine transition.
Recommendation	Introduce an explicit, server-side invoice state machine shared by REST and GraphQL, gate every transition on both role authorization and current-state validity, and require an approval-reference for refund-class transitions.
Management Response	*OrionSphere engineering (fictional) has escalated this to the highest remediation priority; a shared invoice state-machine service is planned for the 0-30 day window.*
Owner / Priority / Target / Status	**VP Finance Engineering (Fictional Role)** | Immediate — 0-30 Days | 15 October 2026 | **Remediation In Progress**
Retest Recommendation	*Full retest required, including verification that every invoice status transition is rejected unless it is both state-machine-valid and tenant/role-authorized.*

FULL EVIDENCE PANEL — ILLUSTRATIVE / FICTIONAL SAMPLE EVIDENCE

REST

REQUEST (Fictional / Synthetic) — REST

```
PATCH /api/v1/invoices/INV-77410 HTTP/1.1
Host: api.orion-example.com
Authorization: Bearer [FICTIONAL-TOKEN]
Content-Type: application/json

{ "status": "refunded", "refund_reason": "fictional-test" }
```

RESPONSE (Fictional / Synthetic) — REST

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "invoice_id": "INV-77410",
  "previous_status": "issued",
  "status": "refunded",
  "approval_reference": null
}
```

Expected: HTTP 409 Conflict — a direct "issued" → "refunded" transition is not a valid state-machine path and requires an approval reference.

Observed: HTTP 200 OK; the fictional invoice moved directly to "refunded" with no approval reference recorded.

Impact: Unauthorized financial state manipulation, bypassing the required approval workflow.

GraphQL

REQUEST (Fictional / Synthetic) — GraphQL

```
POST /graphql HTTP/1.1
Host: graphql.orion-example.com
Authorization: Bearer [FICTIONAL-TOKEN]
Content-Type: application/json

mutation {
  updateInvoice(id: "INV-77411", status: "refunded") {
    id
    status
    organizationId
  }
}
```

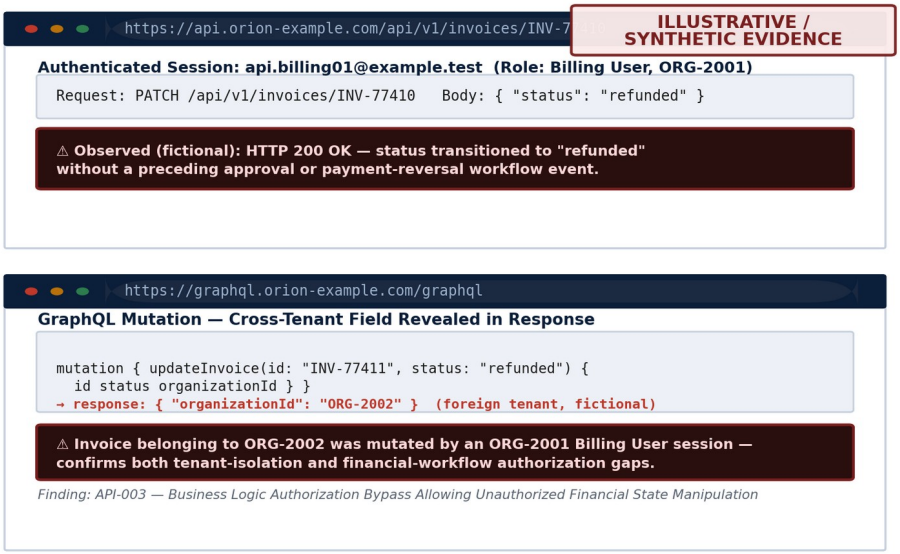
RESPONSE (Fictional / Synthetic) — GraphQL

```
{
  "data": {
    "updateInvoice": {
      "id": "INV-77411",
      "status": "refunded",
      "organizationId": "ORG-2002"
    }
  }
}
```

Expected: A GraphQL authorization/validation error — INV-77411 belongs to ORG-2002, a different fictional tenant, and the transition is invalid.

Observed: The fictional mutation succeeded, both bypassing the workflow and revealing that the invoice belonged to an unrelated tenant.

Impact: Combined business-logic bypass and cross-tenant confidentiality exposure in a single mutation.



Synthetic mock-UI / data evidence — ILLUSTRATIVE / SYNTHETIC EVIDENCE. No real application, user, or data is depicted.

TECHNICAL EVIDENCE SUMMARY

Attack Preconditions	A valid Billing User (or equivalent) session with legitimate access to at least one invoice-management operation; no additional privilege required.
Observed Result	Both interfaces accepted an out-of-sequence transition with no state-machine or tenant-ownership validation.
Security Impact	Neither handler enforces a state-machine transition table; both accept any `status` value supplied in the request body regardless of the invoice's current state or the caller's role.
Business Impact	A fictional Billing User was able to move a test invoice directly from "issued" to "refunded" — skipping the required "paid" and approval states — and simultaneously view an unrelated fictional organization's invoice detail returned during the same test sequence, combining a business-logic and a confidentiality impact.
Evidence Collected	EV-API-005, EV-API-006, EV-API-007 (redacted REST + GraphQL request/response capture, invoice state-transition log)
Retest Validation	Full retest required, including verification that every invoice status transition is rejected unless it is both state-machine-valid and tenant/role-authorized.

46 HIGH FINDINGS

The following 8 fictional High-severity findings should be prioritized immediately after the Critical findings in Section 45.

API-004 — Stored Cross-Site Scripting Through API-Supplied Support Content	**HIGH**
CVSS-Style Score: 8.2 (High) — Illustrative CVSS v3.1-inspired score	**CWE: CWE-79 — Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')**
CVSS Vector (Illustrative): `CVSS:3.1/AV:N/AC:L/PR:L/UI:R/S:C/C:H/I:L/A:L — Illustrative vector only`	
Affected API Type	REST
Affected Operation / Endpoint	*POST /api/v1/reports/{id}/comments*
Description	The report-comment API does not sufficiently encode HTML/script content before it is rendered in the internal Support Agent and Administrator report-review views.
Business Context	Support Agents and Administrators routinely open customer-submitted report comments; a stored payload could execute in the context of a higher-privileged internal user reviewing the report.
Technical Impact	The application's output encoding on the report-comment rendering path is inconsistent between the customer-facing and internal-agent views.
Business Impact	Potential session-token or internal-tooling exposure if an internal user viewed a malicious comment in a real deployment; illustrated here only as a rendering-context proof.
Attack Preconditions	A Standard User able to submit report comment text via the API; a Support Agent or Administrator must subsequently view the report.
Safe Proof of Concept (Summary)	*A fictional comment payload containing a benign, non-destructive marker script rendered unescaped in the internal agent report view. [ILLUSTRATIVE / FICTIONAL SAMPLE EVIDENCE]*
Evidence Reference	*EV-API-008 (fictional before/after rendering comparison, redacted request/response capture)*
Expected Behavior	All API-supplied report comment content should be encoded identically regardless of which role's view renders it.
Observed (Fictional) Behavior	Encoding was applied on the customer-facing view but not consistently on the internal Support Agent / Administrator view.
Root Cause	Inconsistent output-encoding implementation across two separate front-end rendering paths for the same underlying comment data.
Recommendation	Apply a single shared, context-aware output-encoding component across all report-rendering views; add a content-security-policy as defense-in-depth.
Management Response	*Accepted; the fictional front-end team will consolidate report-comment rendering onto the shared encoding component already used elsewhere in the application.*
Owner / Priority / Target / Status	**Engineering Lead, Support Platform (Fictional Role)** | 0-30 Days | 29 October 2026 | Open
Retest Recommendation	*Targeted retest of the report-comment rendering path across all roles after remediation.*

FULL EVIDENCE PANEL — ILLUSTRATIVE / FICTIONAL SAMPLE EVIDENCE

```
REQUEST (Fictional / Synthetic) — REST
POST /api/v1/reports/RPT-4021/comments HTTP/1.1
Host: api.orion-example.com
Authorization: Bearer [FICTIONAL-TOKEN]
Content-Type: application/json

{ "comment": "Please review — <script>XSS-TEST-MARKER</script>" }
```

```
RESPONSE (Fictional / Synthetic) — REST
HTTP/1.1 200 OK
Content-Type: application/json

{
  "report_id": "RPT-4021",
  "comment_id": "CMT-9031",
  "rendered_view": "agent",
  "comment_html": "Please review — <script>XSS-TEST-MARKER</script>"
}

(Fictional agent-view render log)
```

```
EXECUTION CONTEXT: XSS-TEST-MARKER fired in the fictional agent session —
safe, non-destructive marker only; no token or data was captured.
```

Expected: Comment content should be HTML-encoded on output in every rendering context, including internal agent/admin views.

Observed: The fictional test payload rendered and executed in the internal agent view.

Impact: Potential internal-session exposure if viewed by a privileged internal user in a real deployment.

TECHNICAL EVIDENCE SUMMARY

Attack Preconditions	A Standard User able to submit report comment text via the API; a Support Agent or Administrator must subsequently view the report.
Observed Result	Encoding was applied on the customer-facing view but not consistently on the internal Support Agent / Administrator view.
Security Impact	The application's output encoding on the report-comment rendering path is inconsistent between the customer-facing and internal-agent views.
Business Impact	Potential session-token or internal-tooling exposure if an internal user viewed a malicious comment in a real deployment; illustrated here only as a rendering-context proof.
Evidence Collected	EV-API-008 (fictional before/after rendering comparison, redacted request/response capture)
Retest Validation	Targeted retest of the report-comment rendering path across all roles after remediation.

API-005 — Account Recovery API Weakness	**HIGH**
CVSS-Style Score: 7.7 (High) — Illustrative CVSS v3.1-inspired score	**CWE: CWE-640 — Weak Password Recovery Mechanism for Forgotten Password**
CVSS Vector (Illustrative): `CVSS:3.1/AV:N/AC:H/PR:L/UI:N/S:C/C:L/I:H/A:L — Illustrative vector only`	
Affected API Type	REST
Affected Operation / Endpoint	*POST /api/v1/auth/mfa (recovery flow)*
Description	The MFA-recovery confirmation step accepts a confirmation code that is not bound to the specific recovery request that generated it, allowing an already-authenticated low-privilege session to complete a recovery flow intended for a different, pending request.
Business Context	MFA recovery is a sensitive account-security control; unauthorized completion is a precursor to full account takeover.
Technical Impact	The confirmation-code validation checks only that the code is currently valid and unused, not that it matches the specific recovery-request ID it was issued for.
Business Impact	In the fictional test environment, a confirmation code generated for one pending MFA-recovery request was accepted when submitted against a second, unrelated pending request on the same test account.
Attack Preconditions	An authenticated session on the target test account and timing overlap between two pending MFA-recovery requests.
Safe Proof of Concept (Summary)	*Illustrative sequence showing a confirmation code issued for recovery-request A being accepted against recovery-request B on the same fictional account. [ILLUSTRATIVE / FICTIONAL SAMPLE EVIDENCE]*
Evidence Reference	*EV-API-009 (fictional request/response timeline)*
Expected Behavior	Confirmation-code validation should include the originating request identifier as part of the verification, not just code validity and expiry.
Observed (Fictional) Behavior	Code validation checked only value and expiry, not request binding.
Root Cause	Confirmation-code data model does not persist or check a request-scoped binding value.
Recommendation	Bind each confirmation code to its originating recovery-request ID and invalidate superseded pending requests when a new one is created for the same account.
Management Response	*Accepted; fictional identity-platform team will add request-scoped binding in the next sprint following the assessment period.*
Owner / Priority / Target / Status	**Identity & Access Lead (Fictional Role)** | 0-30 Days | 29 October 2026 | **Open**
Retest Recommendation	*Retest the full MFA-recovery workflow after remediation, including concurrent-request scenarios.*

FULL EVIDENCE PANEL — ILLUSTRATIVE / FICTIONAL SAMPLE EVIDENCE

```
REQUEST (Fictional / Synthetic) — REST
POST /api/v1/auth/mfa/recovery HTTP/1.1
Host: api.orion-example.com
Authorization: Bearer [FICTIONAL-TOKEN]
Content-Type: application/json
```

```
{ "request_id": "REQ-B-3302", "confirmation_code": "XXXX-3301" }
(code XXXX-3301 was issued for a separate, earlier recovery request REQ-A-3301)
```

RESPONSE (Fictional / Synthetic) — REST

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "request_id": "REQ-B-3302",
  "status": "confirmed",
  "mfa_reset": true
}

(Fictional observation: code XXXX-3301, issued for Request A, was accepted
against Request B — validation checked code validity/expiry only, not the
originating request_id.)
```

Expected: A confirmation code should be cryptographically bound to the single recovery-request that generated it and rejected for any other request.

Observed: The fictional API accepted the code against a differently-timestamped recovery request for the same account.

Impact: Weakens the integrity of the MFA-recovery control as a defense against account-takeover abuse.

TECHNICAL EVIDENCE SUMMARY

Attack Preconditions	An authenticated session on the target test account and timing overlap between two pending MFA-recovery requests.
Observed Result	Code validation checked only value and expiry, not request binding.
Security Impact	The confirmation-code validation checks only that the code is currently valid and unused, not that it matches the specific recovery-request ID it was issued for.
Business Impact	In the fictional test environment, a confirmation code generated for one pending MFA-recovery request was accepted when submitted against a second, unrelated pending request on the same test account.
Evidence Collected	EV-API-009 (fictional request/response timeline)
Retest Validation	Retest the full MFA-recovery workflow after remediation, including concurrent-request scenarios.

API-006 — REST File Authorization Bypass	**HIGH**
CVSS-Style Score: 7.1 (High) — Illustrative CVSS v3.1-inspired score	**CWE: CWE-862 — Missing Authorization**
CVSS Vector (Illustrative): `CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:N/A:L — Illustrative vector only`	
Affected API Type	REST
Affected Operation / Endpoint	*GET /api/v1/files/{id}*
Description	The file-retrieval endpoint issues a signed object-storage URL based solely on a valid file ID, without confirming that the requesting user's organization owns the referenced file.
Business Context	Files uploaded through reports and invoices may include sensitive fictional attachments scoped to a single organization.
Technical Impact	The download handler validates file-ID existence but not file-to-organization ownership against the requester's session.
Business Impact	A fictional file ID belonging to Test Tenant A was downloadable using a session authenticated to Test Tenant B, creating a fictional cross-tenant document-disclosure risk.
Attack Preconditions	A valid authenticated session and a known or enumerable file identifier belonging to another tenant.
Safe Proof of Concept (Summary)	*Illustrative cross-tenant file retrieval using a sequential fictional file ID. [ILLUSTRATIVE / FICTIONAL SAMPLE EVIDENCE]*
Evidence Reference	*EV-API-010 (redacted request/response, ownership-mismatch confirmation)*
Expected Behavior	Ownership check enforced server-side prior to signed-URL issuance.
Observed (Fictional) Behavior	No ownership check performed; only file-ID existence was validated.
Root Cause	Missing object-level authorization on the file-retrieval path, structurally similar to API-001.
Recommendation	Add a mandatory ownership check (file-to-organization) before signed-URL issuance across all file-serving endpoints.
Management Response	*Accepted; fictional storage-platform team will add ownership verification to the retrieval handler.*
Owner / Priority / Target / Status	**Platform Security Lead (Fictional Role)** | 31-60 Days | 28 November 2026 | **Open**
Retest Recommendation	*Retest file-retrieval and file-listing endpoints for ownership enforcement across all fictional test tenants.*

FULL EVIDENCE PANEL — ILLUSTRATIVE / FICTIONAL SAMPLE EVIDENCE

```
REQUEST (Fictional / Synthetic) — REST
GET /api/v1/files/FILE-51402 HTTP/1.1
Host: api.orion-example.com
Authorization: Bearer [FICTIONAL-TOKEN]
```

```
RESPONSE (Fictional / Synthetic) — REST
HTTP/1.1 302 Found
Location: https://storage.orion-example.test/objects/FILE-51402?sig=[FICTIONAL-SIGNED-URL]
```

Expected: The handler should verify the file's owning organization matches the requester's organization before issuing any signed URL.

Observed: A signed URL was issued for a fictional file owned by a different tenant.

Impact: Cross-tenant fictional document disclosure risk.

TECHNICAL EVIDENCE SUMMARY

Attack Preconditions	A valid authenticated session and a known or enumerable file identifier belonging to another tenant.
Observed Result	No ownership check performed; only file-ID existence was validated.
Security Impact	The download handler validates file-ID existence but not file-to-organization ownership against the requester's session.
Business Impact	A fictional file ID belonging to Test Tenant A was downloadable using a session authenticated to Test Tenant B, creating a fictional cross-tenant document-disclosure risk.
Evidence Collected	EV-API-010 (redacted request/response, ownership-mismatch confirmation)
Retest Validation	Retest file-retrieval and file-listing endpoints for ownership enforcement across all fictional test tenants.

API-007 — GraphQL Excessive Sensitive Data Exposure	**HIGH**
CVSS-Style Score: 7.1 (High) — Illustrative CVSS v3.1-inspired score	**CWE: CWE-213 — Exposure of Sensitive Information Due to Incompatible Policies**
CVSS Vector (Illustrative): `CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:L/A:N — Illustrative vector only`	
Affected API Type	GraphQL
Affected Operation / Endpoint	*query user(id) / users*
Description	GraphQL field resolvers on the `User` type return fields based on a broad, shared object serializer rather than role-aware field-level authorization, so an authenticated Standard User's query returns internal-only fields not shown in that role's UI.
Business Context	Standard Users should see only a limited user directory (name, role, status); billing identifiers and internal risk-scoring fields are intended for Manager and Administrator roles.
Technical Impact	Field-level authorization (excessive data exposure, an OWASP API Security Top-10 pattern) is enforced client-side only; the GraphQL schema exposes the full internal object graph to any authenticated caller who queries the fields directly.
Business Impact	A fictional Standard User's GraphQL response included billing-linkage identifiers and internal risk-scoring metadata not shown in that role's UI, creating a fictional internal-data exposure risk.
Attack Preconditions	Any authenticated session able to query the `user` or `users` field directly (bypassing the UI's field selection).
Safe Proof of Concept (Summary)	*Illustrative response comparison showing UI-rendered fields versus the full fictional GraphQL payload returned to a Standard User session. [ILLUSTRATIVE / FICTIONAL SAMPLE EVIDENCE]*
Evidence Reference	*EV-API-011 (redacted response payload comparison by role)*
Expected Behavior	Field-level, role-aware resolution applied server-side for every GraphQL field, not just object-level authorization.
Observed (Fictional) Behavior	Uniform full-object serialization regardless of requester role.
Root Cause	Absence of a field-level authorization layer in the GraphQL resolver chain for the User type.
Recommendation	Introduce role-aware field resolvers (or a field-authorization directive) for all sensitive fields on User, Organization, and Account types; audit remaining GraphQL types for similar excessive-exposure patterns.
Management Response	*Accepted; fictional API platform team will implement field-level authorization directives across the GraphQL schema.*
Owner / Priority / Target / Status	**API Platform Lead (Fictional Role)** | 31-60 Days | 28 November 2026 | Open
Retest Recommendation	*Retest all sensitive GraphQL fields across each of the seven defined roles to confirm field-level scoping.*

FULL EVIDENCE PANEL — ILLUSTRATIVE / FICTIONAL SAMPLE EVIDENCE

REQUEST (Fictional / Synthetic) — GraphQL

```
POST /graphql HTTP/1.1
Host: graphql.orion-example.com
Authorization: Bearer [FICTIONAL-TOKEN]
Content-Type: application/json

query {
  user(id: "USER-6102") {
    id
    name
    email
    role
    internalRiskScore
    billingAccountReference
    adminNotes
  }
}
```

RESPONSE (Fictional / Synthetic) — GraphQL

```
{
  "data": {
    "user": {
      "id": "USER-6102",
      "name": "Fictional User B1",
      "email": "fictional.b1@example.test",
      "role": "billing_user",
      "internalRiskScore": 14,
      "billingAccountReference": "BILL-REF-70021",
      "adminNotes": "Fictional internal note — not intended for member view"
    }
  }
}
```

Expected: The API should return a role-scoped field set matching what the requesting user is authorized to see, independent of front-end field selection.

Observed: All internal fields were present in the raw fictional GraphQL response regardless of requester role.

Impact: Exposure of internal billing linkage and internal-note metadata to unauthorized roles.

TECHNICAL EVIDENCE SUMMARY

Attack Preconditions	Any authenticated session able to query the `user` or `users` field directly (bypassing the UI's field selection).
Observed Result	Uniform full-object serialization regardless of requester role.
Security Impact	Field-level authorization (excessive data exposure, an OWASP API Security Top-10 pattern) is enforced client-side only; the GraphQL schema exposes the full internal object graph to any authenticated caller who queries the fields directly.
Business Impact	A fictional Standard User's GraphQL response included billing-linkage identifiers and internal risk-scoring metadata not shown in that role's UI, creating a fictional internal-data exposure risk.
Evidence Collected	EV-API-011 (redacted response payload comparison by role)
Retest Validation	Retest all sensitive GraphQL fields across each of the seven defined roles to confirm field-level scoping.

API-008 — GraphQL Nested Resolver Authorization Bypass	**HIGH**
CVSS-Style Score: 7.7 (High) — Illustrative CVSS v3.1-inspired score	**CWE: CWE-863 — Incorrect Authorization**
CVSS Vector (Illustrative): `CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:C/C:H/I:N/A:N — Illustrative vector only`	
Affected API Type	GraphQL
Affected Operation / Endpoint	*query organization → members → billing → invoices (nested)*
Description	The root `organization` query correctly authorizes access to the requester's own organization, but a nested resolver three levels deep (`organization.members.billing.invoices`) does not re-check tenant/role authorization and instead trusts the parent object's already-authorized context to resolve child invoice data that in fact belongs to a different fictional member record than expected.
Business Context	OrionSphere's GraphQL schema deliberately nests billing and invoice data under organization → members for a single efficient query; that convenience makes it easy to assume the root check protects every nested field, which this finding shows is not the case.

Technical Impact	The nested `invoices` resolver retrieves invoice records by member ID supplied deeper in the query tree without re-validating that the member ID is actually a child of the already-authorized root organization.
Business Impact	A crafted nested query allowed a fictional Standard User, authorized only at the root organization level, to retrieve invoice records for a member ID belonging to a different fictional organization by supplying it inside the nested selection.
Attack Preconditions	An authenticated session authorized for at least one organization, and the ability to submit a custom nested GraphQL query (not limited to the pre-built UI query shapes).
Safe Proof of Concept (Summary)	*Illustrative nested query showing the root organization check passing while a nested member/invoice resolver returns cross-tenant fictional invoice data. [ILLUSTRATIVE / FICTIONAL SAMPLE EVIDENCE]*
Evidence Reference	*EV-API-012, EV-API-013 (redacted nested-query capture, resolver-flow trace)*
Expected Behavior	Every resolver in a nested GraphQL query tree — not just the root — must independently re-validate tenant and role authorization for the specific object it resolves.
Observed (Fictional) Behavior	Only the root `organization` resolver performed an authorization check; child resolvers inherited trust from the parent without re-validation.
Root Cause	Authorization was implemented as a root-level guard rather than a per-resolver policy, an easy mistake in deeply nested GraphQL schemas.
Recommendation	Adopt a per-resolver (or per-type) authorization pattern — e.g., a GraphQL directive applied to every sensitive field — so nested resolvers cannot bypass tenant/role checks by relying on parent-level authorization.
Management Response	*Accepted as a high-priority architectural fix; fictional API platform team will introduce field-level authorization directives across all nested resolvers, starting with billing and invoices.*
Owner / Priority / Target / Status	**Director, API Platform (Fictional Role)** | 0-30 Days | 29 October 2026 | **Remediation In Progress**
Retest Recommendation	*Retest all nested query paths at every depth level (not just root-level queries) across all seven defined roles.*

FULL EVIDENCE PANEL — ILLUSTRATIVE / FICTIONAL SAMPLE EVIDENCE

REQUEST (Fictional / Synthetic) — GraphQL

```
POST /graphql HTTP/1.1
Host: graphql.orion-example.com
Authorization: Bearer [FICTIONAL-TOKEN]
Content-Type: application/json

query {
  organization(id: "ORG-2001") {
    id
    members {
      id
      billing(memberIdOverride: "USER-6102") {
        invoices { id status amount }
      }
    }
  }
}
```

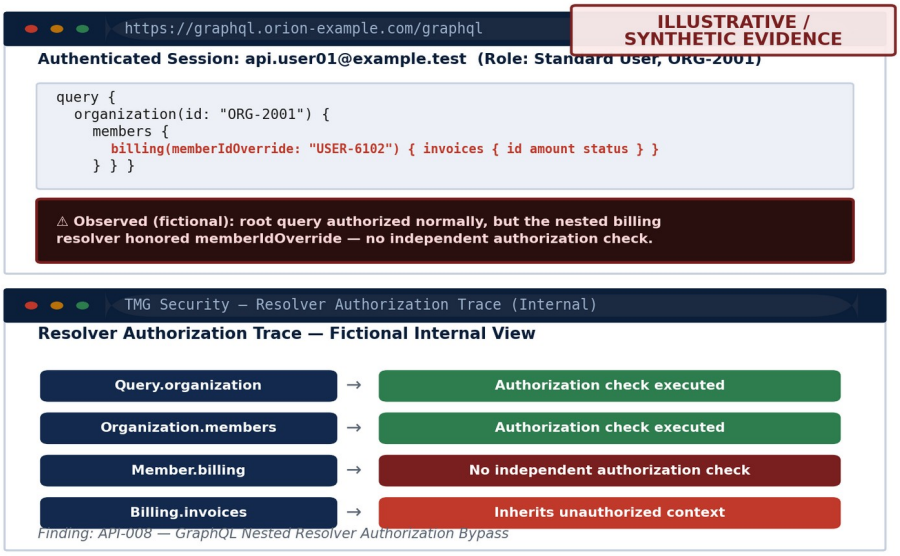
RESPONSE (Fictional / Synthetic) — GraphQL

```
{
  "data": {
    "organization": {
      "id": "ORG-2001",
      "members": [
        {
          "id": "USER-6001",
          "billing": {
            "invoices": [
              { "id": "INV-77411", "status": "issued", "amount": 4820.00 }
            ]
          }
        }
      ]
    }
  }
}

(Fictional observation: INV-77411 belongs to member USER-6102 of ORG-2002,
not to the requesting member USER-6001 of ORG-2001 — the nested "billing"
resolver honored the memberIdOverride argument without re-checking tenancy.)
```

Expected: The nested `billing` resolver should reject or ignore any member reference that does not belong to the already-authorized root organization.

Observed: The fictional resolver honored the nested override and returned invoice data for a member of a different tenant.

Impact: Demonstrates that GraphQL's nested-resolution model requires authorization to be re-validated at every level, not only at the query root.



Synthetic mock-UI / data evidence — ILLUSTRATIVE / SYNTHETIC EVIDENCE. No real application, user, or data is depicted.

TECHNICAL EVIDENCE SUMMARY

Attack Preconditions	An authenticated session authorized for at least one organization, and the ability to submit a custom nested GraphQL query (not limited to the pre-built UI query shapes).
Observed Result	Only the root `organization` resolver performed an authorization check; child resolvers inherited trust from the parent without re-validation.
Security Impact	The nested `invoices` resolver retrieves invoice records by member ID supplied deeper in the query tree without re-validating that the member ID is actually a child of the already-authorized root organization.
Business Impact	A crafted nested query allowed a fictional Standard User, authorized only at the root organization level, to retrieve invoice records for a member ID belonging to a different fictional organization by supplying it inside the nested selection.
Evidence Collected	EV-API-012, EV-API-013 (redacted nested-query capture, resolver-flow trace)
Retest Validation	Retest all nested query paths at every depth level (not just root-level queries) across all seven defined roles.

API-009 — Legacy API v1 Authorization Weakness	**HIGH**
CVSS-Style Score: 7.1 (High) — Illustrative CVSS v3.1-inspired score	**CWE: CWE-863 — Incorrect Authorization**
CVSS Vector (Illustrative): `CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:L/A:N — Illustrative vector only`	
Affected API Type	REST
Affected Operation / Endpoint	*GET /api/v1/accounts/{id} (legacy; superseded by /api/v2/accounts/{id})*
Description	The legacy v1 account-retrieval endpoint remains reachable after the introduction of the v2 API and enforces an older, weaker authorization check that the v2 equivalent does not share.
Business Context	OrionSphere migrated account retrieval to v2 with strengthened tenant-scoping, but the v1 route was never deprecated or authorization-hardened to match, leaving two logically equivalent endpoints with materially different security postures.
Technical Impact	The v1 handler authorizes on account-ID existence and a coarse organization-membership flag, while v2 additionally verifies role-appropriate field scoping and tenant ownership at the service layer; v1 was not updated when v2's authorization model was introduced.
Business Impact	A fictional test session was able to retrieve account detail through v1 that the equivalent v2 call correctly denied, demonstrating that the legacy route is an effective authorization downgrade path.
Attack Preconditions	Any authenticated session with knowledge that the deprecated v1 route remains reachable.
Safe Proof of Concept (Summary)	*Illustrative side-by-side comparison: the same fictional account ID is denied via v2 but returned via the legacy v1 route. [ILLUSTRATIVE / FICTIONAL SAMPLE EVIDENCE]*

Evidence Reference	*EV-API-014 (fictional v1/v2 response comparison)*
Expected Behavior	Deprecated API versions should either be retired or receive the same authorization hardening as their replacement, whichever is current for as long as both remain reachable.
Observed (Fictional) Behavior	The v1 route remained reachable with its original, weaker authorization model in place.
Root Cause	Deprecation and authorization-hardening were not tracked as a single migration unit; v1 was left running with backward-compatible but stale authorization logic.
Recommendation	Either retire the v1 account-retrieval route on a defined sunset date or back-port the v2 authorization model to it immediately; add an API-lifecycle gate that blocks new authorization changes from shipping to only one version.
Management Response	*Accepted; fictional platform team will set a v1 sunset date and back-port authorization hardening in the interim.*
Owner / Priority / Target / Status	**API Platform Lead (Fictional Role)** | 31-60 Days | 28 November 2026 | **Remediation Planned**
Retest Recommendation	*Retest all remaining v1 routes for authorization parity with v2, and confirm the sunset date is tracked to closure.*

FULL EVIDENCE PANEL — ILLUSTRATIVE / FICTIONAL SAMPLE EVIDENCE

REQUEST (Fictional / Synthetic) — REST

```
GET /api/v2/accounts/ACC-30021 HTTP/1.1
Host: api.orion-example.com
Authorization: Bearer [FICTIONAL-TOKEN]

GET /api/v1/accounts/ACC-30021 HTTP/1.1
Host: api.orion-example.com
Authorization: Bearer [FICTIONAL-TOKEN]
```

RESPONSE (Fictional / Synthetic) — REST

```
(v2) HTTP/1.1 403 Forbidden
{ "error": "not_authorized_for_account" }

(v1) HTTP/1.1 200 OK
{
  "account_id": "ACC-30021",
  "organization_id": "ORG-2002",
  "balance": 18420.00
}
```

Expected: Both API versions should apply the same authorization outcome for the same fictional account and requester.

Observed: v2 correctly denied the request; the legacy v1 route returned the fictional account detail for an unrelated tenant.

Impact: The legacy route provides an authorization-downgrade path around the hardened v2 controls.

TECHNICAL EVIDENCE SUMMARY

Attack Preconditions	Any authenticated session with knowledge that the deprecated v1 route remains reachable.
Observed Result	The v1 route remained reachable with its original, weaker authorization model in place.
Security Impact	The v1 handler authorizes on account-ID existence and a coarse organization-membership flag, while v2 additionally verifies role-appropriate field scoping and tenant ownership at the service layer; v1 was not updated when v2's authorization model was introduced.
Business Impact	A fictional test session was able to retrieve account detail through v1 that the equivalent v2 call correctly denied, demonstrating that the legacy route is an effective authorization downgrade path.
Evidence Collected	EV-API-014 (fictional v1/v2 response comparison)
Retest Validation	Retest all remaining v1 routes for authorization parity with v2, and confirm the sunset date is tracked to closure.

API-010 — Server-Side Request Forgery Through Integration Webhook API	**HIGH**
CVSS-Style Score: 8.1 (High) — Illustrative CVSS v3.1-inspired score	**CWE: CWE-918 — Server-Side Request Forgery (SSRF)**
CVSS Vector (Illustrative): `CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:N/A:H — Illustrative vector only`	
Affected API Type	REST
Affected Operation / Endpoint	*POST /api/v1/webhooks/validate*

Description	The webhook "validate" feature, which performs a server-side test request to a customer-supplied callback URL before saving a webhook subscription, does not restrict the destination beyond basic URL-format validation.
Business Context	Manager and API Integration User roles can register outbound webhook callback URLs for integration events; the validation feature is intended to confirm reachability before activation.
Technical Impact	No destination allow-listing, DNS-resolution restriction, or private-IP-range blocking was observed in the fictional test environment.
Business Impact	The fictional server was observed issuing an outbound request to an attacker-controlled observer endpoint supplied as the webhook URL, confirming the server performs the fetch itself rather than only validating URL syntax.
Attack Preconditions	An authenticated Manager or API Integration User session able to configure a webhook, and a network-reachable listener to observe the callback (a safe, attacker-controlled test endpoint in this illustrative case).
Safe Proof of Concept (Summary)	*A safe, non-destructive attacker-controlled test endpoint received a single fictional server-initiated request when submitted as the webhook target, confirming SSRF-class behavior without accessing any internal resource. [ILLUSTRATIVE / FICTIONAL SAMPLE EVIDENCE]*
Evidence Reference	*EV-API-015 (fictional observer-endpoint log, redacted request/response)*
Expected Behavior	Outbound validation requests should be restricted via an egress allow-list and should explicitly block RFC 1918 / link-local / metadata-service address ranges.
Observed (Fictional) Behavior	No such restriction was observed for the externally-routable test destination used in this illustrative assessment.
Root Cause	The webhook-validation feature performs a server-side fetch without a destination allow-list or private-range blocklist.
Recommendation	Implement an egress allow-list and explicit blocklist for private/link-local/metadata IP ranges on all server-initiated outbound fetches; route validation requests through an isolated, non-privileged network egress path.
Management Response	*Accepted as a priority item; fictional platform-security team will implement destination filtering on the webhook-validation service.*
Owner / Priority / Target / Status	**Platform Security Lead (Fictional Role)** | 0-30 Days | 29 October 2026 | **Remediation Planned**
Retest Recommendation	*Retest with a broader set of safe, non-internal illustrative destinations to confirm the allow-list/blocklist behaves as expected; internal-range testing remains out of scope per the rules of engagement.*

FULL EVIDENCE PANEL — ILLUSTRATIVE / FICTIONAL SAMPLE EVIDENCE

REQUEST (Fictional / Synthetic) — REST

```
POST /api/v1/webhooks/validate HTTP/1.1
Host: api.orion-example.com
Authorization: Bearer [FICTIONAL-TOKEN]
Content-Type: application/json

{ "callback_url": "https://observer.example.test/callback/447" }
```

RESPONSE (Fictional / Synthetic) — REST

```
HTTP/1.1 200 OK
Content-Type: application/json

{ "reachable": true, "latency_ms": 118 }

(Observer endpoint log — fictional)
INBOUND: GET /callback/447  Source: OrionSphere-Webhook-Validator/1.0
```

Expected: The validation feature should resolve and restrict outbound destinations to public, non-internal hosts, and should never be usable to reach link-local or private-IP-range addresses.

Observed: The fictional server made an outbound request to the supplied attacker-controlled test endpoint with no destination restriction observed for the tested case.

Impact: In a real deployment, this pattern could potentially be abused to probe internal network ranges or cloud metadata endpoints; this assessment did not attempt or demonstrate access to any internal or metadata address.

TECHNICAL EVIDENCE SUMMARY

Attack Preconditions	An authenticated Manager or API Integration User session able to configure a webhook, and a network-reachable listener to observe the callback (a safe, attacker-controlled test endpoint in this illustrative case).
Observed Result	No such restriction was observed for the externally-routable test destination used in this illustrative assessment.
Security Impact	No destination allow-listing, DNS-resolution restriction, or private-IP-range blocking was observed in the fictional test environment.
Business Impact	The fictional server was observed issuing an outbound request to an attacker-controlled observer endpoint supplied as the webhook URL, confirming the server performs the fetch itself rather than only validating URL syntax.
Evidence Collected	EV-API-015 (fictional observer-endpoint log, redacted request/response)

Retest Validation	Retest with a broader set of safe, non-internal illustrative destinations to confirm the allow-list/blocklist behaves as expected; internal-range testing remains out of scope per the rules of engagement.

API-011 — Privilege Escalation Through Role Mutation	**HIGH**
CVSS-Style Score: 7.6 (High) — Illustrative CVSS v3.1-inspired score	**CWE: CWE-269 — Improper Privilege Management**
CVSS Vector (Illustrative): `CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:L/I:H/A:L — Illustrative vector only`	
Affected API Type	GraphQL
Affected Operation / Endpoint	*mutation updateMemberRole*
Description	The `updateMemberRole` mutation checks that the requesting user is authenticated and belongs to the organization, but does not additionally verify that the requester's own role is authorized to grant the requested target role.
Business Context	Role assignment is a privileged administrative action; a Standard User granting themselves or another member the Manager role bypasses the intended approval chain.
Technical Impact	Role-change authorization is enforced only at the organization-membership level, not the requester's-own-role level.
Business Impact	A fictional Standard User account successfully updated its own role to Manager, exposing organization-management, billing, and member-management functions to a former low-privilege user.
Attack Preconditions	An authenticated Standard User session within the target organization.
Safe Proof of Concept (Summary)	*Illustrative role-update mutation submitted by a Standard User targeting their own membership record, successfully elevating to Manager in the fictional test environment. [ILLUSTRATIVE / FICTIONAL SAMPLE EVIDENCE]*
Evidence Reference	*EV-API-016 (redacted request/response, before/after role-state capture)*
Expected Behavior	Role-update mutations should be authorized based on the requester's current role and should reject any role-change request targeting the requester's own account.
Observed (Fictional) Behavior	No requester-role authorization check was applied.
Root Cause	Missing function-level access control on a privileged administrative mutation.
Recommendation	Add explicit requester-role authorization to the role-update resolver, and disallow self-targeted role changes entirely; log all role-change events for audit review.
Management Response	*Accepted; fictional identity-platform team will add requester-role validation and self-elevation blocking.*
Owner / Priority / Target / Status	**Identity & Access Lead (Fictional Role)** | 0-30 Days | 29 October 2026 | **Remediation In Progress**
Retest Recommendation	*Retest role-management operations across all seven defined roles to confirm correct requester-role enforcement in both directions.*

FULL EVIDENCE PANEL — ILLUSTRATIVE / FICTIONAL SAMPLE EVIDENCE

REQUEST (Fictional / Synthetic) — GraphQL

```
POST /graphql HTTP/1.1
Host: graphql.orion-example.com
Authorization: Bearer [FICTIONAL-TOKEN]
Content-Type: application/json

mutation {
  updateMemberRole(orgId: "ORG-2001", userId: "USER-6001", role: "manager") {
    id
    role
  }
}
```

RESPONSE (Fictional / Synthetic) — GraphQL

```
{
  "data": {
    "updateMemberRole": { "id": "USER-6001", "role": "manager" }
  }
}
```

Expected: Only an existing Manager or Administrator should be authorized to change a member's role, and a user should never be able to elevate their own role.

Observed: The fictional mutation accepted the self-elevation request without a requester-role authorization check.

Impact: Full vertical privilege escalation within the affected fictional organization.

TECHNICAL EVIDENCE SUMMARY

Attack Preconditions	An authenticated Standard User session within the target organization.
Observed Result	No requester-role authorization check was applied.
Security Impact	Role-change authorization is enforced only at the organization-membership level, not the requester's-own-role level.
Business Impact	A fictional Standard User account successfully updated its own role to Manager, exposing organization-management, billing, and member-management functions to a former low-privilege user.
Evidence Collected	EV-API-016 (redacted request/response, before/after role-state capture)
Retest Validation	Retest role-management operations across all seven defined roles to confirm correct requester-role enforcement in both directions.

47 MEDIUM FINDINGS

The following 12 fictional Medium-severity findings reflect defense-in-depth and GraphQL-specific hardening gaps recommended for remediation within the 31-60 day window of the illustrative roadmap (Section 53).

API-012 — GraphQL Introspection Exposes Privileged Administrative Operations	**MEDIUM**
CVSS-Style Score: 6.5 (Medium) — Illustrative CVSS v3.1-inspired score	**CWE: CWE-200 — Exposure of Sensitive Information to an Unauthorized Actor**
Affected API Type	GraphQL
Affected Operation / Endpoint	*GraphQL introspection query (__schema)*
Description	GraphQL introspection is enabled in the assessed fictional environment and, beyond ordinary schema discovery, reveals the names and argument shapes of several administrative-only mutations and internal object types not referenced anywhere in the public-facing client application.
Business Context	Introspection being enabled is not itself a control failure — many APIs intentionally expose it — but the specific administrative mutation names and internal type names it discloses here narrow an attacker's reconnaissance effort considerably.
Technical Impact	Standard introspection response includes `AdminAuditQuery`, `internalRiskScore`, and other administration-only schema elements alongside ordinary customer-facing types, with no schema-visibility segmentation by caller role.
Business Impact	Reduced reconnaissance cost for identifying high-value administrative attack surface; distinct from, but related to, the exposure/impact distinction called out in Section 24.
Safe Proof of Concept (Summary)	*Fictional introspection query returned a safe synthetic schema excerpt including administrative-only mutation and type names. [ILLUSTRATIVE / FICTIONAL SAMPLE EVIDENCE]*
Evidence Reference	*EV-API-017 (fictional introspection response excerpt)*
Expected Behavior	Either disable introspection in production, or segment the exposed schema so administrative-only types and mutations are not visible to introspection queries issued by non-administrative credentials.
Observed (Fictional) Behavior	The full schema, including administrative-only elements, was returned to any authenticated introspection query.
Root Cause	Introspection is enabled platform-wide with no role-based schema visibility segmentation.
Recommendation	Adopt schema segmentation (or disable introspection in production) so only the fields and operations relevant to the caller's role are discoverable; retain full introspection in non-production environments for developer tooling.
Management Response	*Accepted; fictional API platform team will evaluate schema segmentation for production introspection responses.*
Owner / Priority / Target / Status	**API Platform Lead (Fictional Role)** | 31-60 Days | 28 November 2026 | **Open**
Retest Recommendation	*Retest introspection responses across roles to confirm administrative-only elements are no longer disclosed.*

API-013 — GraphQL Query Complexity Enforcement Gap	**MEDIUM**
CVSS-Style Score: 6.1 (Medium) — Illustrative CVSS v3.1-inspired score	**CWE: CWE-400 — Uncontrolled Resource Consumption**
Affected API Type	GraphQL
Affected Operation / Endpoint	*POST /graphql (all nested query operations)*
Description	The GraphQL API lacks effective query-complexity enforcement; a harmless synthetic query with several nested relationships was accepted without a complexity-based rejection.
Business Context	Query-complexity limits are a standard GraphQL defense-in-depth control against resource-consumption risk; their absence increases the platform's exposure to resource-intensive query patterns.
Technical Impact	No complexity-scoring middleware was observed rejecting deeply nested or field-heavy fictional queries below an extreme threshold.
Business Impact	Resource-consumption risk from unbounded nested query patterns, illustrated here only with a safe, single, harmless test query — no resource-exhaustion automation was performed.
Safe Proof of Concept (Summary)	*A single safe synthetic nested query (Query Depth: 8, Estimated Complexity: 240) was accepted with no complexity-based rejection. [ILLUSTRATIVE / FICTIONAL SAMPLE EVIDENCE]*

Evidence Reference	*EV-API-018 (fictional query-complexity test log — depth 8, estimated complexity 240, safe test threshold not enforced)*
Expected Behavior	Queries exceeding a defined complexity budget should be rejected with a clear error before resolver execution begins.
Observed (Fictional) Behavior	The fictional test query was accepted and executed with no complexity-based rejection at the observed depth/complexity values.
Root Cause	No query-complexity analysis middleware is configured ahead of the resolver execution pipeline.
Recommendation	Introduce a query-complexity scoring middleware (cost-per-field plus depth multiplier) with a documented budget, rejecting queries that exceed it before resolver execution.
Management Response	*Accepted; fictional API platform team will evaluate and configure a complexity-scoring middleware.*
Owner / Priority / Target / Status	**API Platform Lead (Fictional Role)** | 31-60 Days | 28 November 2026 | **Remediation Planned**
Retest Recommendation	*Retest with a range of safe synthetic complexity values to confirm the enforced budget behaves as expected.*

API-014 — GraphQL Query Depth Control Gap	**MEDIUM**
CVSS-Style Score: 5.9 (Medium) — Illustrative CVSS v3.1-inspired score	**CWE: CWE-400 — Uncontrolled Resource Consumption**
Affected API Type	GraphQL
Affected Operation / Endpoint	*POST /graphql (recursive relationship traversal)*
Description	Related to, but distinct from, the query-complexity gap in API-013: the API does not separately enforce a maximum query-nesting depth, which recursive relationships in the schema (organization → members → organization → members…) can otherwise exploit independent of raw field-complexity scoring.
Business Context	Depth limits address a different resource-consumption pattern than complexity scoring — a narrow-but-deep query can be cheap by field count yet expensive by recursive evaluation.
Technical Impact	No maximum-depth validator was observed rejecting a fictional test query that recursed through a self-referencing relationship several levels deep.
Business Impact	Resource-consumption risk from unbounded recursive traversal of self-referencing relationships, again illustrated only with a single safe test query.
Safe Proof of Concept (Summary)	*A fictional recursive test query traversing a self-referencing relationship to a depth of 8 was accepted without a depth-limit rejection. [ILLUSTRATIVE / FICTIONAL SAMPLE EVIDENCE]*
Evidence Reference	*EV-API-019 (fictional query-depth test log)*
Expected Behavior	A maximum query-nesting depth should be enforced independently of complexity scoring, particularly for self-referencing relationship types.
Observed (Fictional) Behavior	No depth-limit rejection was observed for the tested fictional recursive query.
Root Cause	Depth limiting is not implemented as a distinct control from complexity scoring.
Recommendation	Add an explicit maximum-depth validator (recommended: depth 6-8 for this schema) independent of complexity scoring, applied before resolver execution.
Management Response	*Accepted; will be implemented alongside the API-013 complexity-scoring middleware.*
Owner / Priority / Target / Status	**API Platform Lead (Fictional Role)** | 31-60 Days | 28 November 2026 | **Remediation Planned**
Retest Recommendation	*Retest recursive relationship traversal at varying depths to confirm the enforced limit.*

API-015 — GraphQL Batching / Alias-Based Excessive Object Retrieval	**MEDIUM**
CVSS-Style Score: 5.8 (Medium) — Illustrative CVSS v3.1-inspired score	**CWE: CWE-799 — Improper Control of Interaction Frequency**
Affected API Type	GraphQL
Affected Operation / Endpoint	*POST /graphql (aliased multi-object queries)*
Description	GraphQL query aliasing allows multiple logically distinct object lookups to be combined into a single HTTP request, which is not accounted for by the platform's per-request rate-limiting model.
Business Context	Per-request rate limiting assumes roughly one object lookup per request; aliasing breaks that assumption by letting a single request perform many lookups at once.
Technical Impact	The rate limiter counts HTTP requests, not the number of aliased object lookups contained within a single GraphQL

	request body.
Business Impact	A fictional test request using three aliased user lookups in a single call counted as only one request against the rate limit, illustrating (without extracting bulk data) how aliasing could be combined with automation to exceed the intended per-object lookup rate.
Safe Proof of Concept (Summary)	*A fictional query using three aliased, synthetic user lookups in a single request was accepted and counted as a single rate-limited request. [ILLUSTRATIVE / FICTIONAL SAMPLE EVIDENCE]*
Evidence Reference	*EV-API-020 (fictional aliased-query rate-limit accounting log)*
Expected Behavior	Rate-limiting accounting should count the number of resolved objects/fields within a GraphQL request, not just the HTTP request itself.
Observed (Fictional) Behavior	The fictional aliased request was accounted as a single unit against the rate limit regardless of the number of aliased lookups it contained.
Root Cause	Rate-limiting middleware operates at the HTTP-request layer, ahead of GraphQL query parsing, and has no visibility into aliased field counts.
Recommendation	Extend rate-limiting accounting to the GraphQL layer, counting resolved object lookups (including aliased duplicates) rather than raw HTTP requests; combine with the complexity-scoring work from API-013.
Management Response	*Accepted; fictional API platform team will extend rate-limit accounting into the GraphQL execution layer.*
Owner / Priority / Target / Status	**API Platform Lead (Fictional Role)** | 31-60 Days | 28 November 2026 | **Open**
Retest Recommendation	*Retest with a range of safe synthetic alias counts to confirm accounting reflects resolved object count.*

API-016 — Insufficient Rate Limiting on Authentication Endpoints	**MEDIUM**
CVSS-Style Score: 5.9 (Medium) — Illustrative CVSS v3.1-inspired score	**CWE: CWE-307 — Improper Restriction of Excessive Authentication Attempts**
Affected API Type	REST
Affected Operation / Endpoint	*POST /api/v1/auth/login, POST /api/v1/auth/mfa*
Description	The login and MFA-verification endpoints permit a high fictional volume of requests for the same account within a short window before any throttling was observed.
Business Context	Authentication-endpoint request volume is a common target for credential-stuffing and brute-force-style abuse.
Technical Impact	In the fictional test environment, 50 sequential login requests for the same test account completed without a rate-limit response.
Business Impact	Increased exposure to fictional credential-stuffing-style abuse patterns if this gap existed against a real deployment.
Safe Proof of Concept (Summary)	*A controlled, fictional burst of 50 login requests against a single test account completed without an application-level rate-limit response. [ILLUSTRATIVE / FICTIONAL SAMPLE EVIDENCE]*
Evidence Reference	*EV-API-021 (fictional request-count/response-code table)*
Expected Behavior	Application-level rate limiting (per-account and per-source-IP) should trigger well before 50 requests.
Observed (Fictional) Behavior	No application-level throttling response was observed within the tested fictional volume.
Root Cause	Rate limiting for these endpoints is not implemented at the application layer.
Recommendation	Implement per-account and per-IP rate limiting with exponential backoff on all authentication endpoints.
Management Response	*Accepted; fictional identity-platform team will extend rate-limiting middleware to authentication endpoints.*
Owner / Priority / Target / Status	**Identity & Access Lead (Fictional Role)** | 31-60 Days | 28 November 2026 | **Remediation Planned**
Retest Recommendation	*Retest with controlled fictional request volumes to confirm throttling activates at an appropriate threshold.*

API-017 — Access Token Revocation Weakness on Logout	**MEDIUM**
CVSS-Style Score: 6.3 (Medium) — Illustrative CVSS v3.1-inspired score	**CWE: CWE-613 — Insufficient Session Expiration**
Affected API Type	REST
Affected Operation / Endpoint	*POST /api/v1/auth/logout*
Description	Logout invalidates the refresh token but the short-lived access token issued to the client remains fictionally valid until its natural expiry, rather than being actively revoked.

Business Context	Users reasonably expect logout to immediately end their session on a given device, particularly on shared or lost devices.
Technical Impact	A fictional access token captured before logout remained usable against protected endpoints for the remainder of its ~15-minute fictional lifetime after logout.
Business Impact	Extended fictional exposure window for a captured access token following an explicit logout event.
Safe Proof of Concept (Summary)	*A fictional access token remained accepted by protected endpoints for a short window following an explicit logout call. [ILLUSTRATIVE / FICTIONAL SAMPLE EVIDENCE]*
Evidence Reference	*EV-API-022 (fictional token-lifetime timeline)*
Expected Behavior	Logout should revoke or blacklist the current access token in addition to the refresh token.
Observed (Fictional) Behavior	Only the refresh token was invalidated; the access token remained valid until natural expiry.
Root Cause	The token architecture relies on statelessness for the access token with no revocation-list check.
Recommendation	Introduce a lightweight server-side revocation check (short-TTL denylist) for access tokens on logout, or reduce access-token lifetime further to shrink the exposure window.
Management Response	*Accepted as a defense-in-depth improvement; fictional platform team will evaluate a revocation-list approach.*
Owner / Priority / Target / Status	**Platform Security Lead (Fictional Role)** | 31-60 Days | 28 November 2026 | **Open**
Retest Recommendation	*Retest logout behavior and confirm access-token rejection immediately following logout.*

API-018 — Mass Assignment via REST and GraphQL Update Operations	**MEDIUM**
CVSS-Style Score: 6.1 (Medium) — Illustrative CVSS v3.1-inspired score	**CWE: CWE-915 — Improperly Controlled Modification of Dynamically-Determined Object Attributes**
Affected API Type	REST & GraphQL
Affected Operation / Endpoint	*PATCH /api/v1/users/{id} · GraphQL mutation updateUser*
Description	Both the REST user-update endpoint and the equivalent GraphQL `updateUser` mutation bind the full request payload to the underlying data model, allowing fields not exposed in the UI (such as `riskTier` or `adminFlag`) to be modified if included in the request.
Business Context	Internal-only fields such as risk classification and admin flags are intended to be set exclusively by internal review processes, not by ordinary user-update calls.
Technical Impact	Both handlers use a generic object-binding pattern without an explicit allow-list, so any field present on the underlying model can be set via either interface.
Business Impact	A fictional test request including an undocumented `riskTier` field successfully modified that internal value alongside the legitimate profile fields, via both REST and GraphQL.
Safe Proof of Concept (Summary)	*A fictional profile-update request containing an additional undocumented field was accepted and applied via both REST and GraphQL. [ILLUSTRATIVE / FICTIONAL SAMPLE EVIDENCE]*
Evidence Reference	*EV-API-023 (fictional before/after object-state comparison, REST and GraphQL)*
Expected Behavior	Update handlers should bind only an explicit allow-list of client-editable fields to the underlying model, identically across REST and GraphQL.
Observed (Fictional) Behavior	Both handlers bound the entire request payload, including non-exposed internal fields.
Root Cause	Use of a generic object-binding pattern without an explicit field allow-list on a privileged data model, present in both transport layers.
Recommendation	Replace generic body-binding with an explicit allow-listed update DTO (REST) and input type (GraphQL) for every write operation that touches an object containing internal-only fields.
Management Response	*Accepted; fictional API platform team will introduce allow-listed update models across both REST and GraphQL write operations.*
Owner / Priority / Target / Status	**API Platform Lead (Fictional Role)** | 31-60 Days | 28 November 2026 | **Open**
Retest Recommendation	*Retest all profile/object update operations for mass-assignment exposure of internal-only fields, across both interfaces.*

API-019 — Webhook Replay Protection Missing	**MEDIUM**
CVSS-Style Score: 5.7 (Medium) — Illustrative CVSS v3.1-inspired score	**CWE: CWE-345 — Insufficient Verification of Data Authenticity**

Affected API Type	REST
Affected Operation / Endpoint	*POST (customer-hosted webhook receiver, outbound event delivery)*
Description	Outbound webhook event payloads are signed, but the signature does not incorporate a timestamp or nonce, and the fictional receiver reference implementation does not reject a captured-and-resent (replayed) event.
Business Context	Customers build automation on webhook events; a replayed event could cause duplicate downstream processing if the customer's own receiver does not independently de-duplicate.
Technical Impact	A fictional captured webhook payload, when resent unmodified, passed signature validation a second time because the signature covers only the payload body, not a timestamp or nonce.
Business Impact	Potential duplicate downstream processing (e.g., duplicate fictional transaction-event triggers) for customers whose own receivers do not independently de-duplicate.
Safe Proof of Concept (Summary)	*A captured fictional webhook payload replayed unmodified passed signature validation on resend. [ILLUSTRATIVE / FICTIONAL SAMPLE EVIDENCE]*
Evidence Reference	*EV-API-024 (fictional signature-validation replay test)*
Expected Behavior	Signatures should incorporate a timestamp/nonce, with the platform documenting a recommended freshness window and duplicate-event-ID check for receivers.
Observed (Fictional) Behavior	Signature validation covered payload content only, with no timestamp or nonce component.
Root Cause	Webhook signing scheme does not include replay-protection elements.
Recommendation	Add a timestamp and nonce (or monotonic event ID) to the signed payload envelope, document a freshness-window check, and provide a reference receiver implementation demonstrating replay rejection.
Management Response	*Accepted; fictional integrations team will update the webhook signing scheme and reference documentation.*
Owner / Priority / Target / Status	**Integrations Platform Lead (Fictional Role)** | 31-60 Days | 28 November 2026 | **Remediation Planned**
Retest Recommendation	*Retest webhook signature validation for replay rejection after the timestamp/nonce scheme is introduced.*

API-020 — CORS Configuration Trusts Unnecessary Origins	**MEDIUM**
CVSS-Style Score: 5.4 (Medium) — Illustrative CVSS v3.1-inspired score	**CWE: CWE-942 — Permissive Cross-domain Policy with Untrusted Domains**
Affected API Type	REST & GraphQL
Affected Operation / Endpoint	*All /api/v1/* and /graphql endpoints*
Description	The API's CORS configuration reflects the request's Origin header for a broader set of fictional subdomains than the application actually uses, including several fictional preview/staging patterns.
Business Context	An overly broad CORS allow-list increases the number of origins from which authenticated cross-origin requests could be attempted if any of those origins were ever compromised or repurposed.
Technical Impact	The fictional API reflected `Access-Control-Allow-Origin` for a wildcard-pattern staging domain not currently in active use, with `Access-Control-Allow-Credentials: true` also set.
Business Impact	Broader-than-necessary attack surface for any future cross-origin credentialed-request abuse.
Safe Proof of Concept (Summary)	*A fictional cross-origin request from a stale staging-pattern origin received a permissive CORS response including credentialed-request support. [ILLUSTRATIVE / FICTIONAL SAMPLE EVIDENCE]*
Evidence Reference	*EV-API-025 (fictional CORS response header capture)*
Expected Behavior	CORS allow-list should be a minimal, explicit set of current production front-end origins.
Observed (Fictional) Behavior	A broader wildcard-pattern allow-list was in effect, including unused staging patterns.
Root Cause	CORS configuration was not pruned as staging/preview environments were decommissioned.
Recommendation	Reduce the CORS allow-list to the minimal explicit set of active origins and remove wildcard pattern-matching for credentialed requests.
Management Response	*Accepted; fictional platform team will audit and tighten the CORS configuration.*
Owner / Priority / Target / Status	**Platform Security Lead (Fictional Role)** | 31-60 Days | 28 November 2026 | **Remediation Planned**
Retest Recommendation	*Retest CORS response headers against the pruned allow-list from a range of fictional test origins.*

API-021 — Verbose GraphQL Error Responses Reveal Internal Details	**MEDIUM**

API-022 — API Versioning Weakness — Inconsistent Deprecation Enforcement	MEDIUM
CVSS-Style Score: 5.0 (Medium) — Illustrative CVSS v3.1-inspired score	**CWE: CWE-1059 — Insufficient Technical Documentation**
Affected API Type	REST
Affected Operation / Endpoint	*Multiple /api/v1/* routes marked deprecated in documentation*
Description	Several REST routes marked "deprecated" in the fictional API documentation remain fully functional with no deprecation warning header, sunset date enforcement, or reduced support posture observed in practice.
Business Context	Consistent deprecation lifecycle management (this finding) is distinct from the specific v1/v2 authorization gap already covered in API-009; this finding is about lifecycle process, that one about a concrete authorization regression.
Technical Impact	No `Deprecation` or `Sunset` HTTP headers were observed on any of the sampled fictional deprecated routes, and none returned a warning in their response body.
Business Impact	Reduced ability for API consumers (and TMG's own retest process) to distinguish actively-supported endpoints from those pending retirement.
Safe Proof of Concept (Summary)	*Fictional sampled requests to documentation-marked-deprecated routes returned normal 200 responses with no deprecation signaling headers. [ILLUSTRATIVE / FICTIONAL SAMPLE EVIDENCE]*
Evidence Reference	*EV-API-027 (fictional deprecated-route header sample)*
Expected Behavior	Deprecated routes should return standard `Deprecation`/`Sunset` headers and be tracked against an explicit retirement date.
Observed (Fictional) Behavior	No deprecation signaling was present on any sampled fictional deprecated route.
Root Cause	API lifecycle governance process does not yet enforce deprecation signaling as a release gate.
Recommendation	Add `Deprecation`/`Sunset` header emission for all documented-deprecated routes and track each against a governance-approved retirement date.
Management Response	*Accepted; fictional API platform team will add deprecation-header middleware and a lifecycle tracking process.*
Owner / Priority / Target / Status	**API Platform Lead (Fictional Role)** | 31-60 Days | 28 November 2026 | **Open**
Retest Recommendation	*Retest deprecated-route headers and confirm each has a tracked retirement date.*

API-023 — Business Logic Replay in Discount Redemption Workflow	MEDIUM
CVSS-Style Score: 5.8 (Medium) — Illustrative CVSS v3.1-inspired score	**CWE: CWE-841 — Improper Enforcement of Behavioral Workflow**
Affected API Type	REST
Affected Operation / Endpoint	*POST /api/v1/subscriptions/{id}/discount/redeem*
Description	A single-use discount code was, in the fictional test environment, capable of being redeemed a second time on the same test organization when the redemption request was resent shortly after the first, due to a narrow validation window.
Business Context	Single-use promotional discounts are relied upon for controlled revenue-impact promotions; repeat redemption undermines that control.
Technical Impact	Two rapidly-sequential fictional redemption requests for the same single-use code both returned success in the test environment, applying the discount twice.
Business Impact	Fictional revenue-integrity impact from duplicated discount application.
Safe Proof of Concept (Summary)	*Two fictional rapid-sequence redemption requests for the same single-use code both succeeded in the test environment. [ILLUSTRATIVE / FICTIONAL SAMPLE EVIDENCE]*
Evidence Reference	*EV-API-028 (fictional redemption-count/state log)*
Expected Behavior	Redemption state should be checked and updated atomically so a second concurrent or rapid-sequence request for the same code is rejected.
Observed (Fictional) Behavior	A narrow window between the validity check and the state update allowed a second success in the fictional test.
Root Cause	Non-atomic check-then-update pattern in the discount-redemption handler.
Recommendation	Enforce atomic, database-level uniqueness/locking on redemption-state updates for single-use codes.

Management Response	*Accepted; fictional billing-platform team will introduce an atomic redemption-state transition.*
Owner / Priority / Target / Status	**Director, Billing Platform (Fictional Role)** | 31-60 Days | 28 November 2026 | **Open**
Retest Recommendation	*Retest with controlled concurrent-request scenarios to confirm single-use enforcement.*

48 LOW FINDINGS

The following 9 fictional Low-severity findings are hardening opportunities recommended for the 61-90 day window of the illustrative roadmap.

API-024 — Technology / Version Disclosure via Response Headers	LOW
CVSS-Style Score: 3.1 (Low) — Illustrative CVSS v3.1-inspired score	**CWE: CWE-200 — Exposure of Sensitive Information to an Unauthorized Actor**
Affected API Type	REST & GraphQL
Affected Operation / Endpoint	*All application responses (Server / X-Powered-By headers)*
Description	Fictional response headers disclosed the specific backend framework version in use across both REST and GraphQL endpoints.
Business Context	Version disclosure narrows the search space for an attacker researching known issues in a specific framework version.
Technical Impact	Fictional header capture showed a specific framework version string on every sampled response.
Business Impact	Marginal reconnaissance benefit to a would-be attacker; no direct exploitation demonstrated.
Safe Proof of Concept (Summary)	*Fictional header capture across sampled endpoints consistently disclosed backend framework version. [ILLUSTRATIVE / FICTIONAL SAMPLE EVIDENCE]*
Evidence Reference	*EV-API-029 (fictional header sample)*
Expected Behavior	Version-identifying headers suppressed or genericized in production responses.
Observed (Fictional) Behavior	Version string present on all sampled fictional responses.
Root Cause	Default framework configuration was not hardened to suppress version headers.
Recommendation	Disable or genericize version-disclosing headers at the framework/edge layer.
Management Response	*Accepted as a hardening item for the defense-in-depth backlog.*
Owner / Priority / Target / Status	**Platform Engineering Lead (Fictional Role)** | 61-90 Days | 13 December 2026 | **Open**
Retest Recommendation	*Confirm header suppression on retest.*

API-025 — Missing Security Header Hardening (Permissions-Policy / Referrer-Policy)	LOW
CVSS-Style Score: 2.6 (Low) — Illustrative CVSS v3.1-inspired score	**CWE: CWE-693 — Protection Mechanism Failure**
Affected API Type	REST
Affected Operation / Endpoint	*All application front-end responses*
Description	No explicit Permissions-Policy or Referrer-Policy headers were observed, leaving browser defaults in effect across the fictional application.
Business Context	These headers are a defense-in-depth layer reducing incidental data leakage and restricting unused browser feature access.
Technical Impact	Fictional header capture showed no explicit Permissions-Policy or Referrer-Policy value set.
Business Impact	Minor defense-in-depth gap; no direct exploitation demonstrated.
Safe Proof of Concept (Summary)	*Fictional header capture confirmed both headers were absent across sampled pages. [ILLUSTRATIVE / FICTIONAL SAMPLE EVIDENCE]*
Evidence Reference	*EV-API-030 (fictional header sample)*
Expected Behavior	Explicit Permissions-Policy and `strict-origin-when-cross-origin` (or stricter) Referrer-Policy applied platform-wide.
Observed (Fictional) Behavior	No explicit policy set for either header; browser defaults in effect.
Root Cause	Not yet included in the centralized security-header baseline.
Recommendation	Add Permissions-Policy and Referrer-Policy to the centralized security-header rollout.
Management Response	*Accepted; will be bundled into the centralized header-hardening rollout.*

Owner / Priority / Target / Status	**Platform Engineering Lead (Fictional Role)** | 61-90 Days | 13 December 2026 | **Remediation Planned**
Retest Recommendation	*Confirm header presence on retest.*

API-026 — Unused / Unnecessary HTTP Methods Enabled	**LOW**
CVSS-Style Score: 3.0 (Low) — Illustrative CVSS v3.1-inspired score	**CWE: CWE-16 — Configuration**
Affected API Type	REST
Affected Operation / Endpoint	*Application-wide HTTP method handling*
Description	Several fictional REST endpoints accept the `TRACE` and `OPTIONS` HTTP methods without an explicit, deliberate handler, returning default framework behavior rather than a defined response.
Business Context	Unused HTTP methods left in their framework-default state are a minor attack-surface and configuration-hygiene item.
Technical Impact	Fictional method-enumeration testing found `TRACE` enabled on a subset of sampled endpoints with default framework behavior.
Business Impact	Minor configuration-hygiene gap; no direct exploitation demonstrated.
Safe Proof of Concept (Summary)	*Fictional HTTP method enumeration found TRACE enabled by default on a subset of endpoints. [ILLUSTRATIVE / FICTIONAL SAMPLE EVIDENCE]*
Evidence Reference	*EV-API-031 (fictional method-enumeration table)*
Expected Behavior	Unused HTTP methods explicitly disabled at the edge/WAF or application layer.
Observed (Fictional) Behavior	Default framework behavior in effect for unused methods on a subset of endpoints.
Root Cause	No explicit method allow-list configured at the edge layer.
Recommendation	Configure an explicit HTTP method allow-list at the CDN/WAF edge layer, disabling TRACE and unused methods platform-wide.
Management Response	*Accepted as a hardening item for the edge configuration backlog.*
Owner / Priority / Target / Status	**Platform Security Lead (Fictional Role)** | 61-90 Days | 13 December 2026 | **Open**
Retest Recommendation	*Confirm method allow-list enforcement on retest.*

API-027 — Minor Input Validation Inconsistencies Across Endpoints	**LOW**
CVSS-Style Score: 3.4 (Low) — Illustrative CVSS v3.1-inspired score	**CWE: CWE-20 — Improper Input Validation**
Affected API Type	REST & GraphQL
Affected Operation / Endpoint	*Multiple endpoints (field length / format constraints)*
Description	A handful of fictional fields (e.g., organization display name, webhook description) enforce different maximum-length and character-set constraints between their REST and GraphQL entry points, though both ultimately write to the same underlying field.
Business Context	Inconsistent validation is primarily a data-quality and consistency concern rather than a direct security control gap, though it can occasionally mask a stricter server-side invariant being bypassed via the looser path.
Technical Impact	Fictional field-length tests showed the GraphQL mutation accepting up to 20 additional characters beyond the REST endpoint's documented limit for the same underlying field.
Business Impact	Minor data-consistency risk; no security control bypass demonstrated beyond the length/format discrepancy itself.
Safe Proof of Concept (Summary)	*Fictional side-by-side field-length tests showed differing enforcement between REST and GraphQL for the same underlying field. [ILLUSTRATIVE / FICTIONAL SAMPLE EVIDENCE]*
Evidence Reference	*EV-API-032 (fictional field-validation comparison table)*
Expected Behavior	Field-level validation constraints should be defined once (e.g., in a shared schema/validation layer) and enforced identically across REST and GraphQL.
Observed (Fictional) Behavior	Validation constraints diverged between the two entry points for several fictional fields.
Root Cause	REST and GraphQL input validation are implemented independently rather than sharing a single validation definition.
Recommendation	Centralize field-level validation rules in a single shared schema/validation layer consumed by both REST and GraphQL

	handlers.
Management Response	*Accepted as a data-quality and consistency hardening item.*
Owner / Priority / Target / Status	**Platform Engineering Lead (Fictional Role)** | 61-90 Days | 13 December 2026 | **Open**
Retest Recommendation	*Confirm aligned validation behavior across REST and GraphQL on retest.*

API-028 — Non-Sensitive GraphQL Schema Metadata Exposure	**LOW**
CVSS-Style Score: 2.4 (Low) — Illustrative CVSS v3.1-inspired score	**CWE: CWE-200 — Exposure of Sensitive Information to an Unauthorized Actor**
Affected API Type	GraphQL
Affected Operation / Endpoint	*GraphQL introspection (__schema.description fields)*
Description	Several fictional GraphQL type and field descriptions returned via introspection include internal engineering shorthand (e.g., references to internal service names) that, while not sensitive on their own, add marginally to platform fingerprinting.
Business Context	A low-impact observation distinct from API-012's administrative-surface exposure; this concerns descriptive text rather than the existence of privileged operations.
Technical Impact	Fictional introspection description fields referenced internal service names not otherwise documented publicly.
Business Impact	Marginal fingerprinting value only; no privileged operation or sensitive data disclosed.
Safe Proof of Concept (Summary)	*Fictional introspection description-field review found internal service-name references in schema documentation strings. [ILLUSTRATIVE / FICTIONAL SAMPLE EVIDENCE]*
Evidence Reference	*EV-API-033 (fictional schema description excerpt)*
Expected Behavior	Schema description strings intended for public/production consumption should avoid internal-only service naming.
Observed (Fictional) Behavior	Several description strings referenced internal service names.
Root Cause	Schema documentation strings were authored for internal developer convenience without a public-facing review pass.
Recommendation	Review and generalize public-facing schema description strings to remove internal service-naming references.
Management Response	*Accepted as a low-priority documentation hardening item.*
Owner / Priority / Target / Status	**API Platform Lead (Fictional Role)** | 61-90 Days | 13 December 2026 | **Open**
Retest Recommendation	*Confirm generalized description strings on retest.*

API-029 — Residual Debug Logging in Client Bundle	**LOW**
CVSS-Style Score: 2.8 (Low) — Illustrative CVSS v3.1-inspired score	**CWE: CWE-532 — Insertion of Sensitive Information into Log File**
Affected API Type	REST
Affected Operation / Endpoint	*Front-end application bundle (browser console output)*
Description	The fictional production front-end bundle retains verbose `console.log` debug statements, including non-sensitive internal state objects, that were disabled in most but not all modules.
Business Context	Debug logging left enabled in production increases the front-end's information footprint and can occasionally evolve to include sensitive data if not consistently reviewed.
Technical Impact	Fictional browser console output during testing included internal component-state logging on a subset of pages; no credentials or tokens were observed in this fictional sample.
Business Impact	Minor information-footprint increase; no sensitive data disclosed in this fictional sample.
Safe Proof of Concept (Summary)	*Fictional browser console capture showed residual debug logging on a subset of application pages. [ILLUSTRATIVE / FICTIONAL SAMPLE EVIDENCE]*
Evidence Reference	*EV-API-034 (fictional console-output sample)*
Expected Behavior	Debug logging stripped from production builds via the CI/CD build pipeline.
Observed (Fictional) Behavior	Debug logging present on a subset of pages, inconsistently stripped.
Root Cause	Build-time log-stripping configuration is not uniformly applied across all front-end modules.

Recommendation	Extend the existing build-time log-stripping step to cover all modules; add a CI check to fail the build on residual debug logging.
Management Response	*Accepted as a hardening item for the front-end build pipeline.*
Owner / Priority / Target / Status	**Front-End Engineering Lead (Fictional Role)** | 61-90 Days | 13 December 2026 | **Open**
Retest Recommendation	*Confirm log-stripping coverage across all modules on retest.*

API-030 — API Documentation / Enforcement Mismatch	**LOW**
CVSS-Style Score: 2.7 (Low) — Illustrative CVSS v3.1-inspired score	**CWE: N/A — Documentation / Process Observation**
Affected API Type	REST
Affected Operation / Endpoint	*Published OpenAPI documentation vs. enforced behavior*
Description	The published fictional OpenAPI documentation describes a maximum file-upload size that is two megabytes smaller than the limit actually enforced server-side.
Business Context	A minor documentation mismatch rather than a technical control gap; the enforced control itself was validated as adequate during file-handling testing.
Technical Impact	No security control weakness — this is a documentation-accuracy inconsistency identified during file-handling testing.
Business Impact	Minor developer-experience friction for API consumers relying on the published limit.
Safe Proof of Concept (Summary)	*Fictional side-by-side comparison of documented versus enforced file-upload size limits. [ILLUSTRATIVE / FICTIONAL SAMPLE EVIDENCE]*
Evidence Reference	*EV-API-035 (fictional documentation/enforcement comparison)*
Expected Behavior	Published documentation should match the enforced server-side limit exactly.
Observed (Fictional) Behavior	Documentation understated the enforced limit by two megabytes.
Root Cause	Documentation was not updated when the server-side limit was last adjusted.
Recommendation	Update the published OpenAPI documentation to match the current enforced file-upload limit.
Management Response	*Accepted; a documentation-only fix.*
Owner / Priority / Target / Status	**API Platform Lead (Fictional Role)** | 61-90 Days | 13 December 2026 | **Open**
Retest Recommendation	*Confirm documentation/enforcement alignment on retest.*

API-031 — Stale Endpoint Metadata in API Registry	**LOW**
CVSS-Style Score: 2.3 (Low) — Illustrative CVSS v3.1-inspired score	**CWE: N/A — Documentation / Process Observation**
Affected API Type	REST
Affected Operation / Endpoint	*Internal API registry / catalog entries*
Description	Several entries in the fictional internal API registry reference an owning team that no longer maintains the corresponding endpoint, complicating incident response and change-ownership tracking.
Business Context	Accurate API-registry ownership metadata supports faster, safer incident response and change management.
Technical Impact	No technical control gap — a metadata/process observation identified while cross-referencing the endpoint register (Appendix B) against internal ownership records.
Business Impact	Slower incident-response routing if ownership metadata is relied upon during a real event.
Safe Proof of Concept (Summary)	*Fictional registry cross-reference identified stale ownership metadata for a subset of endpoints. [ILLUSTRATIVE / FICTIONAL SAMPLE EVIDENCE]*
Evidence Reference	*EV-API-036 (fictional registry excerpt vs. current ownership)*
Expected Behavior	API registry ownership metadata should be reviewed and updated as part of every team-reorganization or ownership-transfer process.
Observed (Fictional) Behavior	Ownership metadata for a subset of endpoints did not reflect current team assignments.
Root Cause	Registry updates are not yet a required step in the team-reorganization process.
Recommendation	Add API-registry ownership review as a mandatory checklist item for team-reorganization and ownership-transfer

	processes.
Management Response	*Accepted; registry update scheduled.*
Owner / Priority / Target / Status	**Platform Security Lead (Fictional Role)** | 61-90 Days | 13 December 2026 | **Open**
Retest Recommendation	*Not required for closure; confirm on next general assessment.*

API-032 — Incomplete Session Artifact Cleanup on Logout	**LOW**
CVSS-Style Score: 2.5 (Low) — Illustrative CVSS v3.1-inspired score	**CWE: CWE-613 — Insufficient Session Expiration**
Affected API Type	REST
Affected Operation / Endpoint	*Front-end logout handler*
Description	Logout clears the primary authentication tokens but leaves one non-privileged UI-preference artifact (a cached "last viewed organization" value) in browser local storage.
Business Context	The retained artifact does not grant access on its own, but represents an inconsistency in the logout "clean state" behavior worth correcting for defense-in-depth.
Technical Impact	Fictional local-storage inspection after logout showed the non-privileged preference value still present; no authentication material remained.
Business Impact	Minor defense-in-depth inconsistency; no authentication material exposed.
Safe Proof of Concept (Summary)	*Fictional local-storage inspection after logout confirmed only a non-privileged UI-preference value remained. [ILLUSTRATIVE / FICTIONAL SAMPLE EVIDENCE]*
Evidence Reference	*EV-API-037 (fictional local-storage state capture, before/after logout)*
Expected Behavior	Logout should clear all application-related local-storage entries, not only authentication tokens.
Observed (Fictional) Behavior	One non-privileged preference value persisted after logout.
Root Cause	The logout handler's storage-clearing routine targets an explicit key list that does not include the newer preference key.
Recommendation	Update the logout handler to clear the full application storage namespace rather than an explicit key list.
Management Response	*Accepted as a minor hardening item.*
Owner / Priority / Target / Status	**Front-End Engineering Lead (Fictional Role)** | 61-90 Days | 13 December 2026 | **Open**
Retest Recommendation	*Confirm full storage-namespace clearing on retest.*

49 INFORMATIONAL FINDINGS

The following 10 fictional Informational observations carry no direct exploitation path and are recommended for the ongoing API-governance and defense-in-depth backlog.

API-033 — Public GraphQL Documentation Portal (Intended Exposure — Review Recommended)	INFORMATIONAL
CVSS-Style Score: N/A — Informational observation, no CVSS score assigned	**CWE: N/A**
Affected API Type	GraphQL
Affected Operation / Endpoint	*/graphql/docs*
Description	An interactive GraphQL documentation portal is publicly reachable without authentication; this appears to be an intended developer-experience feature but is noted for periodic review to confirm no sensitive internal-only operations are described there.
Business Context	Public API documentation is a common and often intentional practice; the observation is limited to recommending periodic content review.
Technical Impact	N/A — reviewed documentation content did not disclose any fictional internal-only operation in this sample.
Business Impact	None identified in this fictional sample; recommendation is preventive.
Safe Proof of Concept (Summary)	*Fictional documentation portal reachable without authentication; content reviewed and found consistent with the public API surface. [ILLUSTRATIVE / FICTIONAL SAMPLE EVIDENCE]*
Evidence Reference	*EV-API-038 (fictional documentation content review notes)*
Expected Behavior	N/A — recommendation, not a control failure.
Observed (Fictional) Behavior	Publicly reachable, content consistent with intended public API surface.
Root Cause	N/A.
Recommendation	Maintain a periodic review process to ensure internal-only operations are never included in the public documentation set.
Management Response	*Accepted; will be folded into the existing documentation review cadence.*
Owner / Priority / Target / Status	**API Platform Lead (Fictional Role)** | Backlog — Defense-in-Depth | Backlog — Next Planning Cycle | **Open**
Retest Recommendation	*Not required for closure; confirm on next general assessment.*

API-034 — Introspection Intentionally Available in Non-Production Environment	INFORMATIONAL
CVSS-Style Score: N/A — Informational observation, no CVSS score assigned	**CWE: N/A**
Affected API Type	GraphQL
Affected Operation / Endpoint	*GraphQL introspection (staging environment)*
Description	GraphQL introspection is fully enabled in the fictional staging environment used for this assessment, which is expected and appropriate for a pre-production environment used by developers and QA.
Business Context	This observation exists to explicitly distinguish the staging-environment introspection posture (appropriate, informational) from the production-relevant finding in API-012 (meaningful, Medium).
Technical Impact	N/A — expected behavior for the assessed environment tier.
Business Impact	None; this is the expected and appropriate configuration for the assessed environment.
Safe Proof of Concept (Summary)	*Fictional confirmation that introspection is intentionally enabled in the staging environment used for this assessment. [ILLUSTRATIVE / FICTIONAL SAMPLE EVIDENCE]*
Evidence Reference	*EV-API-039 (fictional environment-configuration confirmation)*
Expected Behavior	N/A — recommendation, not a control failure.
Observed (Fictional) Behavior	Introspection enabled, consistent with staging-environment expectations.

Root Cause	N/A.
Recommendation	No action required for the staging tier; ensure the production-tier gap tracked in API-012 remains the operative remediation target.
Management Response	*Acknowledged; tracked jointly with API-012.*
Owner / Priority / Target / Status	**API Platform Lead (Fictional Role)** | Backlog — Defense-in-Depth | Backlog — Next Planning Cycle | **Open**
Retest Recommendation	*Not required for closure.*

API-035 — Non-Sensitive Schema Metadata Present in Build Artifacts	**INFORMATIONAL**
CVSS-Style Score: N/A — Informational observation, no CVSS score assigned	**CWE: CWE-200 — Exposure of Sensitive Information to an Unauthorized Actor**
Affected API Type	GraphQL
Affected Operation / Endpoint	*Front-end build metadata / generated schema types*
Description	The fictional front-end bundle includes a generated GraphQL schema-types file with a non-sensitive build-identifier comment (schema hash and generation date).
Business Context	Low-impact observation; noted for completeness alongside the fingerprinting-related items above.
Technical Impact	N/A — observation only.
Business Impact	Marginal fingerprinting value only.
Safe Proof of Concept (Summary)	*Fictional bundle inspection located a non-sensitive schema-build-identifier comment. [ILLUSTRATIVE / FICTIONAL SAMPLE EVIDENCE]*
Evidence Reference	*EV-API-040 (fictional bundle excerpt)*
Expected Behavior	N/A — recommendation, not a control failure.
Observed (Fictional) Behavior	Non-sensitive build metadata present in generated schema-types file comments.
Root Cause	Default code-generation tool behavior; not a discrete control gap.
Recommendation	Optional: strip build-identifier comments from production bundles as a minor hardening step.
Management Response	*Acknowledged; low priority.*
Owner / Priority / Target / Status	**Front-End Engineering Lead (Fictional Role)** | Backlog — Defense-in-Depth | Backlog — Next Planning Cycle | **Open**
Retest Recommendation	*Not required for closure.*

API-036 — Security.txt Not Present	**INFORMATIONAL**
CVSS-Style Score: N/A — Informational observation, no CVSS score assigned	**CWE: N/A**
Affected API Type	REST
Affected Operation / Endpoint	*/.well-known/security.txt*
Description	The fictional application does not publish a security.txt file describing a coordinated vulnerability-disclosure contact and process.
Business Context	A published security.txt is a low-cost best practice that streamlines external researcher reporting.
Technical Impact	N/A — observation only.
Business Impact	None identified; recommendation is preventive.
Safe Proof of Concept (Summary)	*Fictional request to /.well-known/security.txt returned 404. [ILLUSTRATIVE / FICTIONAL SAMPLE EVIDENCE]*
Evidence Reference	*EV-API-041 (fictional request/response)*
Expected Behavior	N/A — recommendation, not a control failure.
Observed (Fictional) Behavior	File not present in the fictional test environment.
Root Cause	Not yet implemented.
Recommendation	Publish a security.txt file per the informal community convention, pointing to a dedicated fictional OrionSphere vulnerability-disclosure contact (e.g., security@orion-example.com).
Management Response	*Accepted as a low-effort improvement for a future release.*

Owner / Priority / Target / Status	**Platform Security Lead (Fictional Role)** | Backlog — Defense-in-Depth | Backlog — Next Planning Cycle | **Open**
Retest Recommendation	*Not required for closure; confirm on next general assessment.*

API-037 — API Inventory / Asset-Management Improvement Opportunity	**INFORMATIONAL**
CVSS-Style Score: N/A — Informational observation, no CVSS score assigned	**CWE: N/A**
Affected API Type	REST & GraphQL
Affected Operation / Endpoint	*N/A — process observation*
Description	The fictional organization maintains its REST endpoint and GraphQL operation inventories in separate, independently-updated spreadsheets rather than a single authoritative API catalog.
Business Context	A unified inventory reduces the risk of shadow or forgotten endpoints — directly relevant to the shadow-API testing performed in this assessment.
Technical Impact	N/A — process observation.
Business Impact	Increased risk of future shadow/undocumented API drift if not addressed.
Safe Proof of Concept (Summary)	*Fictional review of API inventory management practices identified fragmented tracking across REST and GraphQL. [ILLUSTRATIVE / FICTIONAL SAMPLE EVIDENCE]*
Evidence Reference	*EV-API-042 (fictional inventory-process review notes)*
Expected Behavior	N/A — recommendation, not a control failure.
Observed (Fictional) Behavior	Fragmented inventory tracking across REST and GraphQL.
Root Cause	No single authoritative API catalog tool currently in use.
Recommendation	Adopt a unified API catalog tool covering both REST and GraphQL, integrated with the CI/CD pipeline so new operations are automatically registered.
Management Response	*Accepted; will be evaluated as part of the broader API-governance roadmap.*
Owner / Priority / Target / Status	**API Platform Lead (Fictional Role)** | Backlog — Defense-in-Depth | Backlog — Next Planning Cycle | **Open**
Retest Recommendation	*Not required for closure.*

API-038 — Logging Coverage Recommendation for Authorization Events	**INFORMATIONAL**
CVSS-Style Score: N/A — Informational observation, no CVSS score assigned	**CWE: N/A**
Affected API Type	REST & GraphQL
Affected Operation / Endpoint	*N/A — process observation*
Description	Centralized logging captures authentication events (a positive control, Section 56), but does not yet include a dedicated log entry for GraphQL resolver-level authorization denials distinct from REST authorization denials.
Business Context	Monitoring enhancements complement the underlying control fixes (API-002, API-008, API-011) and support faster detection of similar authorization-bypass patterns in the future.
Technical Impact	N/A — observation only.
Business Impact	Slower detection of future authorization-bypass attempts at the GraphQL resolver layer specifically.
Safe Proof of Concept (Summary)	*Fictional review of logging configuration found GraphQL resolver-level authorization denials are not distinctly logged from REST denials. [ILLUSTRATIVE / FICTIONAL SAMPLE EVIDENCE]*
Evidence Reference	*EV-API-043 (fictional logging-configuration review)*
Expected Behavior	N/A — recommendation, not a control failure.
Observed (Fictional) Behavior	Authorization denials are logged, but not distinctly tagged by transport (REST vs. GraphQL) or resolver.
Root Cause	Logging schema has not yet been updated to reflect this assessment's GraphQL-specific findings.
Recommendation	Add resolver-level tagging to GraphQL authorization-denial log entries, informed by API-002, API-008, and API-011.

Management Response	*Accepted; will be added alongside the related GraphQL authorization remediations.*
Owner / Priority / Target / Status	**Security Operations Lead (Fictional Role)** | Backlog — Defense-in-Depth | Backlog — Next Planning Cycle | **Open**
Retest Recommendation	*Not required for closure; confirm alongside API-002/API-008/API-011 retest.*

API-039 — Monitoring Enhancement Opportunity for GraphQL Resolver Errors	**INFORMATIONAL**
CVSS-Style Score: N/A — Informational observation, no CVSS score assigned	**CWE: N/A**
Affected API Type	GraphQL
Affected Operation / Endpoint	*N/A — process observation*
Description	GraphQL resolver errors are captured in general application logs but are not yet surfaced on a dedicated monitoring dashboard distinct from REST error-rate monitoring.
Business Context	A dedicated GraphQL error-rate view would help operations staff distinguish resolver-specific issues (including the verbose-error pattern in API-021) from broader platform incidents.
Technical Impact	N/A — observation only.
Business Impact	Slower operational triage of GraphQL-specific incidents relative to REST incidents.
Safe Proof of Concept (Summary)	*Fictional review of the monitoring dashboard confirmed no GraphQL-specific error-rate view exists separate from overall API error monitoring. [ILLUSTRATIVE / FICTIONAL SAMPLE EVIDENCE]*
Evidence Reference	*EV-API-044 (fictional monitoring-dashboard review)*
Expected Behavior	N/A — recommendation, not a control failure.
Observed (Fictional) Behavior	GraphQL resolver errors are aggregated with general API error monitoring rather than surfaced separately.
Root Cause	Monitoring dashboards were built before GraphQL adoption expanded and have not been updated.
Recommendation	Add a dedicated GraphQL resolver error-rate panel to the operations monitoring dashboard.
Management Response	*Accepted; dashboard update scheduled.*
Owner / Priority / Target / Status	**Platform Security Lead (Fictional Role)** | Backlog — Defense-in-Depth | Backlog — Next Planning Cycle | **Open**
Retest Recommendation	*Not required for closure.*

API-040 — API Lifecycle Governance Recommendation	**INFORMATIONAL**
CVSS-Style Score: N/A — Informational observation, no CVSS score assigned	**CWE: N/A**
Affected API Type	REST & GraphQL
Affected Operation / Endpoint	*N/A — design observation*
Description	There is currently no formal governance gate requiring a security-authorization review before a new REST endpoint or GraphQL operation ships to production.
Business Context	This is a forward-looking process recommendation rather than a specific control failure, intended to catch authorization gaps (like several found in this assessment) before release rather than in a subsequent pentest.
Technical Impact	N/A — process/design observation.
Business Impact	Ongoing risk of new authorization gaps shipping undetected between assessment cycles.
Safe Proof of Concept (Summary)	*Cross-referenced design observation drawn from the authorization-related findings throughout this assessment (API-001, API-002, API-008, API-011). [ILLUSTRATIVE / FICTIONAL SAMPLE EVIDENCE]*
Evidence Reference	*Cross-reference: API-001, API-002, API-008, API-011*
Expected Behavior	N/A — recommendation, not a control failure.
Observed (Fictional) Behavior	No formal pre-release authorization review gate currently exists.
Root Cause	Architectural/process gap rather than a specific implementation defect.
Recommendation	Introduce a lightweight security-authorization review checklist as a required step before any new REST endpoint or GraphQL operation ships to production.
Management Response	*Accepted as a longer-term governance consideration for the fictional product roadmap.*

Owner / Priority / Target / Status	**VP Engineering (Fictional Role)** | Backlog — Defense-in-Depth | Backlog — Next Planning Cycle | **Open**
Retest Recommendation	*Not required for closure; revisit as part of the next full assessment cycle.*

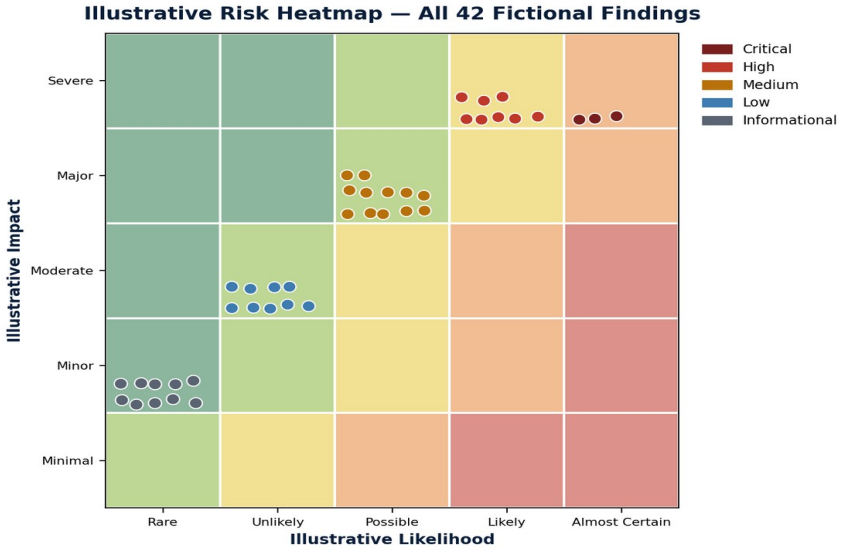
API-041 — Deprecated Endpoint Cleanup Backlog	**INFORMATIONAL**
CVSS-Style Score: N/A — Informational observation, no CVSS score assigned	**CWE: N/A**
Affected API Type	REST
Affected Operation / Endpoint	*Multiple /api/v1/* routes*
Description	Beyond the specific authorization regression tracked in API-009, several additional legacy v1 routes are documented as deprecated but have no tracked retirement date in the fictional project backlog.
Business Context	Complements API-009 and API-022; this item is the broader backlog-hygiene recommendation rather than a specific technical gap.
Technical Impact	N/A — process observation.
Business Impact	Growing legacy-endpoint surface area if not proactively retired.
Safe Proof of Concept (Summary)	*Fictional backlog review found deprecated v1 routes without a tracked retirement date beyond the one covered in API-009. [ILLUSTRATIVE / FICTIONAL SAMPLE EVIDENCE]*
Evidence Reference	*EV-API-045 (fictional backlog excerpt)*
Expected Behavior	N/A — recommendation, not a control failure.
Observed (Fictional) Behavior	Deprecated routes lack a tracked retirement date in the project backlog.
Root Cause	Deprecation was recorded in documentation but not carried through to a tracked backlog item.
Recommendation	Create a tracked backlog item with a target retirement date for every documented-deprecated route.
Management Response	*Accepted; backlog items will be created for all remaining deprecated routes.*
Owner / Priority / Target / Status	**API Platform Lead (Fictional Role)** | Backlog — Defense-in-Depth | Backlog — Next Planning Cycle | **Open**
Retest Recommendation	*Not required for closure; confirm alongside API-009 and API-022 retest.*

API-042 — Recommend Automated Authorization Regression Testing	**INFORMATIONAL**
CVSS-Style Score: N/A — Informational observation, no CVSS score assigned	**CWE: N/A**
Affected API Type	REST & GraphQL
Affected Operation / Endpoint	*N/A — process observation*
Description	Authorization testing across roles is currently performed manually during periodic assessments rather than as an automated regression suite run in CI.
Business Context	An automated authorization regression suite, built from this assessment's role/operation matrix (Appendix D), would catch a meaningful share of this report's findings automatically on every future code change.
Technical Impact	N/A — process/design observation.
Business Impact	Authorization regressions currently surface only at the next periodic assessment rather than at commit time.
Safe Proof of Concept (Summary)	*Cross-referenced recommendation drawn from the authorization findings identified throughout this assessment. [ILLUSTRATIVE / FICTIONAL SAMPLE EVIDENCE]*
Evidence Reference	*Cross-reference: Sections 15-19, Appendix D*
Expected Behavior	N/A — recommendation, not a control failure.
Observed (Fictional) Behavior	No automated authorization regression suite currently exists.
Root Cause	Authorization testing has historically been treated as a manual, periodic activity.
Recommendation	Build an automated authorization regression suite, seeded from the role/operation matrix in Appendix D, and run it in CI on every pull request touching an authorization-relevant code path.
Management Response	*Accepted as a longer-term testing-maturity investment.*

Owner / Priority / Target / Status	**VP Engineering (Fictional Role)** | Backlog — Defense-in-Depth | Backlog — Next Planning Cycle | **Open**
Retest Recommendation	*Not required for closure; revisit as part of the next full assessment cycle.*

50 RISK HEATMAP

The heatmap below plots all 42 fictional findings by illustrative likelihood and impact, reflecting the severity assigned to each finding in Sections 45-49. This view is intended to support prioritization discussions alongside the Management Action Plan (Section 54).

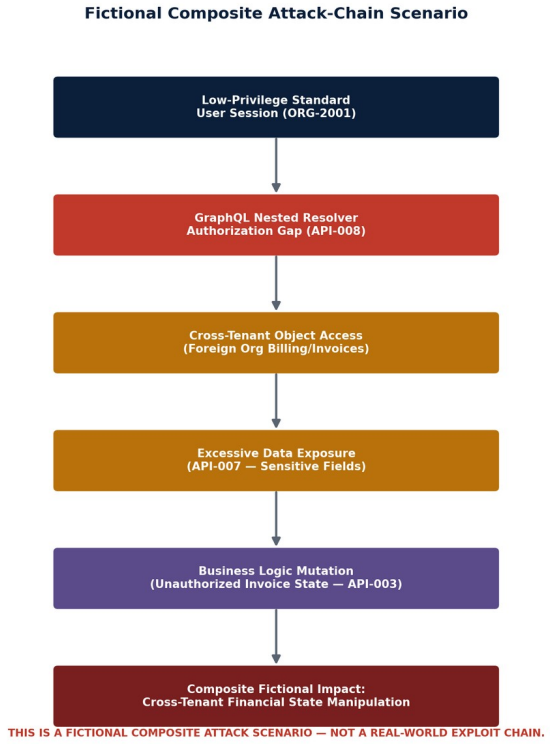


Illustrative risk heatmap — all 42 fictional findings plotted by likelihood and impact.

This heatmap reflects fictional, illustrative severity data only and does not represent the risk posture of any real organization.

51 ATTACK CHAIN ANALYSIS

Individual findings are often most meaningful in combination. This section presents one fictional composite attack-chain scenario, illustrating how several independently Medium-to-Critical findings from this assessment — including a GraphQL-specific trust-boundary gap — could combine into a more significant illustrative impact if left unaddressed together.



Fictional composite attack-chain scenario — not a real-world exploit chain.

Fictional Composite Scenario Narrative

Starting from a low-privilege, authenticated Standard User session in ORG-2001, the GraphQL nested-resolver authorization gap (API-008) permits traversal into another member's billing and invoice data through a nested query, even though the root organization query is correctly scoped. The excessive GraphQL data-exposure finding (API-007) means that object — and equivalent objects reachable the same way — returns more internal detail than intended, including risk-scoring and billing-reference fields. Independently, the business-logic authorization bypass in invoice/refund state handling (API-003) shows that workflow-state enforcement has gaps elsewhere in the platform, reachable through both REST and GraphQL. Combined, an attacker following this fictional path could reach cross-tenant financial data exposure together with unauthorized financial-state manipulation — a composite illustrative impact more significant than any single finding in isolation, and a concrete example of why resolver-level authorization testing (Section 31) is treated as a priority discipline in this methodology.

THIS IS A FICTIONAL COMPOSITE ATTACK SCENARIO. It does not describe, and must not be construed as, an exploit chain against any real system.

52 BUSINESS IMPACT ANALYSIS

The table below maps each Critical and High fictional finding to the illustrative business dimensions it most directly affects, supporting prioritization conversations beyond pure technical severity.

Finding ID	Confidentiality	Integrity	Fraud / Revenue Risk	Customer Trust	Privacy
API-001	High	Low	Medium	High	High
API-002	Low	High	High	Medium	Low
API-003	Medium	High	High	High	Medium
API-004	Medium	Medium	Low	Medium	Low
API-005	Medium	Low	Low	Medium	Medium
API-006	Medium	Low	Low	Medium	Medium
API-007	High	Low	Low	Medium	High
API-008	High	Medium	Medium	High	High

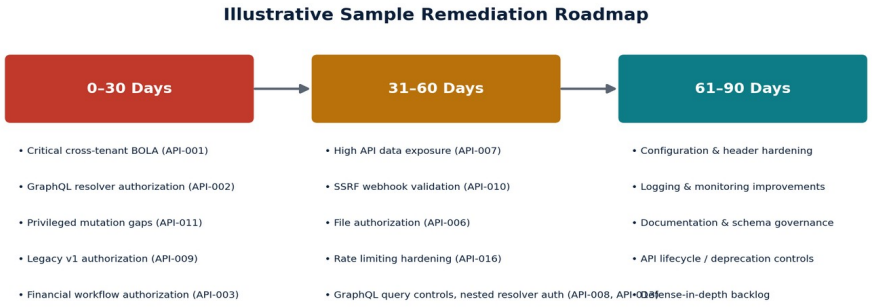
Finding ID	Confidentiality	Integrity	Fraud / Revenue Risk	Customer Trust	Privacy
API-009	Medium	Low	Low	Low	Low
API-010	Medium	Medium	Low	Low	Low
API-011	Low	High	Medium	Medium	Low

Illustrative Regulatory & Operational Risk Note

As a fictional enterprise SaaS / FinTech infrastructure platform, OrionSphere's illustrative regulatory risk considerations would include data-protection and financial-services-adjacent expectations around access control, tenant data segregation, and financial-transaction integrity. This report does not assess compliance with any specific regulatory framework; the fictional findings above are offered only as illustrative business-impact context.

53 REMEDIATION ROADMAP

The illustrative sample remediation roadmap below sequences all 42 fictional findings across three phases, prioritizing Critical authorization and GraphQL resolver findings first.



ILLUSTRATIVE SAMPLE REMEDIATION ROADMAP — fictional sequencing and dates.

0-30 Days

- Critical cross-tenant BOLA — API-001
- GraphQL privileged mutation authorization bypass — API-002
- Business logic / financial workflow authorization bypass — API-003
- Legacy v1 authorization weakness — API-009
- Privilege escalation through role mutation — API-011

31-60 Days

- High API data exposure — API-007
- GraphQL nested resolver authorization bypass — API-008
- SSRF through integration webhook API — API-010
- REST file authorization bypass — API-006
- Stored XSS through API-supplied support content — API-004
- Account recovery API weakness — API-005
- Rate limiting hardening & GraphQL query controls — API-016, API-013, API-014, API-015

61-90 Days

- Configuration & header hardening — API-020, API-024, API-025, API-026
- Logging & monitoring improvements — API-038, API-039
- Documentation & schema governance — API-030, API-033, API-035, API-037
- API lifecycle / deprecation controls — API-022, API-031, API-040, API-041
- Remaining defense-in-depth backlog — API-017, API-018, API-019, API-021, API-023, API-027, API-028, API-029, API-032, API-036, API-042

ILLUSTRATIVE SAMPLE REMEDIATION ROADMAP — sequencing, owners, and dates are fictional.

54 MANAGEMENT ACTION PLAN

The table below tracks the fictional management response, ownership, and validation approach for all 42 findings, ensuring every finding identified in Sections 45-49 appears here exactly once.

Finding ID	Owner	Priority	Target Date	Status	Validation Method	Remediation (Summary)
API-001	VP Engineering	Immediate — 0-30 Days	15 October 2026	Remediation In Progress	Full or Targeted Retest	Implement a single, centrally-enforced tenant-scoping check…
API-002	Director, API Platform	Immediate — 0-30 Days	15 October 2026	Remediation In Progress	Full or Targeted Retest	Apply the shared role-authorization guard used by REST bill…
API-003	VP Finance Engineering	Immediate — 0-30 Days	15 October 2026	Remediation In Progress	Full or Targeted Retest	Introduce an explicit, server-side invoice state machine sh…
API-004	Engineering Lead, Support…	0-30 Days	29 October 2026	Open	Full or Targeted Retest	Apply a single shared, context-aware output-encoding compon…
API-005	Identity & Access Lead	0-30 Days	29 October 2026	Open	Full or Targeted Retest	Bind each confirmation code to its originating recovery-req…
API-006	Platform Security Lead	31-60 Days	28 November 2026	Open	Full or Targeted Retest	Add a mandatory ownership check (file-to-organization) befo…
API-007	API Platform Lead	31-60 Days	28 November 2026	Open	Full or Targeted Retest	Introduce role-aware field resolvers (or a field-authorizat…
API-008	Director, API Platform	0-30 Days	29 October 2026	Remediation In Progress	Full or Targeted Retest	Adopt a per-resolver (or per-type) authorization pattern —…
API-009	API Platform Lead	31-60 Days	28 November 2026	Remediation Planned	Full or Targeted Retest	Either retire the v1 account-retrieval route on a defined s…
API-010	Platform Security Lead	0-30 Days	29 October 2026	Remediation Planned	Full or Targeted Retest	Implement an egress allow-list and explicit blocklist for p…
API-011	Identity & Access Lead	0-30 Days	29 October 2026	Remediation In Progress	Full or Targeted Retest	Add explicit requester-role authorization to the role-updat…
API-012	API Platform Lead	31-60 Days	28 November 2026	Open	Full or Targeted Retest	Adopt schema segmentation (or disable introspection in prod…
API-013	API Platform Lead	31-60 Days	28 November 2026	Remediation Planned	Full or Targeted Retest	Introduce a query-complexity scoring middleware (cost-per-f…
API-014	API Platform Lead	31-60 Days	28 November 2026	Remediation Planned	Full or Targeted Retest	Add an explicit maximum-depth validator (recommended: depth…
API-015	API Platform Lead	31-60 Days	28 November 2026	Open	Full or Targeted Retest	Extend rate-limiting accounting to the GraphQL layer, count…
API-016	Identity & Access Lead	31-60 Days	28 November 2026	Remediation Planned	Full or Targeted Retest	Implement per-account and per-IP rate limiting with exponen…
API-017	Platform Security Lead	31-60 Days	28 November 2026	Open	Full or Targeted Retest	Introduce a lightweight server-side revocation check (short…
API-018	API Platform Lead	31-60 Days	28 November 2026	Open	Full or Targeted Retest	Replace generic body-binding with an explicit allow-listed…
API-019	Integrations Platform Lead	31-60 Days	28 November 2026	Remediation Planned	Full or Targeted Retest	Add a timestamp and nonce (or monotonic event ID) to the si…
API-020	Platform Security Lead	31-60 Days	28 November 2026	Remediation Planned	Full or Targeted Retest	Reduce the CORS allow-list to the minimal explicit set of a…
API-021	Platform Engineering Lead	31-60 Days	28 November 2026	Open	Full or Targeted Retest	Standardize a centralized GraphQL error-formatting layer th…
API-022	API Platform Lead	31-60 Days	28 November 2026	Open	Full or Targeted Retest	Add `Deprecation`/`Sunset` header emission for all document…
API-023	Director, Billing Platform	31-60 Days	28 November 2026	Open	Full or Targeted Retest	Enforce atomic, database-level uniqueness/locking on redemp…
API-024	Platform Engineering Lead	61-90 Days	13 December 2026	Open	Full or Targeted Retest	Disable or genericize version-disclosing headers at the fra…
API-025	Platform Engineering Lead	61-90 Days	13 December 2026	Remediation Planned	Full or Targeted Retest	Add Permissions-Policy and Referrer-Policy to the centraliz…
API-026	Platform Security Lead	61-90 Days	13 December 2026	Open	Full or Targeted Retest	Configure an explicit HTTP method allow-list at the CDN/WAF…
API-027	Platform Engineering Lead	61-90 Days	13 December 2026	Open	Full or Targeted Retest	Centralize field-level validation rules in a single shared…
API-028	API Platform Lead	61-90 Days	13 December 2026	Open	Full or Targeted Retest	Review and generalize public-facing schema description stri…
API-029	Front-End Engineering Lead	61-90 Days	13 December 2026	Open	Full or Targeted Retest	Extend the existing build-time log-stripping step to cover…
API-030	API Platform Lead	61-90 Days	13 December 2026	Open	Full or Targeted Retest	Update the published OpenAPI documentation to match the cur…

Finding ID	Owner	Priority	Target Date	Status	Validation Method	Remediation (Summary)
API-031	Platform Security Lead	61-90 Days	13 December 2026	Open	Documentation / Config Review	Add API-registry ownership review as a mandatory checklist...
API-032	Front-End Engineering Lead	61-90 Days	13 December 2026	Open	Full or Targeted Retest	Update the logout handler to clear the full application sto...
API-033	API Platform Lead	Backlog — Defense-in-Depth	Backlog — Next Planning Cycle	Open	Documentation / Config Review	Maintain a periodic review process to ensure internal-only...
API-034	API Platform Lead	Backlog — Defense-in-Depth	Backlog — Next Planning Cycle	Open	Documentation / Config Review	No action required for the staging tier; ensure the product...
API-035	Front-End Engineering Lead	Backlog — Defense-in-Depth	Backlog — Next Planning Cycle	Open	Documentation / Config Review	Optional: strip build-identifier comments from production b...
API-036	Platform Security Lead	Backlog — Defense-in-Depth	Backlog — Next Planning Cycle	Open	Documentation / Config Review	Publish a security.txt file per the informal community conv...
API-037	API Platform Lead	Backlog — Defense-in-Depth	Backlog — Next Planning Cycle	Open	Documentation / Config Review	Adopt a unified API catalog tool covering both REST and Gra...
API-038	Security Operations Lead	Backlog — Defense-in-Depth	Backlog — Next Planning Cycle	Open	Documentation / Config Review	Add resolver-level tagging to GraphQL authorization-denial...
API-039	Platform Security Lead	Backlog — Defense-in-Depth	Backlog — Next Planning Cycle	Open	Documentation / Config Review	Add a dedicated GraphQL resolver error-rate panel to the op...
API-040	VP Engineering	Backlog — Defense-in-Depth	Backlog — Next Planning Cycle	Open	Documentation / Config Review	Introduce a lightweight security-authorization review check...
API-041	API Platform Lead	Backlog — Defense-in-Depth	Backlog — Next Planning Cycle	Open	Documentation / Config Review	Create a tracked backlog item with a target retirement date...
API-042	VP Engineering	Backlog — Defense-in-Depth	Backlog — Next Planning Cycle	Open	Documentation / Config Review	Build an automated authorization regression suite, seeded f...

55 RETEST RECOMMENDATIONS

TMG Security recommends a structured retest following remediation, applying the methodology below. This section describes the recommended retest approach for this fictional engagement; it does not state that remediation has actually occurred.

Recommended Retest Scope

- Full retest of all three Critical findings (API-001 through API-003) and all eight High findings (API-004 through API-011), including role-based regression testing across related REST endpoints and GraphQL operations.
- Targeted retest of each Medium finding against its specific affected endpoint, GraphQL operation, or workflow.
- Confirmation-only review of Low and Informational items, batched into the next general assessment cycle where full retest is not independently warranted.

Retest Methodology

- Evidence Review — confirm that a code or configuration change addressing the finding's root cause has been deployed to the fictional test environment.
- Original PoC Reproduction — attempt the original safe PoC steps (REST and/or GraphQL, as applicable) to confirm the previously observed fictional behavior no longer occurs.
- Role-Based Regression — re-verify authorization behavior across all seven fictional roles for the affected endpoint or operation, not only the role used in the original PoC.
- REST Regression — re-run the affected REST endpoint's test cases from Appendix A to confirm no adjacent regression.
- GraphQL Regression — re-run the affected GraphQL operation's test cases, including nested-field and resolver-level checks where the original finding involved resolver authorization.
- Resolver Authorization Validation — for GraphQL resolver-authorization findings (API-002, API-008, API-011), independently verify that authorization is now enforced at every nested resolver level, not only at the root query/mutation.
- Schema Re-Review — re-run introspection and schema-metadata review to confirm no newly introduced privileged operation is unintentionally exposed.

- Query-Complexity Validation — re-test the complexity/depth/batching findings (API-013 through API-015) against the same synthetic queries used originally.
- Business-Logic Regression — re-run the affected multi-step workflow end-to-end to confirm state-integrity enforcement, not only the single-request PoC.
- Closure Criteria — a finding is marked Closed only after reproduction confirms remediation and regression testing raises no new concern.

This retest methodology is illustrative. This report does not claim that any finding has been remediated or retested.

56 POSITIVE SECURITY CONTROLS

This fictional assessment is not intended to be entirely negative. TMG Security identified a number of security controls operating effectively across the OrionSphere Enterprise Platform's REST and GraphQL surfaces, summarized below as illustrative fictional observations.

Control Area	Illustrative Observation
Multi-Factor Authentication	MFA is available and functional for privileged roles, with single-use recovery codes and enforcement at login for enrolled accounts.
Refresh-Token Rotation	Refresh tokens rotate on use and invalidate prior tokens, limiting the value of a captured fictional refresh token.
TLS / HTTPS Enforcement	TLS configuration disallows deprecated protocol versions and weak cipher suites across both REST and GraphQL endpoints tested.
API Gateway	A centralized fictional API gateway fronts both the REST and GraphQL layers, providing a consistent point for future authorization-policy enforcement.
Web Application Firewall	A cloud-based fictional WAF sits in front of both API interfaces, providing a baseline layer of edge protection.
Request Validation (Partial)	Type-level GraphQL input validation and REST schema validation correctly reject malformed structural input (see Sections 21-22).
Centralized Logging	Authentication and role-change events are captured in centralized fictional logging, supporting the monitoring enhancement noted in API-038 and API-039.
GraphQL Authorization (Partial)	Root-level GraphQL query and mutation authorization is correctly enforced in the majority of operations tested — the gaps identified are concentrated in nested-resolver and nested-field authorization (see Section 31).
Rate Limiting (Partial)	Rate limiting is effectively implemented on the primary login endpoint, demonstrating the pattern needed elsewhere (see API-016).
Object-Level Authorization (Partial)	Root-level object-level authorization is correctly enforced on the majority of REST endpoints and GraphQL queries tested — the confirmed gaps are concentrated in the specific findings detailed in Sections 16 and 19.

All observations above are fictional and were constructed for this illustrative sample assessment.

57 TESTING LIMITATIONS

The following limitations apply to this fictional sample assessment and should be read together with the Important Disclaimer (Section 02).

- OrionSphere Technologies, Inc. and the OrionSphere Enterprise Platform are fictional; no real organization, REST API, or GraphQL API was assessed.
- No real credentials, API keys, cloud credentials, or production systems were used or accessed.
- No real customer data was accessed at any point.

- No production environment was tested; only a fictional staging/controlled assessment environment is represented.
- No destructive testing, including destructive race-condition automation, was performed or simulated as executed.
- No denial-of-service testing, including GraphQL resource-exhaustion exploitation, was performed.
- No bulk-data-extraction technique was constructed or demonstrated, including from the batching/alias finding (API-015).
- No social engineering was performed.
- No physical security testing was performed.
- No real cloud-provider environment, infrastructure, or cloud metadata service was tested or accessed.
- No third-party systems were tested.
- All evidence, screenshots, GraphQL queries, and request/response examples in this report are synthetic.

This report describes a fictional, illustrative testing exercise only.

58 APPENDIX A — API TEST CASE REGISTER

The register below documents all 140 fictional test cases executed during this illustrative assessment, spanning 28 testing categories across REST and GraphQL, with a result and cross-reference to any corresponding finding. Every failed test ("Exception Identified") maps to a finding documented in Sections 45-49.

Category	Objective	Result	Finding
Authentication	Verify /api/v1/auth/login rejects invalid credentials with a generic, non-enumerable error.	No Exception Identified	—
Authentication	Verify authentication throttling applies after repeated failed login attempts.	Exception Identified	API-016
Authentication	Verify JWT access tokens carry a bounded, documented expiration window.	No Exception Identified	—
Authentication	Verify refresh-token rotation invalidates the prior refresh token on use.	No Exception Identified	—
Authentication	Verify bearer tokens are rejected once the associated session has been explicitly logged out.	Exception Identified	API-017
Authentication	Verify API-key authentication on /api/v1/integration/* endpoints is independently scoped from session auth.	No Exception Identified	—
Authorization	Verify Standard User cannot access another fictional tenant's organization profile via REST.	Exception Identified	API-001
Authorization	Verify Standard User cannot access another fictional tenant's organization profile via GraphQL.	Exception Identified	API-001
Authorization	Verify server-side authorization is enforced independent of front-end role-based UI hiding.	Exception Identified	API-002
Authorization	Verify Administrator-only REST and GraphQL operations reject non-administrator sessions.	No Exception Identified	—
Authorization	Verify Billing User cannot escalate to Manager-level organization functions.	No Exception Identified	—
Authorization	Verify Support Agent session cannot invoke billing/subscription mutations.	No Exception Identified	—
BOLA	Verify GET /api/v1/organizations/{id} enforces tenant ownership independent of client-supplied ID.	Exception Identified	API-001
BOLA	Verify PATCH /api/v1/invoices/{id} enforces tenant ownership before permitting a status change.	Exception Identified	API-003
BOLA	Verify GET /api/v1/files/{id} enforces file-owner / organization scoping.	Exception Identified	API-006
BOLA	Verify GET /api/v1/transactions/{id} rejects cross-account object references.	No Exception Identified	—
BOLA	Verify GraphQL query invoice(id) enforces the same tenant-ownership check as its REST equivalent.	Exception Identified	API-003
BOLA	Verify account-scoped nested collections (transactions, invoices) cannot be enumerated across accounts.	No Exception Identified	—
BFLA	Verify Standard User session cannot invoke GraphQL mutation updateOrganizationBilling.	Exception Identified	API-002
BFLA	Verify Billing User session cannot access REST admin endpoints (/api/v1/admin/*).	No Exception Identified	—
BFLA	Verify Support Agent session cannot invoke GraphQL mutation updateMemberRole.	Exception Identified	API-011
BFLA	Verify API Integration User session cannot invoke human-user-only mutations (e.g. acceptInvitation).	No Exception Identified	—
BFLA	Verify hidden/unlinked administrative mutations remain authorization-gated at the resolver layer.	No Exception Identified	—
Field-Level Authorization	Verify GraphQL query user(id) omits internalRiskScore for non-privileged roles.	Exception Identified	API-007
Field-Level Authorization	Verify GraphQL query user(id) omits billingAccountReference for non-privileged roles.	Exception Identified	API-007
Field-Level Authorization	Verify GraphQL query user(id) omits adminNotes for non-privileged roles.	Exception Identified	API-007
Field-Level Authorization	Verify REST GET /api/v1/organizations/{id}/members omits internal role metadata and risk scoring for Standard User.	Exception Identified	API-007
Field-Level Authorization	Verify field-level authorization is enforced consistently between REST and GraphQL representations of the same object.	No Exception Identified	—
Tenant Isolation	Verify organization ID scoping is derived from session context, not request parameters.	Exception Identified	API-001
Tenant Isolation	Verify GraphQL global object IDs cannot be substituted across organizations to access nested billing data.	Exception Identified	API-008
Tenant Isolation	Verify file IDs cannot be used to access another tenant's fictional file objects.	Exception Identified	API-006
Tenant Isolation	Verify GraphQL fragments and aliases cannot be used to bypass tenant-scoping checks.	No Exception Identified	—

Category	Objective	Result	Finding
Tenant Isolation	Verify GraphQL mutations (updateInvoice, updateOrganizationBilling) enforce tenant ownership at the resolver layer.	Exception Identified	API-002
Tenant Isolation	Verify subscription event streams (organizationEvents) do not leak cross-tenant events.	Exception Identified	API-002
REST	Verify REST endpoint enumeration via /api/v1/docs matches the enforced, tested endpoint surface.	Exception Identified	API-030
REST	Verify HTTP methods not required by a resource are rejected (TRACE, unused PUT/DELETE).	Exception Identified	API-026
REST	Verify REST error responses use a consistent, non-verbose error envelope.	No Exception Identified	—
REST	Verify REST API versioning enforces equivalent authorization across /api/v1/ and /api/v2/ routes.	Exception Identified	API-009
REST	Verify hidden/undocumented REST parameters do not alter authorization or business-logic outcomes.	No Exception Identified	—
REST	Verify predictable sequential object identifiers do not materially aid unauthorized object discovery.	No Exception Identified	—
GraphQL	Verify the GraphQL endpoint enforces authentication consistently with its REST equivalents.	No Exception Identified	—
GraphQL	Verify GraphQL error responses do not leak resolver names or internal service metadata.	Exception Identified	API-021
GraphQL	Verify GraphQL mutations validate input types and reject malformed payloads safely.	No Exception Identified	—
GraphQL	Verify GraphQL query cost is bounded independent of client-supplied query shape.	Exception Identified	API-013
GraphQL	Verify deprecated GraphQL fields remain access-controlled equivalently to their replacements.	Exception Identified	API-022
GraphQL	Verify GraphQL subscriptions require authentication and tenant-scoped authorization.	No Exception Identified	—
Introspection	Verify GraphQL introspection (__schema) exposure aligns with the documented environment posture.	Exception Identified	API-012
Introspection	Verify introspection does not expose privileged administrative mutations without independent authorization controls.	Exception Identified	API-012
Introspection	Verify __type queries do not expose non-sensitive-but-undocumented internal schema metadata.	Exception Identified	API-028
Introspection	Verify introspection-derived schema does not itself grant execution privileges beyond normal authorization.	No Exception Identified	—
Introspection	Verify staging-environment introspection availability is intentional and documented.	Exception Identified	API-034
Resolver Security	Verify Organization.members.billing nested resolver enforces the same authorization as its root query.	Exception Identified	API-008
Resolver Security	Verify resolver-level authorization checks execute independent of parent-object authorization outcome.	No Exception Identified	—
Resolver Security	Verify resolver error paths do not silently default to an authorized state on missing context.	No Exception Identified	—
Resolver Security	Verify resolvers enforce authorization prior to data-layer access, not only at the API-gateway layer.	No Exception Identified	—
Resolver Security	Verify custom scalar/field resolvers apply consistent role-based filtering across query paths.	No Exception Identified	—
Mutations	Verify updateOrganizationBilling enforces Manager/Administrator-only authorization.	Exception Identified	API-002
Mutations	Verify updateInvoice enforces tenant and role authorization prior to status transition.	Exception Identified	API-003
Mutations	Verify updateMemberRole enforces Manager-only authorization and audit logging.	Exception Identified	API-011
Mutations	Verify updateUser rejects client-supplied role/status/billingTier/adminFlag fields from non-privileged roles.	Exception Identified	API-018
Mutations	Verify createWebhook validates destination URLs against internal/private address ranges.	Exception Identified	API-010
Mutations	Verify redeemDiscount enforces single-use semantics under rapid-sequence requests.	Exception Identified	API-023
Subscriptions	Verify organizationEvents subscription enforces tenant-scoped event filtering.	Exception Identified	API-002
Subscriptions	Verify notificationEvents subscription cannot be subscribed to on behalf of another user.	No Exception Identified	—
Subscriptions	Verify webhookDeliveryEvents subscription requires Manager-level authorization.	No Exception Identified	—
Subscriptions	Verify subscription lifecycle (connect/disconnect) is authorization-checked on each event delivery, not only at connect time.	No Exception Identified	—
Nested Queries	Verify nested organization → members → billing → invoices traversal enforces authorization at every level.	Exception Identified	API-008

Category	Objective	Result	Finding
Nested Queries	Verify recursive members.members relationship traversal is authorization-checked at each recursion depth.	Exception Identified	API-014
Nested Queries	Verify deeply nested queries (depth 8) are evaluated against a documented safe complexity threshold.	Exception Identified	API-013
Nested Queries	Verify nested field resolution does not bypass parent-object tenant scoping.	No Exception Identified	—
Nested Queries	Verify nested query results are consistent with equivalent flat REST queries for the same object.	No Exception Identified	—
Aliases	Verify aliased multi-object queries (3x user lookups) are still individually authorization-checked.	Exception Identified	API-015
Aliases	Verify alias-based query shaping does not circumvent per-field rate limiting.	No Exception Identified	—
Aliases	Verify fragment-based alias reuse does not alter authorization outcome for sensitive fields.	No Exception Identified	—
Batching	Verify batched/aliased requests are rate-limited equivalently to sequential single requests.	Exception Identified	API-015
Batching	Verify batch query cost accounting reflects the sum of individual resolver costs.	No Exception Identified	—
Batching	Verify batching cannot be used to enumerate objects beyond normal pagination limits.	No Exception Identified	—
Query Depth	Verify GraphQL query depth is capped at a documented, enforced maximum.	Exception Identified	API-014
Query Depth	Verify depth-limiting is applied server-side and not solely advisory in client tooling.	No Exception Identified	—
Query Depth	Verify depth-limit violations return a clear, non-verbose error rather than partial execution.	No Exception Identified	—
Query Complexity	Verify GraphQL query complexity scoring rejects requests above the documented safe threshold.	Exception Identified	API-013
Query Complexity	Verify complexity scoring accounts for nested list multipliers, not only field count.	No Exception Identified	—
Query Complexity	Verify complexity-threshold violations are logged for monitoring and trend analysis.	Exception Identified	API-039
Injection	Verify SQL-injection-style fictional payloads in REST filter parameters are safely parameterized.	No Exception Identified	—
Injection	Verify NoSQL-injection-style fictional payloads in GraphQL filter arguments do not alter query logic.	No Exception Identified	—
Injection	Verify path-traversal-style fictional payloads in file-reference parameters are rejected.	No Exception Identified	—
Injection	Verify stored HTML/script content submitted via API is encoded on output in all rendering contexts.	Exception Identified	API-004
Injection	Verify server-side template-injection-style fictional payloads in text fields are not evaluated.	No Exception Identified	—
Data Exposure	Verify REST GET /api/v1/organizations/{id}/members applies role-aware field scoping.	Exception Identified	API-007
Data Exposure	Verify GraphQL field resolvers apply role-aware filtering rather than broad object serialization.	Exception Identified	API-007
Data Exposure	Verify API responses do not include internal identifiers beyond what the client role requires.	No Exception Identified	—
Data Exposure	Verify financial data fields (balances, risk scores) are scoped to authorized roles across REST and GraphQL.	No Exception Identified	—
Data Exposure	Verify audit-event data exposure is limited to Administrator and Manager roles.	No Exception Identified	—
Mass Assignment	Verify PATCH /api/v1/users/{id} rejects client-supplied role/status/billingTier/adminFlag fields.	Exception Identified	API-018
Mass Assignment	Verify GraphQL mutation updateUser rejects the same hidden-field set as its REST equivalent.	Exception Identified	API-018
Mass Assignment	Verify updateAccount mutation does not bind client-supplied riskTier without explicit authorization.	Exception Identified	API-018
Mass Assignment	Verify mass-assignment protections are enforced server-side via allow-listing, not client-side form field hiding.	No Exception Identified	—
Rate Limiting	Verify /api/v1/auth/login enforces rate limiting after repeated failed attempts.	Exception Identified	API-016
Rate Limiting	Verify /api/v1/auth/password-reset/request enforces application-level rate limiting.	Exception Identified	API-016
Rate Limiting	Verify /api/v1/auth/mfa enforces rate limiting on OTP submission attempts.	Exception Identified	API-016
Rate Limiting	Verify the GraphQL endpoint applies request-level rate limiting independent of REST rate limits.	No Exception Identified	—
Rate Limiting	Verify expensive mutations (generateReport, exportTransactions) are throttled per account.	No Exception Identified	—
Rate Limiting	Verify file-upload and report-generation operations enforce reasonable per-user request caps.	No Exception Identified	—

Category	Objective	Result	Finding
Webhooks	Verify webhook destination validation rejects internal/private network targets.	Exception Identified	API-010
Webhooks	Verify webhook payloads include a verifiable signature validated by receivers.	No Exception Identified	—
Webhooks	Verify webhook delivery includes a timestamp and nonce sufficient to detect replay.	Exception Identified	API-019
Webhooks	Verify webhook secret values are never returned in plaintext after initial issuance.	No Exception Identified	—
Webhooks	Verify webhook event IDs are unique and idempotency-checked on the receiving side.	No Exception Identified	—
File APIs	Verify file-download signed URLs enforce organization-ownership checks.	Exception Identified	API-006
File APIs	Verify uploaded file type validation relies on server-side content inspection, not client-supplied metadata.	No Exception Identified	—
File APIs	Verify uploaded filenames are sanitized to prevent path traversal on storage.	No Exception Identified	—
File APIs	Verify file-share links cannot be generated for files outside the requesting user's authorization scope.	Exception Identified	API-006
File APIs	Verify file metadata endpoints do not expose internal storage paths or bucket identifiers.	No Exception Identified	—
CORS	Verify CORS Access-Control-Allow-Origin reflects only the explicit, active front-end origin set.	Exception Identified	API-020
CORS	Verify CORS preflight responses do not grant credentials to untrusted origins.	Exception Identified	API-020
CORS	Verify GraphQL endpoint CORS configuration matches REST endpoint CORS policy.	Exception Identified	API-020
Error Handling	Verify REST error responses do not include stack traces or internal service names.	No Exception Identified	—
Error Handling	Verify GraphQL error extensions do not include internal resolver names or database details.	Exception Identified	API-021
Error Handling	Verify malformed-input error responses return a generic client-facing message.	Exception Identified	API-021
Error Handling	Verify debug metadata is not present in production-representative error payloads.	No Exception Identified	—
Versioning	Verify legacy /api/v1/accounts/{id} enforces authorization equivalent to /api/v2/accounts/{id}.	Exception Identified	API-009
Versioning	Verify deprecated endpoints marked in documentation are either removed or equivalently hardened.	Exception Identified	API-022
Versioning	Verify version-negotiation headers do not allow silent downgrade to a weaker-authorization route.	No Exception Identified	—
Versioning	Verify deprecated GraphQL fields carry accurate @deprecated annotations matching actual enforcement.	Exception Identified	API-022
Shadow APIs	Verify legacy/deprecated REST routes discovered during enumeration are inventoried and access-controlled.	Exception Identified	API-041
Shadow APIs	Verify internal API registry entries reflect the actual deployed and tested endpoint surface.	Exception Identified	API-031
Shadow APIs	Verify undocumented endpoints discovered during testing are not exempt from standard authorization review.	No Exception Identified	—
Shadow APIs	Verify stale endpoint metadata does not misrepresent current authentication requirements.	Exception Identified	API-031
Documentation	Verify published REST OpenAPI documentation accurately reflects enforced authentication requirements.	Exception Identified	API-030
Documentation	Verify GraphQL public documentation portal does not expose administrative operation descriptions.	Exception Identified	API-033
Documentation	Verify schema field descriptions do not contain internal implementation detail beyond client need.	Exception Identified	API-035
Documentation	Verify /.well-known/security.txt is present and references a current disclosure contact.	Exception Identified	API-036
Documentation	Verify API inventory documentation is updated as part of the standard release process.	Exception Identified	API-037
REST	Verify HTTP response headers apply recommended hardening (Permissions-Policy, Referrer-Policy).	Exception Identified	API-025
REST	Verify request-body field-length and format constraints are applied consistently across related endpoints.	Exception Identified	API-027
REST	Verify production-representative front-end bundles do not emit residual debug logging to the browser console.	Exception Identified	API-029
Authentication	Verify client-side session artifacts (tokens, cached state) are fully cleared following logout.	Exception Identified	API-032
Documentation	Verify authorization-relevant events (role changes, privileged mutations) have adequate logging coverage for monitoring.	Exception Identified	API-038
Documentation	Verify authorization test coverage (Sections 15-19, Appendix D) is a candidate for automated regression testing.	Exception Identified	API-042

Category	Objective	Result	Finding
Rate Limiting	Verify report-generation requests are capped per-account to prevent resource-exhaustion patterns.	**No Exception Identified**	—

59 APPENDIX B — REST ENDPOINT REGISTER

The register below documents all 126 fictional REST endpoints identified during attack-surface discovery (Section 11) and REST enumeration (Section 12), with method, authentication, role, function, and testing result.

Endpoint	Method	Auth	Role	Function	Tested	Result	Finding
/api/v1/auth/login	POST	None	Visitor	Credential-based login, session/token issuance	Yes	Finding	API-016
/api/v1/auth/refresh	POST	Refresh Token	Standard User	Access-token refresh	Yes	Clean	—
/api/v1/auth/logout	POST	Session	Standard User	Session/token termination	Yes	Finding	API-017
/api/v1/auth/mfa	POST	Session (partial)	Standard User	MFA challenge submission	Yes	Finding	API-016
/api/v1/auth/mfa/enroll	POST	Session	Standard User	MFA enrollment	Yes	Clean	—
/api/v1/auth/mfa/verify	POST	Session (partial)	Standard User	MFA code verification	Yes	Clean	—
/api/v1/auth/mfa/recovery	POST	None	Standard User	MFA account-recovery code exchange	Yes	Finding	API-005
/api/v1/auth/password-reset/request	POST	None	Visitor	Password-reset request	Yes	Finding	API-016
/api/v1/auth/password-reset/confirm	POST	None	Visitor	Password-reset confirmation	Yes	Clean	—
/api/v1/auth/api-keys	GET	Session	API Integration User	API key listing	Yes	Clean	—
/api/v1/users	GET	Session	Administrator	Platform-wide user listing	Yes	Clean	—
/api/v1/users	POST	Session	Administrator	User creation	Yes	Clean	—
/api/v1/users/{id}	GET	Session	Standard User	User object retrieval	Yes	Clean	—
/api/v1/users/{id}	PATCH	Session	Standard User	User profile / attribute update	Yes	Finding	API-018
/api/v1/users/{id}	DELETE	Session	Administrator	User account deletion	Yes	Clean	—
/api/v1/users/{id}/roles	GET	Session	Manager	User role assignment retrieval	Yes	Clean	—
/api/v1/users/{id}/sessions	GET	Session	Standard User	Active session listing	Yes	Clean	—
/api/v1/users/{id}/sessions/{sessionId}	DELETE	Session	Standard User	Session revocation	Yes	Finding	API-017
/api/v1/organizations	GET	Session	Standard User	Organization listing (scoped)	Yes	Clean	—
/api/v1/organizations	POST	Session	Administrator	Organization creation	Yes	Clean	—
/api/v1/organizations/{id}	GET	Session	Standard User	Organization profile retrieval	Yes	Finding	API-001
/api/v1/organizations/{id}	PATCH	Session	Manager	Organization profile update	Yes	Clean	—
/api/v1/organizations/{id}	DELETE	Session	Administrator	Organization deletion	Yes	Clean	—
/api/v1/organizations/{id}/members	GET	Session	Standard User	Member directory retrieval	Yes	Finding	API-007
/api/v1/organizations/{id}/members	POST	Session	Manager	Member invitation	Yes	Clean	—
/api/v1/organizations/{id}/members/{userId}	DELETE	Session	Manager	Member removal	Yes	Clean	—
/api/v1/organizations/{id}/roles	GET	Session	Manager	Organization role listing	Yes	Clean	—
/api/v1/organizations/{id}/roles/{userId}	PATCH	Session	Manager	Member role assignment	Yes	Finding	API-011
/api/v1/accounts	GET	Session	Administrator	Platform-wide account listing	Yes	Clean	—
/api/v1/accounts/{id}	GET	Session	Standard User	Account detail (legacy v1)	Yes	Finding	API-009
/api/v2/accounts/{id}	GET	Session	Standard User	Account detail (current v2)	Yes	Clean	—
/api/v1/accounts/{id}	PATCH	Session	Billing User	Account attribute update (legacy v1)	Yes	Finding	API-009
/api/v2/accounts/{id}	PATCH	Session	Billing User	Account attribute update (current v2)	Yes	Clean	—
/api/v1/accounts/{id}/ownership	POST	Session	Manager	Account ownership transfer	Yes	Finding	API-023
/api/v1/accounts/{id}/audit	GET	Session	Administrator	Account audit trail	Yes	Clean	—
/api/v1/accounts/{id}/close	POST	Session	Manager	Account closure workflow	Yes	Clean	—
/api/v1/transactions	GET	Session	Billing User	Transaction listing (scoped)	Yes	Clean	—
/api/v1/transactions/{id}	GET	Session	Billing User	Transaction detail retrieval	Yes	Finding	API-001
/api/v1/transactions/{id}/replay-token	GET	Session	Billing User	Idempotency / replay-token retrieval	Yes	Finding	API-022
/api/v1/transactions/export	GET	Session	Billing User	Bulk transaction export	Yes	Clean	—
/api/v1/transactions/{id}/dispute	POST	Session	Billing User	Transaction dispute filing	Yes	Clean	—
/api/v1/transactions/{id}/void	POST	Session	Billing User	Transaction void request	Yes	Clean	—
/api/v1/invoices	GET	Session	Billing User	Invoice listing (scoped)	Yes	Clean	—
/api/v1/invoices/{id}	GET	Session	Billing User	Invoice detail retrieval	Yes	Clean	—
/api/v1/invoices/{id}	PATCH	Session	Billing User	Invoice status update	Yes	Finding	API-003
/api/v1/invoices/{id}/refund	POST	Session	Billing User	Invoice refund state update	Yes	Finding	API-003
/api/v1/invoices/{id}/pdf	GET	Session	Billing User	Invoice PDF export	Yes	Clean	—
/api/v1/invoices/{id}/comments	POST	Session	Billing User	Invoice comment submission	Yes	Clean	—
/api/v1/invoices/{id}/dispute	POST	Session	Billing User	Invoice dispute filing	Yes	Clean	—
/api/v1/invoices/overdue	GET	Session	Billing User	Overdue invoice listing	Yes	Clean	—
/api/v1/subscriptions	GET	Session	Billing User	Subscription listing	Yes	Clean	—
/api/v1/subscriptions/{id}	GET	Session	Billing User	Subscription detail retrieval	Yes	Clean	—
/api/v1/subscriptions/{id}	PATCH	Session	Billing User	Subscription plan / tier update	Yes	Finding	API-018
/api/v1/subscriptions/{id}/cancel	POST	Session	Billing User	Subscription cancellation	Yes	Clean	—
/api/v1/subscriptions/{id}/upgrade	POST	Session	Billing User	Subscription upgrade	Yes	Finding	API-023
/api/v1/subscriptions/{id}/discount/redeem	POST	Session	Billing User	Discount-code redemption	Yes	Finding	API-023
/api/v1/subscriptions/{id}/proration	GET	Session	Billing User	Subscription proration preview	Yes	Clean	—
/api/v1/subscriptions/plans	GET	None	Visitor	Public plan catalog	Yes	Clean	—
/api/v1/files	GET	Session	Standard User	File listing (scoped)	Yes	Clean	—
/api/v1/files	POST	Session	Standard User	File upload	Yes	Clean	—
/api/v1/files/{id}	GET	Session	Standard User	File metadata / signed-URL retrieval	Yes	Finding	API-006
/api/v1/files/{id}	DELETE	Session	Standard User	File deletion	Yes	Clean	—
/api/v1/files/{id}/download	GET	Session	Standard User	File download (signed URL)	Yes	Finding	API-006

Endpoint	Method	Auth	Role	Function	Tested	Result	Finding
/api/v1/files/{id}/share	POST	Session	Standard User	File share-link generation	Yes	Clean	—
/api/v1/files/{id}/versions	GET	Session	Standard User	File version history	Yes	Clean	—
/api/v1/notifications	GET	Session	Standard User	Notification listing	Yes	Clean	—
/api/v1/notifications/{id}/read	PATCH	Session	Standard User	Notification read-state update	Yes	Clean	—
/api/v1/notifications/{id}	DELETE	Session	Standard User	Notification deletion	Yes	Clean	—
/api/v1/notifications/preferences	GET	Session	Standard User	Notification preference retrieval	Yes	Clean	—
/api/v1/notifications/preferences	PATCH	Session	Standard User	Notification preference update	Yes	Clean	—
/api/v1/webhooks	GET	Session	Manager	Webhook subscription listing	Yes	Clean	—
/api/v1/webhooks	POST	Session	Manager	Webhook subscription creation	Yes	Finding	API-011
/api/v1/webhooks/{id}	GET	Session	Manager	Webhook subscription detail	Yes	Clean	—
/api/v1/webhooks/{id}	PATCH	Session	Manager	Webhook subscription update	Yes	Clean	—
/api/v1/webhooks/{id}	DELETE	Session	Manager	Webhook subscription deletion	Yes	Clean	—
/api/v1/webhooks/validate	POST	Session	Manager	Webhook destination validation (server-side fetch)	Yes	Finding	API-010
/api/v1/webhooks/{id}/deliveries	GET	Session	Manager	Webhook delivery log retrieval	Yes	Finding	API-019
/api/v1/reports	GET	Session	Standard User	Report listing	Yes	Clean	—
/api/v1/reports	POST	Session	Standard User	Report generation request	Yes	Clean	—
/api/v1/reports/{id}	GET	Session	Standard User	Report detail retrieval	Yes	Clean	—
/api/v1/reports/{id}/comments	POST	Session	Support Agent	Report comment submission	Yes	Finding	API-004
/api/v1/reports/{id}/export	GET	Session	Standard User	Report export (CSV/PDF)	Yes	Clean	—
/api/v1/reports/scheduled	GET	Session	Manager	Scheduled report configuration listing	Yes	Clean	—
/api/v1/admin/users	GET	Session	Administrator	Platform-wide user administration	Yes	Clean	—
/api/v1/admin/organizations	GET	Session	Administrator	Platform-wide organization administration	Yes	Clean	—
/api/v1/admin/audit-events	GET	Session	Administrator	Platform audit-event retrieval	Yes	Clean	—
/api/v1/admin/feature-flags	GET	Session	Administrator	Feature-flag configuration	Yes	Clean	—
/api/v1/admin/system-config	GET	Session	Administrator	System configuration retrieval	Yes	Clean	—
/api/v1/admin/impersonate	POST	Session	Administrator	Administrative user impersonation	Yes	Clean	—
/api/v1/integration/api-keys	POST	Session	API Integration User	API key issuance	Yes	Clean	—
/api/v1/integration/api-keys/{id}	DELETE	Session	API Integration User	API key revocation	Yes	Clean	—
/api/v1/integration/scopes	GET	Session	API Integration User	API key scope listing	Yes	Clean	—
/api/v1/integration/events	GET	Session	API Integration User	Integration event log retrieval	Yes	Clean	—
/api/v1/integration/ping	GET	API Key	API Integration User	Integration connectivity check	Yes	Clean	—
/api/v1/docs	GET	None	Visitor	Interactive REST API documentation (Swagger/OpenAPI)	Yes	Finding	API-030
/api/v1/openapi.json	GET	None	Visitor	OpenAPI schema document	Yes	Finding	API-030
/.well-known/security.txt	GET	None	Visitor	Vulnerability-disclosure contact metadata	Yes	Finding	API-036
/api/v1/health	GET	None	Visitor	Service health/liveness check	Yes	Clean	—
/api/v1/status	GET	None	Visitor	Service status page	Yes	Clean	—
/api/v1/version	GET	None	Visitor	API version/build metadata	Yes	Finding	API-024
/api/v1/registry	GET	Session	Administrator	Internal API catalog / registry entry listing	Yes	Finding	API-031
/api/v1/organizations/{id}/profile	GET	Session	Manager	Organization profile (legacy alias)	Yes	Finding	API-022
/api/v1/billing/summary	GET	Session	Billing User	Billing summary (deprecated route)	Yes	Finding	API-022
/api/v1/files/{id}/legacy-preview	GET	Session	Standard User	File preview (deprecated route)	Yes	Finding	API-041
/api/v1/users/{id}/legacy-profile	GET	Session	Standard User	User profile (deprecated route)	Yes	Finding	API-041
/api/v1/subscriptions/{id}/legacy-status	GET	Session	Billing User	Subscription status (deprecated route)	Yes	Finding	API-022
/api/v1/reports/{id}/legacy-export	GET	Session	Standard User	Report export (deprecated route)	Yes	Finding	API-041
/api/v1/users/{id}	OPTIONS	None	Visitor	CORS preflight — user resource	Yes	Finding	API-020
/api/v1/organizations/{id}	OPTIONS	None	Visitor	CORS preflight — organization resource	Yes	Finding	API-020
/api/v1/invoices/{id}	TRACE	Session	Billing User	Unused HTTP method probe — invoice resource	Yes	Finding	API-026
/api/v1/accounts/{id}	PUT	Session	Billing User	Unused HTTP method probe — account resource	Yes	Finding	API-026
/api/v1/files/{id}	HEAD	Session	Standard User	Metadata-only probe — file resource	Yes	Clean	—
/api/v1/webhooks/{id}	OPTIONS	None	Visitor	CORS preflight — webhook resource	Yes	Finding	API-020
/api/v1/organizations/{id}/invoices	GET	Session	Billing User	Organization-scoped invoice listing	Yes	Clean	—
/api/v1/organizations/{id}/audit-events	GET	Session	Manager	Organization audit-event retrieval	Yes	Clean	—
/api/v1/organizations/{id}/webhooks	GET	Session	Manager	Organization-scoped webhook listing	Yes	Clean	—
/api/v1/organizations/{id}/subscriptions	GET	Session	Billing User	Organization-scoped subscription listing	Yes	Clean	—
/api/v1/accounts/{id}/transactions	GET	Session	Billing User	Account-scoped transaction listing	Yes	Clean	—
/api/v1/accounts/{id}/invoices	GET	Session	Billing User	Account-scoped invoice listing	Yes	Clean	—
/api/v1/users/{id}/notifications	GET	Session	Standard User	User-scoped notification listing	Yes	Clean	—
/api/v1/users/{id}/files	GET	Session	Standard User	User-scoped file listing	Yes	Clean	—
/api/v1/integration/webhooks	GET	API Key	API Integration User	Integration-user webhook listing	Yes	Clean	—
/api/v1/integration/reports	POST	API Key	API Integration User	Integration-triggered report generation	Yes	Finding	API-040
/api/v1/support/tickets	GET	Session	Support Agent	Support-ticket listing	Yes	Clean	—
/api/v1/support/tickets/{id}	GET	Session	Support Agent	Support-ticket detail retrieval	Yes	Clean	—
/api/v1/notifications/broadcast	POST	Session	Administrator	Platform-wide notification broadcast	Yes	Clean	—

60 APPENDIX C — GRAPHQL OPERATION REGISTER

The register below documents all 78 fictional GraphQL operations identified during schema discovery (Section 13), with type, authentication requirement, required role, resolver, fields, and testing result.

Operation	Type	Auth?	Role	Resolver	Fields	Tested	Result	Finding
me	Query	Yes	Standard User	Query.me	id, name, email, role, organizationId	Yes	Clean	—
user(id)	Query	Yes	Standard User	Query.user	id, name, email, role, internalRiskScore, billingAccountReference, adminNotes	Yes	Finding	API-007
users(filter)	Query	Yes	Administrator	Query.users	id, name, email, role, status	Yes	Clean	—
organization(id)	Query	Yes	Standard User	Query.organization	id, name, plan, members, billingTier	Yes	Finding	API-001
organizations(filter)	Query	Yes	Administrator	Query.organizations	id, name, plan, memberCount	Yes	Clean	—
member(id)	Query	Yes	Standard User	Query.member	id, userId, role, joinedAt	Yes	Clean	—
members(orgId)	Query	Yes	Standard User	Organization.members	id, userId, role, permissions	Yes	Clean	—
account(id)	Query	Yes	Standard User	Query.account	id, ownerId, status, balance	Yes	Clean	—
accounts(filter)	Query	Yes	Administrator	Query.accounts	id, ownerId, status	Yes	Clean	—
transaction(id)	Query	Yes	Billing User	Query.transaction	id, amount, status, accountId	Yes	Clean	—
transactions(filter)	Query	Yes	Billing User	Query.transactions	id, amount, status	Yes	Clean	—
invoice(id)	Query	Yes	Billing User	Query.invoice	id, amount, status, organizationId	Yes	Finding	API-008
invoices(filter)	Query	Yes	Billing User	Query.invoices	id, amount, status	Yes	Clean	—
subscription(id)	Query	Yes	Billing User	Query.subscription	id, planId, status, renewalDate	Yes	Clean	—
subscriptions(filter)	Query	Yes	Billing User	Query.subscriptions	id, planId, status	Yes	Clean	—
files(filter)	Query	Yes	Standard User	Query.files	id, name, ownerId, url	Yes	Clean	—
file(id)	Query	Yes	Standard User	Query.file	id, name, ownerId, url	Yes	Clean	—
notifications(filter)	Query	Yes	Standard User	Query.notifications	id, type, read, createdAt	Yes	Clean	—
auditEvents(filter)	Query	Yes	Administrator	Query.auditEvents	id, actorId, action, targetId, timestamp	Yes	Clean	—
reports(filter)	Query	Yes	Standard User	Query.reports	id, title, status, createdAt	Yes	Clean	—
organization.billing	Query (nested)	Yes	Manager	Organization.billing	invoices, paymentMethods, billingTier	Yes	Finding	API-008
organization.members.billing	Query (nested)	Yes	Standard User	Member.billing	invoices, balance, riskTier	Yes	Finding	API-008
account.transactions	Query (nested)	Yes	Billing User	Account.transactions	id, amount, status	Yes	Clean	—
user.adminNotes	Query (nested field)	Yes	Standard User	User.adminNotes	adminNotes	Yes	Finding	API-007
user.internalRiskScore	Query (nested field)	Yes	Standard User	User.internalRiskScore	internalRiskScore	Yes	Finding	API-007
__schema	Query (introspection)	No	Visitor	Introspection.__schema	types, queryType, mutationType, subscriptionType	Yes	Finding	API-012
__type(name)	Query (introspection)	No	Visitor	Introspection.__type	name, fields, description	Yes	Finding	API-028
reports.export	Query (nested)	Yes	Standard User	Report.exportUrl	exportUrl, generatedAt	Yes	Clean	—
organization.auditEvents	Query (nested)	Yes	Manager	Organization.auditEvents	id, action, actorId	Yes	Clean	—
account.owner	Query (nested)	Yes	Standard User	Account.owner	id, name, email	Yes	Clean	—
invoice.organization	Query (nested)	Yes	Billing User	Invoice.organization	id, name, billingTier	Yes	Finding	API-003
member.permissions	Query (nested)	Yes	Manager	Member.permissions	scopes, grantedBy	Yes	Clean	—
webhook(id)	Query	Yes	Manager	Query.webhook	id, url, events, secretMasked	Yes	Clean	—
webhooks(orgId)	Query	Yes	Manager	Query.webhooks	id, url, events	Yes	Clean	—
reports(orgId, aliasedBatch)	Query (aliased batch)	Yes	Standard User	Query.reports (x3 aliases)	id, title, createdBy	Yes	Finding	API-015
updateUser(id, input)	Mutation	Yes	Standard User	Mutation.updateUser	name, email, role, status, billingTier, adminFlag	Yes	Finding	API-018
updateOrganization(id, input)	Mutation	Yes	Manager	Mutation.updateOrganization	name, plan, billingTier	Yes	Clean	—
updateMemberRole(orgId, userId, role)	Mutation	Yes	Manager	Mutation.updateMemberRole	id, role, updatedBy	Yes	Finding	API-011
createInvoice(input)	Mutation	Yes	Billing User	Mutation.createInvoice	id, amount, status, organizationId	Yes	Clean	—
updateInvoice(id, input)	Mutation	Yes	Billing User	Mutation.updateInvoice	id, status, amount, organizationId	Yes	Finding	API-003
cancelSubscription(id)	Mutation	Yes	Billing User	Mutation.cancelSubscription	id, status, canceledAt	Yes	Clean	—
changeSubscriptionPlan(id, planId)	Mutation	Yes	Billing User	Mutation.changeSubscriptionPlan	id, planId, status	Yes	Clean	—
createWebhook(input)	Mutation	Yes	Manager	Mutation.createWebhook	id, url, events, secret	Yes	Clean	—
updateWebhook(id, input)	Mutation	Yes	Manager	Mutation.updateWebhook	id, url, events	Yes	Clean	—
deleteWebhook(id)	Mutation	Yes	Manager	Mutation.deleteWebhook	id, deleted	Yes	Clean	—
uploadFile(input)	Mutation	Yes	Standard User	Mutation.uploadFile	id, name, url	Yes	Clean	—
generateReport(input)	Mutation	Yes	Standard User	Mutation.generateReport	id, status, requestedAt	Yes	Clean	—
markNotificationRead(id)	Mutation	Yes	Standard User	Mutation.markNotificationRead	id, read	Yes	Clean	—
updateOrganizationBilling(id, input)	Mutation	Yes	Manager	Mutation.updateOrganizationBilling	id, billingTier, paymentMethod	Yes	Finding	API-002
createAccount(input)	Mutation	Yes	Administrator	Mutation.createAccount	id, ownerId, status	Yes	Clean	—
updateAccount(id, input)	Mutation	Yes	Billing User	Mutation.updateAccount	id, status, balance	Yes	Finding	API-018
transferAccountOwnership(id, newOwnerId)	Mutation	Yes	Manager	Mutation.transferAccountOwnership	id, ownerId	Yes	Clean	—
redeemDiscount(subscriptionId, code)	Mutation	Yes	Billing User	Mutation.redeemDiscount	id, discountApplied, redeemedAt	Yes	Finding	API-023
refundInvoice(id, input)	Mutation	Yes	Billing User	Mutation.refundInvoice	id, status, refundedAmount	Yes	Finding	API-003

Operation	Type	Auth?	Role	Resolver	Fields	Tested	Result	Finding
createReportComment(reportId, body)	Mutation	Yes	Support Agent	Mutation.createReportComment	id, body, authorId	Yes	Finding	API-004
deleteFile(id)	Mutation	Yes	Standard User	Mutation.deleteFile	id, deleted	Yes	Clean	—
shareFile(id, targetUserId)	Mutation	Yes	Standard User	Mutation.shareFile	id, sharedWith	Yes	Finding	API-006
inviteMember(orgId, email, role)	Mutation	Yes	Manager	Mutation.inviteMember	id, email, role, status	Yes	Clean	—
acceptInvitation(token)	Mutation	No	Visitor	Mutation.acceptInvitation	id, organizationId, role	Yes	Clean	—
approveWorkflowStep(id)	Mutation	Yes	Manager	Mutation.approveWorkflowStep	id, status, approvedBy	Yes	Finding	API-023
revokeApiKey(id)	Mutation	Yes	API Integration User	Mutation.revokeApiKey	id, revoked	Yes	Clean	—
issueApiKey(scopes)	Mutation	Yes	API Integration User	Mutation.issueApiKey	id, key, scopes	Yes	Clean	—
updateNotificationPreferences(input)	Mutation	Yes	Standard User	Mutation.updateNotificationPreferences	channels, frequency	Yes	Clean	—
transactionEvents(accountId)	Subscription	Yes	Billing User	Subscription.transactionEvents	id, amount, status	Yes	Clean	—
notificationEvents(userId)	Subscription	Yes	Standard User	Subscription.notificationEvents	id, type, createdAt	Yes	Clean	—
organizationEvents(orgId)	Subscription	Yes	Manager	Subscription.organizationEvents	id, action, actorId	Yes	Finding	API-002
webhookDeliveryEvents(webhookId)	Subscription	Yes	Manager	Subscription.webhookDeliveryEvents	id, status, attempt	Yes	Clean	—
auditEventStream(orgId)	Subscription	Yes	Administrator	Subscription.auditEventStream	id, action, actorId, targetId	Yes	Clean	—
reportGenerationEvents(reportId)	Subscription	Yes	Standard User	Subscription.reportGenerationEvents	id, status, progress	Yes	Clean	—
crossTenantEventProbe(orgId)	Subscription	Yes	Standard User	Subscription.organizationEvents (cross-tenant probe)	id, action, organizationId	Yes	Finding	API-002
__typename(User)	Query (introspection)	No	Visitor	Introspection.__typename	__typename	Yes	Clean	—
__type(Mutation)	Query (introspection)	No	Visitor	Introspection.__type	name, fields	Yes	Finding	API-028
__schema.directives	Query (introspection)	No	Visitor	Introspection.__schema.directives	name, description, locations	Yes	Clean	—
nestedQueryDepth8Probe	Query (complexity probe)	Yes	Standard User	Query.organization (depth-8 synthetic probe)	id, members, billing, invoices, auditEvents	Yes	Finding	API-013
recursiveMembersProbe	Query (complexity probe)	Yes	Standard User	Organization.members (recursive relationship probe)	id, members, members.members	Yes	Finding	API-014
aliasedUserLookupBatch	Query (aliased batch)	Yes	Standard User	Query.user (x3 aliases)	id, name, email	Yes	Finding	API-015
malformedInputErrorProbe	Mutation (error-handling probe)	Yes	Standard User	Mutation.updateUser (invalid input)	errors.extensions	Yes	Finding	API-021
deprecatedFieldProbe(legacyStatus)	Query (deprecated field)	Yes	Standard User	Subscription.legacyStatus (deprecated)	legacyStatus	Yes	Finding	API-022

61 APPENDIX D — ROLE & AUTHORIZATION MATRIX

The matrix below maps notable REST and GraphQL operations against all seven fictional OrionSphere roles. Cells marked "Finding" indicate an authorization outcome that deviated from the expected policy and is documented as a finding elsewhere in this report; "Denied" and "Allowed" reflect the expected/observed enforcement outcome for that role.

Operation	Visitor	Std. User	Billing	Manager	Support	Admin	API Integ.
GET /api/v1/organizations/{id}	Denied	Allowed (own org)	Allowed (own org)	Allowed (own org)	Allowed (own org)	Allowed	Allowed (own org)
GraphQL query organization(id)	Denied	Allowed (own org)	Allowed (own org)	Allowed (own org)	Allowed (own org)	Allowed	Allowed (own org)
GraphQL mutation updateOrganizationBilling	Denied	Finding (should be Denied)	Finding (should be Denied)	Allowed	Denied	Allowed	Denied
PATCH /api/v1/invoices/{id}	Denied	Denied	Allowed (own org)	Allowed	Denied	Allowed	Denied
GraphQL mutation updateInvoice	Denied	Denied	Allowed (own org)	Allowed	Denied	Allowed	Denied
POST /api/v1/reports/{id}/comments	Denied	Allowed	Allowed	Allowed	Allowed	Allowed	Denied
POST /api/v1/auth/mfa/recovery	Allowed (own account)	Allowed (own account)	Allowed (own account)	Allowed (own account)	Allowed (own account)	Allowed (own account)	Denied
GET /api/v1/files/{id}	Denied	Finding (should be own-file only)	Finding (should be own-file only)	Allowed	Allowed	Allowed	Finding (should be own-file only)
GraphQL query user(id) — sensitive fields	Denied	Finding (should be omitted)	Finding (should be omitted)	Finding (should be omitted)	Denied	Allowed	Denied
GraphQL organization.members.billing (nested)	Denied	Finding (should be Denied)	Allowed (own org)	Allowed	Denied	Allowed	Denied
GET /api/v1/accounts/{id} (legacy v1)	Denied	Finding (weaker than v2)	Finding (weaker than v2)	Allowed	Allowed	Allowed	Finding (weaker than v2)
POST /api/v1/webhooks/validate	Denied	Denied	Denied	Allowed	Denied	Allowed	Finding (unrestricted destination)
GraphQL mutation updateMemberRole	Denied	Denied	Denied	Allowed	Finding (should be Denied)	Allowed	Denied
GraphQL __schema introspection	Allowed (staging)	Allowed (staging)	Allowed (staging)	Allowed (staging)	Allowed (staging)	Allowed (staging)	Allowed (staging)
PATCH /api/v1/users/{id} (mass assignment fields)	Denied	Finding (hidden fields bound)	Denied	Allowed	Denied	Allowed	Denied
GraphQL mutation updateUser (mass assignment fields)	Denied	Finding (hidden fields bound)	Denied	Allowed	Denied	Allowed	Denied
GET /api/v1/organizations/{id}/members	Denied	Finding (over-exposed fields)	Allowed (own org)	Allowed	Allowed (own org)	Allowed	Allowed (own org)
POST /api/v1/subscriptions/{id}/discount/redeem	Denied	Denied	Finding (replay possible)	Allowed	Denied	Allowed	Denied
GET /api/v1/admin/users	Denied	Denied	Denied	Denied	Denied	Allowed	Denied
GET /api/v1/admin/organizations	Denied	Denied	Denied	Denied	Denied	Allowed	Denied
POST /api/v1/auth/login	Allowed	Allowed	Allowed	Allowed	Allowed	Allowed	Denied
POST /api/v1/auth/logout	Denied	Allowed	Allowed	Allowed	Allowed	Allowed	Denied
GET /api/v1/files (own listing)	Denied	Allowed	Allowed	Allowed	Allowed	Allowed	Allowed
POST /api/v1/webhooks	Denied	Denied	Denied	Allowed	Denied	Allowed	Allowed
GET /api/v1/reports	Denied	Allowed (own)	Allowed (own)	Allowed	Allowed (own)	Allowed	Allowed (own)
GraphQL mutation transferAccountOwnership	Denied	Denied	Denied	Allowed	Denied	Allowed	Denied

Legend: "Allowed (own org)" / "Allowed (own account)" indicates access is correctly scoped to the caller's own object; "Finding (…)" indicates the observed behavior deviated from the expected scoping and is documented under the referenced finding ID in Sections 45-49.

62 APPENDIX E — FINDING REGISTER

The complete register of all 42 fictional findings is provided below for traceability against Sections 45-49 and the Management Action Plan (Section 54).

Finding ID	Title	Severity	CVSS	Status	Ref.
API-001	Cross-Tenant BOLA Across REST and GraphQL Objects	**CRITICAL**	9.1 (Critical) — Illustrative CVSS v3.1-inspired score	Remediation In Progress	§16
API-002	GraphQL Privileged Mutation Authorization Bypass	**CRITICAL**	9.1 (Critical) — Illustrative CVSS v3.1-inspired score	Remediation In Progress	§17
API-003	Business Logic Authorization Bypass Allowing Unauthorized Financial State Manipulation	**CRITICAL**	9.6 (Critical) — Illustrative CVSS v3.1-inspired score	Remediation In Progress	§35
API-004	Stored Cross-Site Scripting Through API-Supplied Support Content	**HIGH**	8.2 (High) — Illustrative CVSS v3.1-inspired score	Open	§23
API-005	Account Recovery API Weakness	**HIGH**	7.7 (High) — Illustrative CVSS v3.1-inspired score	Open	§14
API-006	REST File Authorization Bypass	**HIGH**	7.1 (High) — Illustrative CVSS v3.1-inspired score	Open	§37
API-007	GraphQL Excessive Sensitive Data Exposure	**HIGH**	7.1 (High) — Illustrative CVSS v3.1-inspired score	Open	§25
API-008	GraphQL Nested Resolver Authorization Bypass	**HIGH**	7.7 (High) — Illustrative CVSS v3.1-inspired score	Remediation In Progress	§31
API-009	Legacy API v1 Authorization Weakness	**HIGH**	7.1 (High) — Illustrative CVSS v3.1-inspired score	Remediation Planned	§41
API-010	Server-Side Request Forgery Through Integration Webhook API	**HIGH**	8.1 (High) — Illustrative CVSS v3.1-inspired score	Remediation Planned	§36
API-011	Privilege Escalation Through Role Mutation	**HIGH**	7.6 (High) — Illustrative CVSS v3.1-inspired score	Remediation In Progress	§17
API-012	GraphQL Introspection Exposes Privileged Administrative Operations	**MEDIUM**	6.5 (Medium) — Illustrative CVSS v3.1-inspired score	Open	§27
API-013	GraphQL Query Complexity Enforcement Gap	**MEDIUM**	6.1 (Medium) — Illustrative CVSS v3.1-inspired score	Remediation Planned	§28
API-014	GraphQL Query Depth Control Gap	**MEDIUM**	5.9 (Medium) — Illustrative CVSS v3.1-inspired score	Remediation Planned	§29
API-015	GraphQL Batching / Alias-Based Excessive Object Retrieval	**MEDIUM**	5.8 (Medium) — Illustrative CVSS v3.1-inspired score	Open	§30
API-016	Insufficient Rate Limiting on Authentication Endpoints	**MEDIUM**	5.9 (Medium) — Illustrative CVSS v3.1-inspired score	Remediation Planned	§34
API-017	Access Token Revocation Weakness on Logout	**MEDIUM**	6.3 (Medium) — Illustrative CVSS v3.1-inspired score	Open	§20
API-018	Mass Assignment via REST and GraphQL Update Operations	**MEDIUM**	6.1 (Medium) — Illustrative CVSS v3.1-inspired score	Open	§26
API-019	Webhook Replay Protection Missing	**MEDIUM**	5.7 (Medium) — Illustrative CVSS v3.1-inspired score	Remediation Planned	§36
API-020	CORS Configuration Trusts Unnecessary Origins	**MEDIUM**	5.4 (Medium) — Illustrative CVSS v3.1-inspired score	Remediation Planned	§38
API-021	Verbose GraphQL Error Responses Reveal Internal Details	**MEDIUM**	5.3 (Medium) — Illustrative CVSS v3.1-inspired score	Open	§39
API-022	API Versioning Weakness — Inconsistent Deprecation Enforcement	**MEDIUM**	5.0 (Medium) — Illustrative CVSS v3.1-inspired score	Open	§41
API-023	Business Logic Replay in Discount Redemption Workflow	**MEDIUM**	5.8 (Medium) — Illustrative CVSS v3.1-inspired score	Open	§35

Finding ID	Title	Severity	CVSS	Status	Ref.
API-024	Technology / Version Disclosure via Response Headers	LOW	3.1 (Low) — Illustrative CVSS v3.1-inspired score	Open	§42
API-025	Missing Security Header Hardening (Permissions-Policy / Referrer-Policy)	LOW	2.6 (Low) — Illustrative CVSS v3.1-inspired score	Remediation Planned	§42
API-026	Unused / Unnecessary HTTP Methods Enabled	LOW	3.0 (Low) — Illustrative CVSS v3.1-inspired score	Open	§42
API-027	Minor Input Validation Inconsistencies Across Endpoints	LOW	3.4 (Low) — Illustrative CVSS v3.1-inspired score	Open	§21
API-028	Non-Sensitive GraphQL Schema Metadata Exposure	LOW	2.4 (Low) — Illustrative CVSS v3.1-inspired score	Open	§27
API-029	Residual Debug Logging in Client Bundle	LOW	2.8 (Low) — Illustrative CVSS v3.1-inspired score	Open	§42
API-030	API Documentation / Enforcement Mismatch	LOW	2.7 (Low) — Illustrative CVSS v3.1-inspired score	Open	§40
API-031	Stale Endpoint Metadata in API Registry	LOW	2.3 (Low) — Illustrative CVSS v3.1-inspired score	Open	§40
API-032	Incomplete Session Artifact Cleanup on Logout	LOW	2.5 (Low) — Illustrative CVSS v3.1-inspired score	Open	§20
API-033	Public GraphQL Documentation Portal (Intended Exposure — Review Recommended)	INFORMATIONAL	N/A — Informational observation, no CVSS score assigned	Open	§40
API-034	Introspection Intentionally Available in Non-Production Environment	INFORMATIONAL	N/A — Informational observation, no CVSS score assigned	Open	§27
API-035	Non-Sensitive Schema Metadata Present in Build Artifacts	INFORMATIONAL	N/A — Informational observation, no CVSS score assigned	Open	§40
API-036	Security.txt Not Present	INFORMATIONAL	N/A — Informational observation, no CVSS score assigned	Open	§40
API-037	API Inventory / Asset-Management Improvement Opportunity	INFORMATIONAL	N/A — Informational observation, no CVSS score assigned	Open	§40
API-038	Logging Coverage Recommendation for Authorization Events	INFORMATIONAL	N/A — Informational observation, no CVSS score assigned	Open	§31
API-039	Monitoring Enhancement Opportunity for GraphQL Resolver Errors	INFORMATIONAL	N/A — Informational observation, no CVSS score assigned	Open	§39
API-040	API Lifecycle Governance Recommendation	INFORMATIONAL	N/A — Informational observation, no CVSS score assigned	Open	§41
API-041	Deprecated Endpoint Cleanup Backlog	INFORMATIONAL	N/A — Informational observation, no CVSS score assigned	Open	§41
API-042	Recommend Automated Authorization Regression Testing	INFORMATIONAL	N/A — Informational observation, no CVSS score assigned	Open	§15

63 APPENDIX F — TEST ACCOUNT REGISTER

The fictional test accounts below were used throughout this assessment. No real passwords are recorded in this report; all credential values are placeholders.

Fictional Account	Role	User ID	Organization	Credential
api.user01@example.test	Standard User	USER-6001	ORG-2001	*[Fictional Test Credential]*
api.billing01@example.test	Billing User	USER-6010	ORG-2001	*[Fictional Test Credential]*
api.manager01@example.test	Manager	USER-6020	ORG-2001	*[Fictional Test Credential]*
api.support01@example.test	Support Agent	USER-6030	ORG-2001	*[Fictional Test Credential]*
api.admin01@example.test	Administrator	USER-6040	Platform-Wide	*[Fictional Test Credential]*
api.integration01@example.test	API Integration User	USER-6050	ORG-2001	*[Fictional Test Credential]*
api.user02@example.test	Standard User	USER-6102	ORG-2002	*[Fictional Test Credential]*

Illustrative Purpose (per account):

- api.user01@example.test — Primary standard-user account, ORG-2001 — REST and GraphQL testing.
- api.billing01@example.test — Invoice, subscription, and transaction workflow testing.
- api.manager01@example.test — Member/role management and webhook configuration testing.
- api.support01@example.test — Report and support-content submission testing.
- api.admin01@example.test — Administrative REST and GraphQL function testing.
- api.integration01@example.test — API-key/service-account and integration-scope testing.
- api.user02@example.test — Secondary tenant account (ORG-2002) — cross-tenant / BOLA isolation testing.

All accounts, identifiers, and credential placeholders above are fictional. No real credentials are recorded anywhere in this report.

64 APPENDIX G — METHODOLOGY NOTES

This appendix provides supplementary detail on how each stage of TMG Security's 26-stage API testing methodology (Section 09) was applied during this fictional assessment.

REST Testing

REST endpoints were enumerated via authenticated browsing, the published OpenAPI/Swagger documentation, and systematic method/route variant testing, producing the endpoint register in Appendix B.

GraphQL Testing

The GraphQL schema was mapped via introspection (available in this fictional staging environment) and manual query construction, producing the operation register in Appendix C. GraphQL testing was treated as a first-class discipline distinct from REST testing throughout this engagement, given GraphQL's materially different authorization surface.

Role-Based Testing

Each testing category was repeated across all seven relevant roles to independently validate horizontal and vertical authorization boundaries, rather than relying on a single privileged account throughout, producing the role/authorization matrix in Appendix D.

Authorization Testing

Object-level, function-level, field-level, and tenant-isolation authorization were tested as distinct disciplines (Sections 16-19), each with its own dedicated test cases in Appendix A.

Resolver Testing

GraphQL resolver authorization was tested at both the root and nested-field level, deliberately probing whether authorization checks performed at a parent object propagate to child resolvers — the discipline that produced this report's most technically significant findings (API-002, API-008, API-011).

Manual Validation

Every candidate finding was manually validated by a TMG Security fictional tester before inclusion in this report; no automated-scanner output was included without manual confirmation.

Business Logic Analysis

Multi-step fictional workflows — subscription changes, discount redemption, invoice/refund state transitions, account-ownership transfer — were mapped and tested for state-manipulation and sequence-bypass conditions that conventional endpoint-level testing does not reliably surface.

Evidence Collection

All fictional evidence was captured and referenced using the evidence-reference identifiers cited throughout Sections 45-49, maintaining traceability from finding to evidence, including dual REST/GraphQL evidence panels where a finding was reproduced through both interfaces.

Reporting

Findings were documented using the consistent format defined in Section 44, then reconciled across the Dashboard, Findings Summary, Detailed Findings, Finding Register, and Management Action Plan to ensure full internal consistency.

Retest Methodology

A structured, role-based, REST-and-GraphQL-aware retest methodology (Section 55) was defined for use following remediation, with closure criteria requiring reproduction to fail and regression testing to raise no new concern.

65 FINAL CONCLUSION

Illustrative Security Assessment Conclusion

> *"OrionSphere Enterprise Platform demonstrates a strong baseline in several areas — MFA, refresh-token rotation, TLS enforcement, and root-level authorization on the majority of operations tested — but the fictional assessment identified meaningful authorization weaknesses, GraphQL-specific trust-boundary issues, business-logic weaknesses, and excessive API data exposure that require prioritized remediation."*

The three Critical and eight High fictional findings in this sample report require immediate attention and should be the organization's first priority, given their concentration around authorization and trust-boundary enforcement — and, distinctively for an API-focused assessment, around whether authorization checks made at a root REST endpoint or GraphQL query correctly propagate to nested fields and mutations. The twelve Medium findings should be integrated into the standard remediation backlog and addressed within the illustrative 31-60 day window. The nine Low and ten Informational findings support defense-in-depth and API-governance improvements and do not represent an immediate exploitation path in this fictional sample, but should not be indefinitely deferred.

TMG Security's simulated recommendation is that OrionSphere invest specifically in centralized, resolver-level authorization enforcement for the GraphQL API — rather than per-mutation checks implemented inconsistently — as the single highest-leverage architectural improvement suggested by this fictional assessment's findings. TMG Security further recommends that OrionSphere retest all Critical and High findings following remediation, per the methodology in Section 55, before considering this fictional assessment's core concerns resolved. A follow-on assessment cycle — informed by the API-lifecycle and monitoring recommendations in Section 56 and Appendix G — is recommended within twelve months.

THIS IS A FICTIONAL SAMPLE ASSESSMENT.

66 CONTACT & DISCLAIMER

Cybersecurity | *SOC* | *VAPT* | *GRC* | *MDR* | *Compliance Solutions* | *Corporate Trainings*
Web: www.tmgsec.com **Email:** security@tmgsec.com
USA Office: 117 South Lexington Street, STE 100, Harrisonville, MO 64701
Support: support@tmgsec.com | **Phone:** +1 (816) 793-1929
WhatsApp: +1 (480) 680-0342

Final Disclaimer

FICTIONAL SAMPLE REPORT — NOT AN ACTUAL CLIENT API PENETRATION TEST

This report was created by TMG Security solely to demonstrate API penetration-testing methodology, technical reporting standards, evidence presentation, and remediation guidance across REST and GraphQL interfaces. OrionSphere Technologies, Inc. and the OrionSphere Enterprise Platform are fictional. No real client environment was assessed. No real REST or GraphQL API infrastructure was accessed. No real credentials, API keys, personal data, production systems, cloud resources, or customer information were accessed. All GraphQL schemas, queries, and mutations are synthetic. All vulnerabilities, requests, responses, screenshots, findings, severity ratings, CVSS scores, and remediation actions are illustrative. No real exploitation occurred.