THREAT MODEL EXCERPT

Threat Model and Attack-Surface Assessment

Anonymized and abridged sample from a source-informed product security engagement.

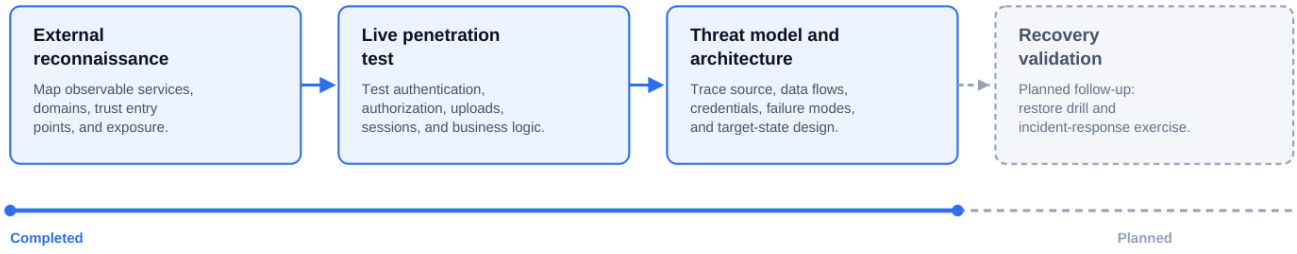
CLIENT	PLATFORM	ASSESSMENT	DELIVERABLE
Confidential European B2B software company	Multi-tenant content and website platform	Static source and configuration review with carried-over live evidence	Threat model, risk register, target architecture, implementation specs

Engagement context

The assessed product was a multi-tenant platform used to create, manage, and publish client websites. The engagement examined the system as a complete operating model rather than as a collection of isolated endpoints: application code, authorization, data storage, rendering, deployment, credentials, recovery, and incident readiness were assessed together.

> **HOSTING-SUITABILITY QUESTION**
>
> Could the platform responsibly host real client data at its current stage, and what evidence or controls would be required before that decision became defensible?

Multi-phase engagement



The threat model was the third completed phase of a broader security engagement. Recovery validation was intentionally deferred until an independent backup system existed.

Scope and evidence boundary

The review was evidence-disciplined: every material state claim was classified according to what had actually been observed. Repository configuration was not treated as proof of production state, and carried-over live results were identified separately from source-derived conclusions.

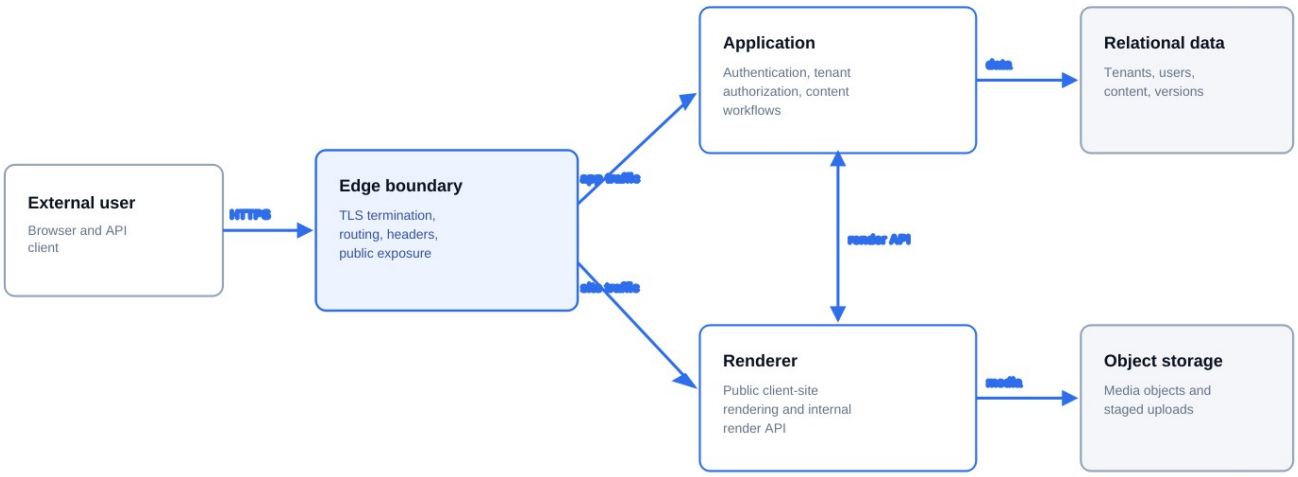
Evidence class	Meaning
Verified	Demonstrated directly from source/configuration or recorded live evidence.
Client-confirmed	Explicitly confirmed by the platform owner.
Inferred	Supported by evidence but dependent on an unverified runtime or organizational condition.
Unknown	Could not be determined; converted into a concrete evidence request.

Methodology

1. Inventory the repository, runtime components, assets, identities, and external dependencies.
2. Reconstruct trust boundaries and data flows across the application, renderer, edge, database, object storage, queues, and operators.
3. Define security invariants for tenant isolation, authorization, media publication, credential authority, deployment provenance, recovery, and auditability.
4. Trace attacker-controlled sources to security-sensitive sinks while accounting for every observed control on the path.
5. Build STRIDE scenarios and attack trees for each subsystem, including post-compromise and destructive-event paths.
6. Separate confirmed vulnerabilities from control weaknesses, review concerns, assumptions, and unknowns.
7. Score findings using explicit likelihood and impact criteria, then map each finding to remediation, acceptance tests, ownership, and residual risk.

> **CORE EVIDENCE RULE**
>
> A condition was described as exploitable only when a plausible attacker-controlled source could be traced to a security-sensitive sink and every observed control on the path had been considered. Incomplete traces remained review concerns or unknowns.

Current-state trust boundaries



The simplified public diagram omits identifiers and implementation details. The original assessment included endpoint inventories, credential maps, data-lifecycle matrices, and subsystem-specific threat trees.

Assets and security invariants

Asset class	Primary concern	Representative invariant
Tenant content and media	Confidentiality, integrity, availability	A user can access or modify only resources belonging to an authorized tenant.
Accounts and sessions	Confidentiality and integrity	Session and recovery tokens cannot be reused outside their intended lifetime and scope.
Renderer and internal credentials	Confidentiality and integrity	One compromised component should not automatically grant platform-wide tenant access.
Production data and recovery copies	Availability and integrity	A compromised application process cannot destroy the only usable recovery copy.
Deployment artifacts	Integrity and accountability	Every production artifact can be traced to a reviewed commit and immutable build identity.
Audit evidence	Integrity and availability	Security-relevant actions create durable records outside the authority of the initiating runtime identity.

Threat actors and assume-breach analysis

The assessment considered both pre-compromise and post-compromise actors. This avoided treating strong application authorization as a substitute for infrastructure containment.

- Unauthenticated external attacker probing public services and edge configuration.
- Authenticated low-privilege tenant user attempting horizontal or vertical privilege escalation.
- Compromised staff or platform-administrator account.
- Attacker with code execution inside the application or renderer container.
- Compromised registry, deployment workstation, DNS account, or infrastructure credential.
- Operator error, destructive automation, ransomware, or physical-host failure.

Actor	Capability ceiling	Primary findings this actor reaches
Unauthenticated external	Low, automated/scripted	Edge misconfiguration, unauthenticated telemetry, any anonymously reachable storage path.
Authenticated low-privilege tenant	Medium, manual, one legitimate account	Tenant-scoping gaps, renderer injection reachable via content fields, mass-assignment edge cases.
Compromised staff / admin	High, full platform scope by design	Every tenant simultaneously; blast radius is a credential-strength and MFA question, not an authorization-logic one.
Code execution in a container	High within that container's network reach	Whatever data stores, credentials, and internal services that specific container can reach; bounded by identity scoping.
Compromised infra credential	High, outside the application's own authorization model entirely	Deployment integrity, domain takeover, supply-chain persistence; not caught by any application-layer control.
Operator error / destructive automation	Full production scope, no attacker required	Everything the acting identity can touch; recovery findings, not authorization findings.

Anatomy of one finding

Portfolio samples usually show findings at summary depth, which is how they appear below. To be transparent about the actual rigor behind them, here is one representative finding carried through the full depth used throughout the real report: named security invariant, traced attack narrative, blast radius, detection capability, and residual risk after remediation, not just a problem statement and a fix.

TENANT-02 Cross-tenant object read via an unscoped lookup function **HIGH**

FINDING STATUS
Confirmed vulnerability

EVIDENCE LABEL
Verified, traced to a specific service function and its call site

AFFECTED INVARIANT
A user can access or modify only resources belonging to an authorized tenant.

REQUIRED SECURITY BEHAVIOUR
Every resource lookup by identifier must be scoped to the caller's own tenant before any data is returned, independent of whether the identifier itself is guessable.

OBSERVED IMPLEMENTATION
A shared lookup helper resolved records by primary key only. The route calling it enforced authentication and a capability check, but the capability check validated that the caller could perform the action in general, not that the specific record belonged to the caller's tenant.

ATTACKER-CONTROLLED SOURCE
A record identifier obtainable through an otherwise-correctly-scoped listing endpoint the caller does have legitimate access to elsewhere in the product, which is enough to construct a valid identifier for probing.

SECURITY-SENSITIVE SINK
The unscoped lookup helper, called from three separate routes; only one of the three was independently patched in a prior change, the other two remained exposed.

NUMBERED ATTACK NARRATIVE
1. Actor authenticates normally as a low-privilege member of Tenant A, no special access required. 2. Actor requests a record belonging to Tenant B by substituting a foreign identifier into an otherwise-legitimate request. 3. The route's capability check passes, since it only confirms the actor holds the relevant role, not that the specific record is theirs. 4. The unscoped lookup helper returns the full record with no tenant predicate applied. 5. The response is returned to the actor with no distinguishing signal from a same-tenant request, and no audit event flags the tenant mismatch.

OBSERVED CONTROLS AND EFFECTIVENESS
Authentication: effective, blocks unauthenticated access entirely. Role-based capability check: effective for its own purpose, but not a substitute for object-level tenant scoping and was not treated as one anywhere else in the codebase. Audit logging: present but does not record which tenant a resource belonged to relative to the requester, so this class of access is not distinguishable after the fact from legitimate access.

ATTACK VARIANTS
Unauthenticated: not possible, blocked by authentication. Same-tenant: not applicable, no boundary crossed. Cross-tenant, low-privilege: demonstrated, the primary variant. Insider: identical mechanism, removes the need to obtain an account first. Automated/internet-scale: unlikely as a first move, better suited to a targeted actor who already holds one legitimate account.

BLAST RADIUS
Read access to another tenant's records through this specific object class; the same helper's write path was checked separately and found to enforce scoping correctly, so this instance was read-only. Multi-tenant in scope, since the vulnerable helper is shared code reused across tenant contexts rather than isolated to one deployment.

DETECTION CAPABILITY
None at the time of assessment: no alert existed for a resolved-tenant-mismatch condition. Recommended: emit a security event whenever the resolved record's tenant differs from the requester's own, since there is no legitimate reason for that condition to occur by design.

CONTAINMENT CAPABILITY
No partial mitigation existed short of the code fix; the finding was treated as its own containment action once identified.

IMMEDIATE CONTAINMENT
Patch the shared lookup helper directly to require and apply a tenant predicate, rather than patching each of the three call sites separately, which would leave the same class of bug reachable from any future caller.

PREFERRED CORRECTIVE REMEDIATION
Add tenant scoping inside the shared helper itself so every current and future caller inherits the fix, plus a regression test asserting that a cross-tenant identifier returns not-found rather than the record.

VERIFICATION AND ACCEPTANCE CRITERIA
A request for a foreign-tenant identifier returns a not-found response with no data in the body; same-tenant access continues to function correctly across all three call sites; the new regression test is added to the suite that runs on every change to this module.

RESIDUAL RISK
Low after remediation: the fix closes the vulnerable helper for all current and future callers rather than one call site, so a structurally identical bug elsewhere would require a genuinely new lookup path rather than a missed call site of this one.

Selected findings

The following examples are sanitized. Identifiers, implementation details, and exploit mechanics have been generalized to prevent the sample from functioning as a disclosure of the client environment.

RECOVERY-01 — No independent, restore-tested production recovery system — HIGH

EVIDENCE
Client-confirmed

CURRENT STATE
Application-level version history existed, but it was stored with production data and did not provide disaster recovery. No separately credentialed, off-box restore path had been demonstrated.

BUSINESS CONSEQUENCE
A destructive intrusion, ransomware event, host failure, or operator mistake could cause permanent multi-tenant data loss.

REQUIRED CHANGE
Create off-box backups for relational data, object storage, deployment configuration, and recovery documentation. Separate backup authority from production identities and introduce immutability or versioning.

CLOSURE EVIDENCE
A complete restore into a clean environment succeeds, critical user journeys are verified, actual recovery time is recorded, and production credentials cannot delete retained recovery points.

DEPLOY-01 — Production could not be attributed to the reviewed source revision — HIGH

EVIDENCE
Verified and inferred

CURRENT STATE
The reviewed repository, intended deployment configuration, and observed runtime could not be joined through a recorded commit, build, digest, and deployment event. Mutable image references were part of the process.

BUSINESS CONSEQUENCE
Source-derived security conclusions could not be assumed to describe production; incident containment and deterministic rollback would depend on operator memory.

REQUIRED CHANGE
Build from a clean commit, capture the pushed image digest, sign and deploy by digest, expose build identity at runtime, and retain a known-good rollback artifact.

CLOSURE EVIDENCE
The running digest maps to an approved commit and signed build record; deployment and rollback are reproducible without relying on local workstation history.

CREDENTIAL-01 — Application identities carried unnecessary administrative authority — MEDIUM

EVIDENCE
Verified configuration

CURRENT STATE
The request-serving application used broad database and object-storage identities rather than role-specific runtime credentials.

BUSINESS CONSEQUENCE
Application compromise would inherit schema-level and storage-administration power, amplifying confidentiality, integrity, and recovery impact.

REQUIRED CHANGE
Separate owner, migration, runtime, worker, backup, and storage-administration identities. Grant only required actions and verify denied administrative operations.

CLOSURE EVIDENCE
Normal application flows succeed while runtime identities fail negative tests for schema changes, role administration, bucket-policy changes, identity creation, and backup deletion.

TENANT-01 — One authenticated write path failed to verify the target tenant — MEDIUM

EVIDENCE
Confirmed vulnerability

CURRENT STATE
A create operation accepted a client-selected tenant identifier without confirming that the actor held the required capability on that exact tenant. Read paths remained scoped.

BUSINESS CONSEQUENCE
A low-privilege user could pollute another tenant or a shared library, creating cross-tenant integrity loss without mass confidentiality exposure.

REQUIRED CHANGE

Require explicit target-tenant capability before persistence and reserve global writes for platform administrators.

CLOSURE EVIDENCE

Foreign-tenant and global writes return 403, no record is created after denial, and same-tenant authorized workflows continue to pass.

RENDER-01 Stored renderer injection through insufficiently constrained style input **MEDIUM**

EVIDENCE

Confirmed vulnerability

CURRENT STATE

A privileged content field could cross a style-context boundary, while the renderer policy permitted inline script execution.

BUSINESS CONSEQUENCE

A compromised or malicious privileged user could create persistent browser-side execution on a published client site.

REQUIRED CHANGE

Reject unsafe characters before persistence, encode defensively at output, and replace permissive inline-script policy with per-response nonces.

CLOSURE EVIDENCE

Unsafe values are rejected deterministically, legacy payloads cannot execute, inline-script allowance is absent, and nonces vary per response.

QUEUE-01 Background job payload accepted without schema validation **LOW**

EVIDENCE

Verified, no exploitable sink demonstrated

CURRENT STATE

A background job accepted a user-influenced field without validating it against a schema before use in downstream processing.

BUSINESS CONSEQUENCE

Limited on its own; a malformed or attacker-shaped field could reach a templating or formatting step, with no code-execution sink identified downstream in this review.

REQUIRED CHANGE

Validate job payloads against an explicit schema at enqueue time rather than trusting shape at consume time.

CLOSURE EVIDENCE

Malformed payloads are rejected at enqueue with no job created; existing legitimate workflows continue to function.

EDGE-02 Missing baseline HTTP security headers **MEDIUM**

EVIDENCE

Verified live

CURRENT STATE

The public edge did not set a content-security policy, frame-ancestors restriction, or several other standard hardening headers.

BUSINESS CONSEQUENCE

Reduces defense-in-depth against clickjacking and compounds the impact of RENDER-01 if the render-layer fix is ever incomplete or bypassed.

REQUIRED CHANGE

Add a restrictive header set at the edge, independent of application-layer fixes, so the protection does not depend on every application route setting it correctly itself.

CLOSURE EVIDENCE

Headers present and correctly scoped across all tested public and authenticated routes.

DNS-01 Domain and third-party account access could not be independently verified **MEDIUM**

EVIDENCE

Unknown, converted into a client evidence request

CURRENT STATE

Registrar, DNS provider, and email-provider account control and multi-factor status were outside what a repository-and-live-evidence review can observe directly.

BUSINESS CONSEQUENCE

A single point of failure here (domain hijack, DNS takeover, or email-provider compromise) can undermine every other control in this report, including password-reset integrity, regardless of how well the application itself is built.

REQUIRED CHANGE

Client to confirm who holds access to each account, whether multi-factor authentication is enforced, and whether access is individual or shared.

CLOSURE EVIDENCE

Written confirmation from the platform owner, treated as client-confirmed evidence rather than independently verified, and noted as such in the register.

Ruled-out hypotheses

Recording what was investigated and not confirmed is as important as recording what was. This avoids leaving stakeholders uncertain about whether a plausible-sounding risk was actually checked.

Hypothesis	Why it initially seemed plausible, and why it was ruled out
Broad SQL injection across data-entry forms	The application accepts a large amount of free-text content input, a common injection surface elsewhere. Ruled out: all reviewed query construction used parameterized queries with no string-built SQL found anywhere, and live injection probes did not alter query semantics.
Straightforward vertical privilege escalation via role parameter tampering	Several APIs accept a JSON body that could plausibly include a role or permission field. Ruled out: mass-assignment testing confirmed injected role, ownership, and status fields are silently ignored server-side across every tested write path.
Server-side request forgery via a URL-shaped field in the renderer	The renderer's job is fetching and rendering remote content, a common SSRF shape. Ruled out within tested scope: no code path accepts an arbitrary attacker-controlled URL for a server-side fetch; the one remote-fetch capability found was scoped to a fixed, non-attacker-controlled path.

Representative inventory samples

The full assessment included a complete, non-sampled endpoint matrix, credential and privilege matrix, and tenant-isolation matrix. The rows below are representative examples, not the complete tables, generalized to avoid disclosing the client's actual route structure.

Endpoint authorization matrix (sample rows)

Endpoint class	Auth requirement	Tenant scoping	Evidence
Public render read path	Bearer render credential	Scoped to credential's project	Verified
Project content write path	Session + capability check	Scoped, confirmed live and in source	Verified
Cross-tenant lookup helper (TENANT-02)	Session + capability check	Missing at object level	Verified, see finding
Staff administrative path	Session + platform-staff check	Intentionally unscoped by design	Verified
Health/status endpoint	None	Not applicable	Verified, no sensitive data returned

Credential and privilege matrix (sample rows)

Identity	Scope observed	Shared or individual	Evidence
Application database role	Broader than required; schema-level authority present	Shared service credential	Verified, see CREDENTIAL-01
Object-storage runtime key	Read/write on the working bucket only	Shared service credential	Verified
Renderer internal credential	Scoped to render operations, not general platform access	Per-deployment	Verified
Domain registrar account	Unknown	Unknown	Unknown, see DNS-01

Coverage ledger (sample)

Each subsystem in the full assessment was tracked to completion across recon, control tracing, threat construction, and finding validation before being marked complete. A sample of that tracking:

Subsystem	Entry-point enumeration	Control tracing	Threat construction	Status
Authentication and session	Complete	Complete	Complete	Closed, no open findings
Tenant data access	Complete	Complete	Complete	Closed, see TENANT-01 / TENANT-02
Renderer	Complete	Complete	Complete	Closed, see RENDER-01
Background queue	Complete	Complete	Partial, no dynamic reproduction performed	Open, see QUEUE-01
Third-party account access	Not assessable from available evidence	Not assessable	Not assessable	Open, converted to client evidence request, see DNS-01

Effective controls

The report also documented controls that held up. This was important for avoiding a fear-driven result and for focusing remediation on actual boundaries rather than theoretical checklist gaps.

- Server-side tenant authorization was consistently implemented across the principal read paths and resisted live cross-tenant testing.
- Database access used parameterized ORM operations in the reviewed code paths.
- User registration was disabled and session cookies used secure attributes.
- Rate limiting, mass-assignment resistance, upload integrity checks, and unsafe-link validation performed effectively during separate live testing.
- Several hypotheses, including broad SQL injection and straightforward privilege escalation, were investigated and ruled out within the tested scope.

Proposed target state

The target architecture turned findings into explicit operating boundaries: immutable deployment identity, least-privilege runtime identities, off-box telemetry, project-scoped internal credentials, and recovery authority separated from production.

Implementation-ready specification model

Each material finding was expanded beyond a generic recommendation. The full report used a repeatable implementation handoff structure:

Specification field	Purpose
Current state	What is known, how it was evidenced, and what remains unknown.
Required security behaviour	The invariant the implementation must enforce.
Proposed implementation	A concrete technical direction, clearly labelled as unimplemented guidance.
Migration and rollback	How to introduce the control without losing recoverability.
Required tests	Positive, negative, abuse-case, and failure-mode coverage.
Acceptance and closure criteria	Objective evidence required before the finding may be closed.
Owner and residual risk	Who is accountable and what risk remains after remediation.

Assessment conclusion

> **DECISION**
>
> The application layer was materially stronger than the operational environment. Hosting real client data was not considered defensible until independent recovery, deployment attribution, valid edge trust, least-privilege credentials, and the confirmed application defects were addressed.

The principal value of the engagement was not the number of findings. It was the ability to separate controls that genuinely worked from conditions that changed the business risk of operating the platform, then convert those conditions into engineering work that could be verified.

Limitations of this public sample

The original report contained substantially more analysis. Identifiers and finding mechanics have been altered or omitted. Proposed guidance is not completed engineering work; closure requires deployment evidence and retesting.