

TravelHub

Vulnerability Assessment & Penetration Test · Authorised whitebox engagement

Multi-tenant flight booking — Go API, Angular SPA, PostgreSQL · 12 September 2026



Core access controls held; findings are operational and defence-in-depth

Tenant isolation, SQL parameterisation, mass-assignment and CORS defences all resisted direct attack. 2 High, 5 Medium, 3 Low and 1 Informational finding were identified, chiefly around credential handling, least privilege and transport hardening.

ENGAGEMENT	TARGETS	FINDINGS
Whitebox (source-assisted)	API :8080 · SPA :4200 · PostgreSQL 17	2 High · 5 Medium · 3 Low · 1 Info
CRITICAL / RCE (PRE-AUTH)	METHOD	DATA HANDLING
None	Static review + dynamic testing	Dev DB only; demo data restored

1 Executive summary

TravelHub is architecturally sound on the issues that most often sink a multi-tenant application. Tenant isolation held under direct testing, SQL is parameterised throughout, the JSON decoder rejects unknown fields, request bodies are size-capped, and the Angular client escapes interpolated data by default. The weaknesses found are **operational and defence-in-depth**, not broken access control.

The two most important are a **database superuser connection**, which turns any application compromise into total database control and nullifies the application-layer tenant isolation, and a **timing side-channel that backfired**: the mitigation intended to hide which usernames exist actually makes them *more* distinguishable. None of the findings is a remotely exploitable pre-authentication breach on the current localhost deployment; several become serious the moment the app is exposed beyond localhost.

#	SEVERITY	FINDING
1	HIGH	Application connects to PostgreSQL as a superuser (no least privilege)
2	HIGH	Username enumeration via a timing oracle — equalisation mitigation is miscalibrated and inverts
3	MEDIUM	Bcrypt cost factor 6, below the OWASP-recommended minimum of 10
4	MEDIUM	No rate limiting or account lockout on authentication
5	MEDIUM	Tenant enumeration via timing (unknown tenant short-circuits before password check)
6	MEDIUM	No HTTP security headers (HSTS, nosniff, frame-options, CSP...)
7	MEDIUM	Database connections run in plaintext (<code>sslmode=disable</code>)
8	LOW	Real database password committed in <code>config_test.go</code>
9	LOW	Credentials held reversibly in localStorage and replayed each request
10	LOW	API stores markup verbatim — no server-side output encoding (defence-in-depth)
11	INFO	<code>RequireAdmin</code> middleware wired to no route; role model unenforced

2 Controls that resisted attack

These were actively attacked and held. Recorded so the report is a fair account, not only a list of faults.

CONTROL	TEST PERFORMED	RESULT
SQL injection	' OR '1'='1 , '; DROP TABLE bookings;-- , UNION in search params	Parameterised
ORDER BY injection	sort=(SELECT 1) , sort=price;DROP	Whitelisted
Mass assignment	POST booking with status, totalAmount, bookingReference, tenantId	Rejected 400
Cross-tenant read (IDOR)	Fetch another agency's booking / flight by id	404 not-found
Cross-tenant credentials	Skyline password against Globetrotter tenant	401
Large-body DoS	3 MB request body	400 — 1MB cap
CORS	Reflect Origin: evil.example.com	Not reflected
Error verbosity	Malformed UUID / JSON	Generic
Reflected XSS	Angular template sinks	Auto-escaped

3 Detailed findings

FINDING-1 Application connects to PostgreSQL as a superuser HIGH

CWE-250 Execution with Unnecessary Privileges · OWASP A04:2021 Insecure Design

EVIDENCE

The application connects with the `backend/.env` credentials (`DB_USER=postgres`), which is the cluster superuser:

```
rolname | rolsuper | rolcreatedb | rolcreatorole | rolbypassrls
postgres | t         | t           | t             | t
```

IMPACT

Tenant isolation is enforced entirely in application code (every query filters `tenant_id`), so the DB connection is the last line of defence — and it has none. Any query-control flaw becomes full read/write/DDL over *every* tenant, plus `COPY ... TO PROGRAM` command execution and `rolbypassrls` defeating any future row-level security. The superuser connection silently converts application bugs into total compromise.

RECOMMENDATION

Connect as a least-privilege role; keep migrations under a separate higher-privilege role the service never uses.

```
CREATE ROLE travelapp LOGIN PASSWORD '...';
GRANT CONNECT ON DATABASE travelhub TO travelapp;
GRANT USAGE ON SCHEMA public TO travelapp;
GRANT SELECT, INSERT, UPDATE, DELETE ON ALL TABLES IN SCHEMA public TO travelapp;
-- NOT: SUPERUSER, CREATOROLE, CREATEDB, TRUNCATE, DDL
```

FINDING-2 Username enumeration via an inverted timing oracle

HIGH

CWE-208 Observable Timing Discrepancy · CWE-203 Observable Discrepancy

EVIDENCE

The auth middleware runs a dummy bcrypt comparison for unknown users to equalise timing — a correct instinct — but the dummy hash is **cost 10** while the real stored hashes are **cost 6**. Verifying a cost-10 hash costs $\sim 16\times$ a cost-6 hash, so the "user does not exist" path is far *slower*. Measured (mean of 30):

```
valid user + wrong password : 5.1 ms  
unknown user, valid tenant  : 47.3 ms ← ~9x slower
```

IMPACT

An unauthenticated attacker distinguishes valid from invalid usernames with a single request each, at a $\sim 9\times$ margin needing no statistics — enumerating the tenant's user directory for phishing or focused brute force. The mitigation did not merely fail; it created a *stronger* oracle than doing nothing, because the two branches now use different work factors.

RECOMMENDATION

Make the dummy hash's cost identical to the real one and pin both. Better: verify against a per-deployment constant hash generated at startup with the exact configured cost, so the two branches can never drift apart.

FINDING-3 Bcrypt cost factor below recommended minimum MEDIUM

CWE-916 Password Hash With Insufficient Computational Effort

EVIDENCE

All stored hashes carry the `$2a$06$` prefix (cost 6); `002_seed.sql` relies on `pgcrypto` `gen_salt('bf')`, whose default is 6. The test helper uses `bcrypt.MinCost` (4), so production hashing strength is never exercised by the suite.

IMPACT

Cost 6 is 16× weaker than the OWASP cost-10 floor and 64× weaker than cost 12. Against an offline attacker who obtains the hashes (see Finding 1) it materially shortens crack time.

RECOMMENDATION

Hash at **cost 12 in Go application code** (`bcrypt.GenerateFromPassword(pw, 12)`) rather than in SQL, so the work factor lives with the app and stays consistent. Support rehash-on-login for old hashes.

FINDING-4 No rate limiting or account lockout on authentication MEDIUM

CWE-307 Improper Restriction of Excessive Authentication Attempts · A07:2021

EVIDENCE

25 failed logins in 144 ms → 25x HTTP 401 (no 429, no delay, no lockout)

IMPACT

Online brute force and credential stuffing are unthrottled. Combined with Finding 2 (valid usernames handed to the attacker) and Finding 3 (weaker hashes), account takeover is realistic once exposed beyond localhost.

RECOMMENDATION

Add per-IP and per-account throttling with exponential backoff and temporary lockout, as a middleware in the existing `chain` (token-bucket keyed on IP + `X-Tenant` + email). Return `429` with `Retry-After` ; log repeated failures for alerting.

FINDING-5 Tenant enumeration via timing MEDIUM

CWE-204 Observable Response Discrepancy

EVIDENCE

An unknown tenant short-circuits before any password check, so it is far faster than a known tenant with bad credentials:

```
unknown tenant      : 0.8 ms
valid tenant, wrong pw : 5.1 ms
```

IMPACT

Reveals which agency slugs exist — including *suspended* tenants, which the public `GET /api/tenants` endpoint deliberately hides. Lower impact than Finding 2 since active tenants are already public by design.

RECOMMENDATION

Perform the dummy bcrypt comparison on the unknown-tenant path too, so all failure modes cost the same. Fold into the Finding 2 fix.

FINDING-6 Missing HTTP security headers MEDIUM

CWE-693 Protection Mechanism Failure · A05:2021 Security Misconfiguration

EVIDENCE

API responses carry only `Content-Type`, `Content-Length`, `Date`. Absent: `Strict-Transport-Security`, `X-Content-Type-Options`, `X-Frame-Options`, `Content-Security-Policy`, `Referrer-Policy`, and `Cache-Control` on authenticated JSON. The SPA ships no CSP.

IMPACT

No MIME-sniffing protection, no clickjacking defence, no CSP to blunt a future XSS, and authenticated responses may be cached. Individually minor; collectively they remove several cheap layers.

RECOMMENDATION

Add a `SecureHeaders` middleware to the `Chain`: `nosniff`, `X-Frame-Options: DENY`, `Referrer-Policy: no-referrer`, `Cache-Control: no-store` on authenticated responses, and HSTS once TLS terminates. Serve the SPA with a restrictive CSP.

FINDING-7 Database connections run in plaintext **MEDIUM**

CWE-319 Cleartext Transmission of Sensitive Information

EVIDENCE

`.env` sets `DB_SSLMODE=disable` ; the live connection reports `PLAINTEXT` via `pg_stat_ssl` .

IMPACT

Negligible on localhost. Once the database is on a separate host, credentials and PII (passenger names, passport numbers, contact details) cross the network unencrypted.

RECOMMENDATION

Use `sslmode=verify-full` with a pinned CA for any non-local database; keep `disable` only for loopback dev, and document that production must differ.

FINDING-8 Real database password committed in test source **LOW**

CWE-798 Use of Hard-coded Credentials

EVIDENCE

`config_test.go` contains the actual development DB password as a test literal for the DSN-escaping test.

IMPACT

Low today (dev password, not yet a git repo), but it will enter git history on first push and persist forever, especially damaging if the password is reused. Secrets in fixtures are a common leak vector.

RECOMMENDATION

Use a synthetic password that still exercises escaping (e.g. `p@ss:w/rd?##`). Add a secret-scanning pre-commit hook (gitleaks) before the first commit.

FINDING-9 Credentials stored reversibly client-side and replayed LOW

CWE-522 Insufficiently Protected Credentials · CWE-312 Cleartext Storage

EVIDENCE

`auth.service.ts` stores `btoa(email:password)` in `localStorage` and replays it on every request. `base64` is encoding, not encryption — the plaintext password is trivially recoverable.

IMPACT

Inherent to HTTP Basic auth (the README flags it). Concrete risk: any XSS or compromised SPA dependency reads the reusable password from `localStorage` — worse than a short-lived token — and it travels on every request.

RECOMMENDATION

As the README foresees, move to short-lived tokens with refresh in an `HttpOnly; Secure; SameSite` cookie. Auth logic is already isolated to two backend packages and the Angular service, so the change is contained.

FINDING-10 No server-side output encoding of stored data LOW

CWE-79 Stored XSS sink (currently unreachable)

EVIDENCE

A passenger created with `firstName = <script>alert(document.cookie)</script>` is stored and returned verbatim by the API.

IMPACT

Not currently exploitable — the only consumer is Angular, which output-encodes, and no `innerHTML` sink exists. Latent: a future consumer rendering this as HTML (email, PDF ticket, admin tool, export) would execute it.

RECOMMENDATION

Treat the API as the trust boundary: normalise/validate free-text on input and ensure any non-Angular renderer encodes for its context. Do not rely solely on client escaping.

FINDING-11 Role model implemented but unenforced

INFO

CWE-1220 Insufficient Granularity of Access Control (latent)

EVIDENCE

`RequireAdmin` exists and is unit-tested, and users carry `admin / agent` roles, but `router.go` gates no route with it — every authenticated user reaches every endpoint.

IMPACT

None today (no admin-only functionality). Flagged because the moment one is added it must be remembered to attach `RequireAdmin`, or it will be silently open to all agents — a broken-access-control regression waiting to happen.

RECOMMENDATION

Gate admin functionality explicitly when introduced and add a test asserting an `agent` receives 403, mirroring the existing `TestRequireAdmin`. Prefer a default-deny posture for new routes.

4 Remediation priority

PRIORITY	FINDINGS	RATIONALE
Before any non-local deployment	1, 2, 4, 6, 7	These turn from low-risk to serious the instant the app leaves localhost
Soon	3, 5, 8	Cheap fixes that raise the cost of the attack paths above
As the design evolves	9, 10, 11	Tie to the planned auth change and to new features

A single focused change set closes most high/medium items: a least-privilege DB role (1); `SecureHeaders` + rate-limiting middleware on the existing `Chain` (4, 6); aligned bcrypt cost with hashing moved into Go (2, 3, 5); and `sslmode` flipped for non-local (7).

5 Methodology & limitations

- Testing was against a single-node localhost deployment with seeded data. A production topology behind a reverse proxy / WAF / TLS terminator may already mitigate some network-layer findings (6, 7) — verify against the real deployment.
- Timing was measured over loopback (mean of 30 requests); absolute values differ over a network, but the *relative* discrepancies behind Findings 2 and 5 are structural (different work factors / short-circuit paths) and persist.
- No load, fuzzing, or dependency-CVE (SCA) scanning was performed. Recommend adding `govulncheck` and `npm audit` / OWASP Dependency-Check to CI as a follow-up.
- Scope was the application. Host hardening, PostgreSQL `pg_hba.conf` / listen configuration, and OS patching were out of scope.

Authorised whitebox VAPT of the TravelHub codebase and its localhost instances. Findings are the assessor's own; verify each against your target environment before and after remediation. Print this page (or use `npm run report -style export`) to produce a PDF.