ORDERLY

Security Hardening Report
Part 2: Fixing Issues & Future Development

TABLE OF CONTENTS

The goal of Part 2

Turn an unprotected baseline into a system that is measurably more defensible (and prove it with evidence).

ARCHITECTURE

BEFORE VS AFTER

Component	Technology	Exposed
WAF / Reverse Proxy	Nginx 1.30 + ModSecurity v3 + OWASP CRS 4.25	:8080 / :8443
Identity Provider	Keycloak 26.2	:8180
Frontend	React 19 + Vite	internal
Backend API	Spring Boot 4.0.3 / Java 21	internal
Database	MySQL 8.0	internal

BEFORE: PART 1 BASELINE

3 containers, **all ports exposed to the host**

- Frontend, Backend, MySQL (all reachable)
- No auth · No authz · No rate limit
- No perimeter

Anyone could read, write, or delete any record!

AFTER: PART 2 HARDENED

5 components, **only the WAF is public**

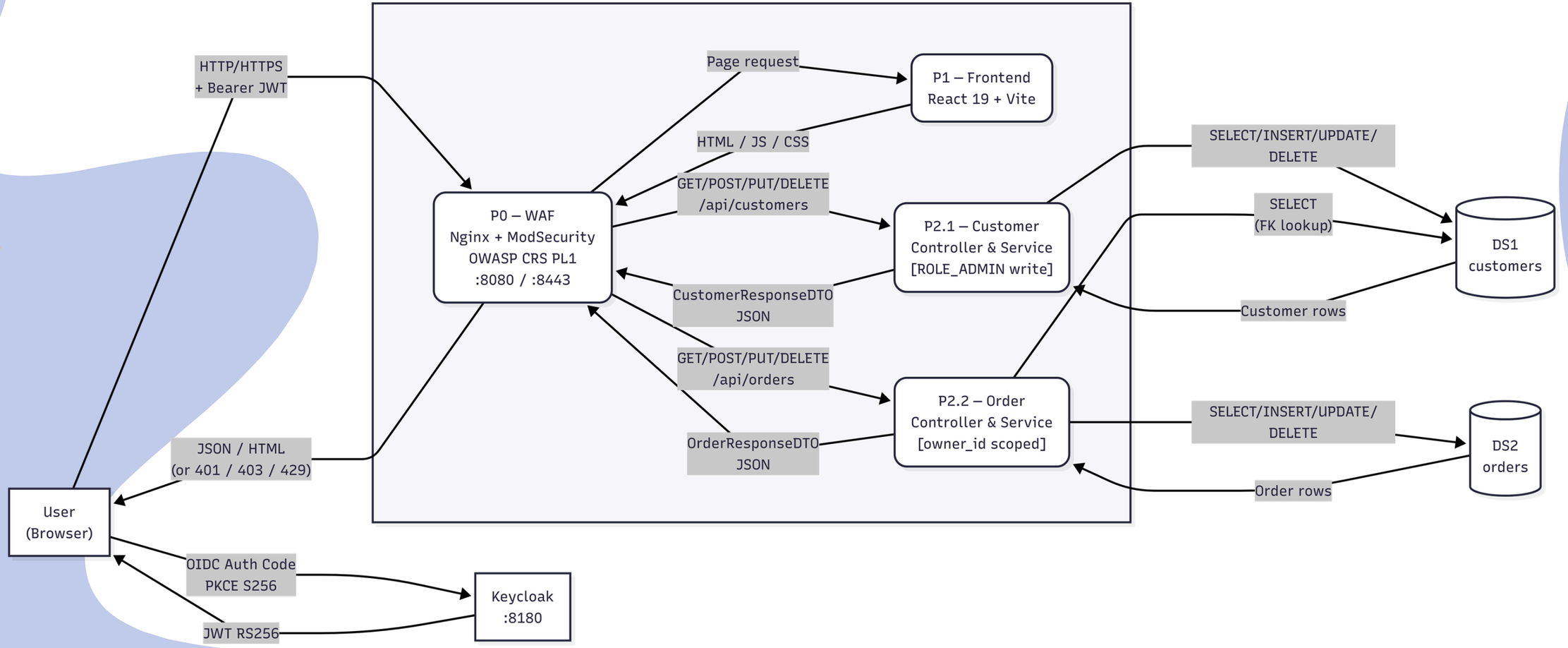
- Nginx + ModSecurity as sole entry point
- Keycloak external Identity Provider
- Backend, Frontend, MySQL internal only

MySQL :3306 no longer reachable from the host

ARCHITECTURE

HARDENED DFD (LEVEL 1)

One public entry point. Every request crosses the Web Application Firewall (WAF) before reaching any internal service.



API HARDENING

OWASP API TOP 10 (2023)

OWASP
TOP 10 API SECURITY RISKS

Rate limiting, two layers:

- Nginx → 60 req/min per IP (429);
- Bucket4j → per JSON Web Token (JWT) sub: 100 GET / 20 POST·PUT / 10 DELETE per min.

Keyed by identity, so rotating IPs does not help an attacker!

The baseline had **no input validation, no output filtering, no data-level access control**. We remediated 6 categories:

API1 ↠ BOLA

- Every order is tied to its creator (owner_id); ownership is checked, not just role

API3 ↠ BOPLA

- Request/response Data Transfer Objects (DTOs); no internal field flows in or out

API4 ↠ Rate limiting

- IP throttling at the edge and per-user limits in the app

API5 ↠ Deny-by-default

- Every route needs an explicit grant

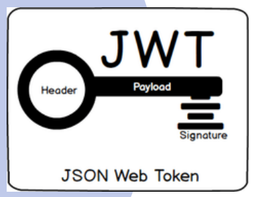
API8 ↠ Config hardening

- Generic errors, security headers, size limits

API9 ↠ OpenAPI

- Machine-readable contract served through the WAF

AUTHENTICATION & IDENTITY

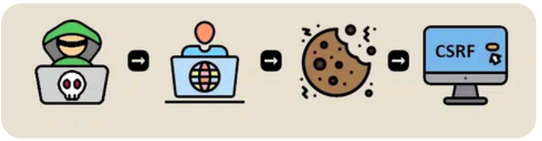


Keycloak 26.2 **JWT RS256** **OIDC Auth Code + PKCE S256** **Refresh rotation**

Decision	Why it matters
External Identity Provider (IdP) (Keycloak)	Token issuance, rotation & revocation without building any of it
Asymmetric signing (RS256)	Backend validates with Keycloak's public key: no shared secret
Proof Key for Code Exchange (PKCE) with S256 challenge	Prevents authorization-code interception
Tokens in memory only	Never in browser storage: resists Cross-Site Scripting (XSS) token theft
Short access token (5 min)	Limits the window if a token is intercepted
Refresh rotation + reuse detection	Each use issues a new token; reuse of an old one kills the whole session

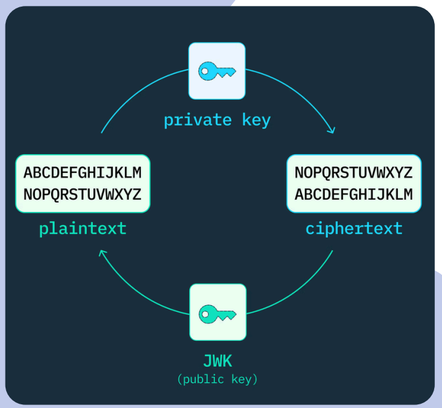
CROSS-SITE REQUEST FORGERY (CSRF)

Not applicable: the API uses Bearer tokens in the Authorization header, not cookies.



AUTHENTICATION

FOUR FLOWS



Stateless backend: every request is validated independently against the Keycloak JSON Web Key Set (JWKS) endpoint.

1 · Login (Auth Code + PKCE)

Redirect to Keycloak with an S256 code challenge → one-time code → exchanged for short-lived tokens → code stripped from the URL.

2 · Proactive Refresh

A 30 s timer refreshes the access token before expiry: issuing a new refresh token and invalidating the old one.

3 · Logout

Redirect to Keycloak's end-session endpoint; all tokens revoked **server-side** before returning to the app.

4 · Refresh Reuse Detection

A stolen, already-rotated refresh token, when replayed, makes Keycloak **terminate the entire session**: forcing re-authentication.

AUTHORIZATION LAYER

DEFENSE IN DEPTH

The baseline had no concept of ownership. Now, three layers enforce access in sequence:

1 · Route

Spring Security maps each HTTP method to a minimum role. No token / wrong role → rejected before any business logic.

2 · Controller

Admin endpoints carry a second @PreAuthorize check: one filter-chain mistake can't open them.

3 · Object

Service layer checks the caller's sub against the stored owner_id before any data operation.

Roles are declared as data: in roles.yml, not hardcoded in business logic. Auditable and changeable without touching application code.

19 unit tests

cover every role × action × resource: including the cases that must return 403.

PERIMETER DEFENSE

WAF

Nginx 1.30.1 | ModSecurity v3 | OWASP CRS 4.25.0 | Blocking mode

Key Configuration:

- ▶ Blocking mode → a match blocks, not just logs
- ▶ Paranoia Level 1 → core coverage, low false positives
- ▶ Inbound anomaly threshold 5 → one CRITICAL match blocks
- ▶ TLS 1.2/1.3 ECDHE → terminates at the perimeter

Covers vs. does not:

a WAF filters inbound requests. It does not rewrite outbound responses: missing headers (CSP, X-Frame-Options) must be set in the Nginx server block. (Confirmed by the DAST: see later.)

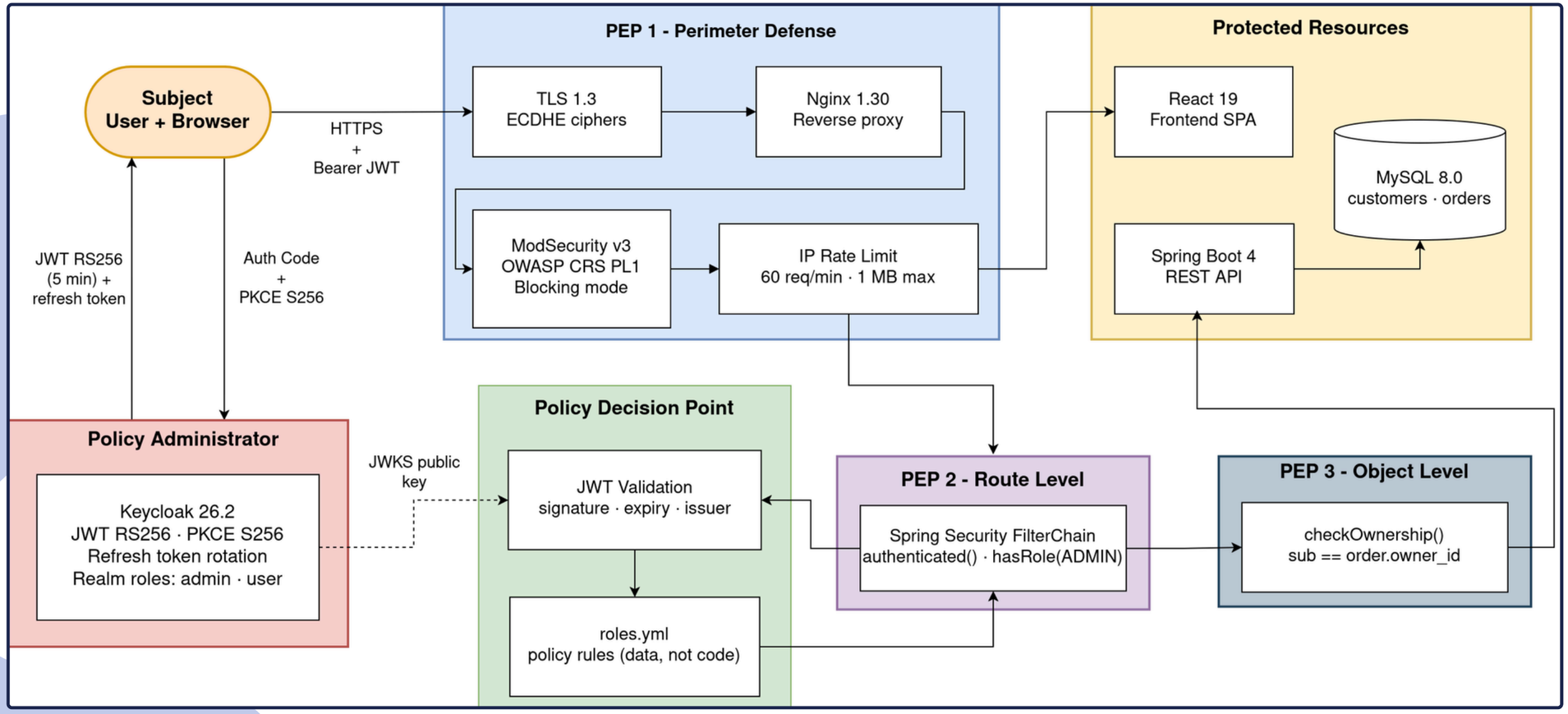
Tuning that mattered:

CRS allows only GET/HEAD/POST/OPTIONS by default. A REST API needs PUT/DELETE explicitly added, otherwise every legitimate update/delete is blocked as a CRITICAL violation. A WAF must be configured for the app it protects.

ZERO TRUST ARCHITECTURE

NIST SP 800-207

Three Policy Enforcement Points in sequence: Perimeter (WAF), Route (Spring Security), Object (ownership). No request reaches the DB without passing all three.



ZERO TRUST

SEVEN TENETS (NIST SP 800-207)

The unmet tenets (2, 4, 5, 7) are architectural gaps: the boundary between a hardened application and a full enterprise deployment. All are documented with a remediation path.

#	Tenet	Status
1	All data sources are resources	Met
2	All communication secured	Partial
3	Per-session access grants	Met
4	Dynamic policy-based access	Partial
5	Monitor & measure posture	Not Met
6	Authentication / authorization strictly enforced	Met
7	Collect maximum telemetry	Not Met

RE-ASSESSMENT

SAST (TRIVY + SONARQUBE)

Why the CVE count rose:

The hardening added security libraries (auth, rate limiting, validation, OpenAPI) absent from the baseline. All 17 backend CVEs are in these new deps and **all have fixed versions**: the fix is a version bump, not a redesign.

The 1 hotspot is a false positive:

CSRF disabled": not applicable with Bearer-token auth. Quality Gate still **PASSED**.

Trivy: dependency CVEs

Target	Before	After
Backend deps	2 HIGH	5 CRIT + 12 HIGH
Frontend deps	0	0
WAF image (Alpine)	–	7 HIGH

SonarQube

Metric	Before	After
Vulnerabilities	0	0
Security Hotspots	0	1 (FP)
Bugs	0	0
Quality Gate	**PASS**	**PASS**

RE-ASSESSMENT

DAST (ZAP, RUN TWICE)

Methodology:

- Part 1 = passive, unauthenticated.
- Part 2 = authenticated active scan, run through the WAF and directly at the app, to isolate each layer's contribution.

Result:

0 exploitable vulnerabilities on the API under authenticated active probing. The 4 remaining true positives are all **missing response headers**: fixable with three Nginx add_header directives + server_tokens off

Finding	P1	WAF	Direct	Veredict
SQL Injection	2H	1H	6H	FP (Vite ?v= param)
Path Traversal	–	0	2H	FP: WAF blocked probe
CSP Not Set	2M	3M	1M	True Positive
Missing Anti-Clickjacking	2M	1M	1M	True Positive
X-Content-Type-Options	1L	syst.	syst.	True Positive
Nginx Version Disclosure	–	2L	0	TP (WAF adds it)
Dotfile probes (.env . . .)	–	54l	0	WAF correctly blocking

ALL 8 ABUSE STORIES

PREVENTED

ID	Story	Control that prevents it
AS-01	Unauthenticated full CRUD	Valid JWT required on all API routes
AS-02	Data harvesting	Ownership filter + per-user rate limiting
AS-03	ID enumeration tampering	Object check → 403 on others' resources
AS-04	Payload overposting	Strict DTO: only declared fields accepted
AS-05	Invalid data injection	Bean validation → 400 field-level detail
AS-06	Error-based info disclosure	Central handler: no stack traces / messages
AS-7	Referential integrity abuse	FK violation caught as 409 Conflict
AS-8	Direct MySQL access	DB port removed from Docker Compose

/ 8 abuse stories classified as Prevented!

RESIDUAL RISKS & FUTURE WORK

All known gaps are documented with a remediation path:

Risk	Category	Path forward
MySQL root user	Elevation	Dedicated least-privilege DB user
No audit trail	Repudiation	Structured logging → ELK
HTTP :8080 open	Info Disclosure	Valid TLS cert for Keycloak; force HTTPS
Backend container as root	Elevation	Non-root user in Dockerfile
Missing response headers	Info Disclosure	add_header directives in Nginx
CRS Paranoia Level 1	WAF bypass	Raise to PL2 after FP tuning
ZT telemetry (tenets 5 & 7)	Monitoring	JSON logs → ELK; OpenTelemetry
No mTLS between containers	Comms (tenet 2)	Service mesh (Istio / Linkerd)

These are the boundary between a hardened application (this project's scope) and a full enterprise deployment.

IN SUMMARY

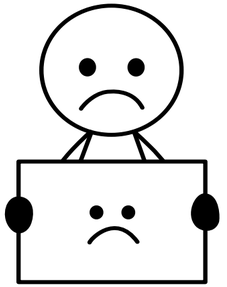
WHAT WE DELIVERED

- [ ] Authentication via Keycloak + JWT RS256
- [ ] Three-layer authorisation (route · controller · object)
- [ ] OWASP API Top 10: 6 categories
- [ ] WAF perimeter in blocking mode
- [ ] Zero Trust framing (NIST SP 800-207)

THE EVIDENCE

- 8/8 abuse stories prevented
- 0 exploitable vulnerabilities (authenticated DAST)
- PASS SonarQube Quality Gate
- 19 authorisation unit tests passing

Measurably more defensible than the Part 1 baseline: every control traceable to a requirement, every residual risk documented.



deti universidade de aveiro
departamento de eletrónica, telecomunicações e informática

THANK YOU!

Security in Software Engineering

18 June 2026

Carolina Reis, nº 131193
Eduardo Lopes, nº 103070