CASE STUDY 1

400+ security findings closed in 45 days

Static-scan and penetration-test remediation in a live Java 8 / Spring Boot 2.7 logistics backend, confirmed by retest scan.

APPLICATION **Logistics rate backend for a global manufacturer**	STACK **Java 8 · Spring Boot 2.7.0**
SCOPE **SAST and penetration-test findings, P1 and P2**	DURATION **45 days, alongside normal releases**

Anonymized. The client and application are not named, and details that could identify them are left out. The fixes are described by class rather than by individual finding.

1 At a glance

400+ P1 high and P2 medium findings closed	**45** days, alongside normal releases	**Retest** scan confirmed the fixes	**7** finding classes fixed at the root

- **Regression tests added** to a codebase that had none, so behavior could be checked during the work and after it.
- **All hard-coded secrets removed** and moved to an enterprise secrets vault, and the exposed keys rotated.

2 Context

The application is a logistics backend for a global manufacturer. It retrieves and updates live shipment rates from several carrier and routing APIs, so that logistics staff can compare and choose rates.

Stack	Java 8, Spring Boot 2.7.0
Integrations	Several third-party carrier rate APIs, each with its own credentials
Sign-in	Single sign-on through the enterprise identity provider
Business role	Rates feed day-to-day shipping decisions, so the service had to keep working while it was being fixed

3 Challenge

The application had been through a static application security test (SAST) and a penetration test. Together the reports listed more than 400 findings rated P1 (high) and P2 (medium), across classes from broken access control to hard-coded secrets. The client needed them closed and verified on retest, while:

- keeping existing behavior and carrier integrations working,
- not stopping releases for a rewrite,
- working in a codebase with no automated tests to catch regressions,
- staying on Java 8 and a Spring Boot line (2.7.x) that is past its open-source support.

4 Approach

`Triage`

1. Merge, confirm, rank

We merged the findings from all reports, grouped them by class, confirmed each against the code, and ranked them by severity and business impact. Likely false positives were documented with the reason, rather than suppressed silently.

`Fix by class`

2. Fix the class, not the line

With 400+ findings, fixing them one at a time would not fit in 45 days. Where a class of issue appeared in many places, we put the fix into one shared mechanism (a filter, a validator, an authorization check) so the whole class closed together and new code inherits the fix.

`Safety net`

3. Add regression tests

The codebase had no automated tests, so we added regression tests around the changed areas to show behavior stayed the same and to prevent future regressions.

Verify

4. Retest

The fixes went back through the scanner, and the retest scan confirmed them.

5 What we fixed, and how

#	FINDING CLASS	WHAT IT MEANS	HOW WE FIXED IT
1	**Broken authorization and IDOR** insecure direct object reference	A signed-in user could reach or change records that belong to someone else by changing an ID in the request.	Authorization checks enforced on the server for every protected endpoint, using Spring Security rules. Every lookup by ID is checked against the caller's identity or role before data is returned or changed. Access is denied by default.
2	**Session and authentication**	The application trusted sign-in state without checking it properly.	After single sign-on and the browser redirect, the access token is checked with the identity provider (OIDC token introspection) before a session is trusted. Identity headers set by the cloud load balancer are validated, so a caller cannot forge them.
3	**CRLF injection** log and header injection	User input containing line breaks could forge log entries or inject HTTP response headers.	Carriage-return and line-feed characters are stripped or encoded before input reaches logs or response headers. Headers are set only through framework APIs.
4	**Cross-site scripting (XSS)**	User-supplied data could be returned to a browser and run as script.	Input validated against an allow-list where the format is known. Output encoded for its context. Correct content types and security headers set on responses.
5	**Directory traversal**	A file name or path from the request could escape the intended folder (for example with `../`).	Paths are resolved and normalized, then checked to stay inside a fixed base directory. File names never come directly from user input.
6	**Vulnerable and outdated components**	Third-party libraries with known published vulnerabilities (CVEs).	Dependencies upgraded to fixed versions compatible with Java 8 and Spring Boot 2.7. Transitive versions pinned through dependency management. Re-scanned after each change.
7	**Hard-coded credentials and API keys**	Passwords and carrier API keys stored in source code or config files.	Secrets removed from code and configuration and moved to an enterprise secrets vault, read at runtime. The exposed keys were rotated, so old copies in history are useless.

6 Outcome

- More than 400 P1 and P2 findings closed in 45 days, alongside normal releases.
- Retest scan confirmed the fixes.
- No secrets remain in the codebase; carrier credentials live in the enterprise vault, and the exposed keys were rotated.
- The application now has regression tests where it had none, so future changes can be checked.

What remains. Java 8 and Spring Boot 2.7 are both past open-source support. Each new library CVE gets harder to fix without moving forward, because fixed versions increasingly need newer Java and Spring. The natural next step is an upgrade to Java 21 and a supported Spring Boot. The new regression tests make that upgrade safer.

7 What a Pashyaa engagement looks like for an application like this

All prices in USD. Fixed scope, fixed price.

STEP	WHAT YOU GET	PRICE
Modernization Assessment	Scan results, dependency and CVE inventory, test-coverage baseline, and a fixed-price plan with clear scope	$900 credited in full if you go ahead with the upgrade
Safety Net	Tests that pin down current behavior on the risky paths (authorization, sign-in, rate retrieval, file handling) so later changes can be proven safe	from $2,500
Upgrade & Migration Module	Move from Java 8 and Spring Boot 2.7 to Java 21 and a supported Spring Boot, including Jakarta namespace and Spring Security changes	$4,000–$15,000 fixed price
Security remediation add-on	Close security findings by class, as in this case study. Scoped and quoted after the assessment	quoted after assessment
Keep-Current Retainer optional	Monthly dependency and CVE updates, re-scans and small fixes, so the application does not fall behind again	$500–$1,500 per app / month

See our sample assessment report for what the first step delivers. AI-assisted where it helps; every change reviewed and tested before release.

Pashyaa Technologies Pvt. Ltd., Pune, India · Java and Spring Boot modernization at a fixed price

pashyaa.com · info@pashyaa.com