SEAL SECURITY

CASE STUDY

3,800+ CVE fixes, signed, sealed and delivered

How a Fortune 500 bank sealed the open source it already runs, in place, with no forced migration, ahead of its enterprise launch.

✓ No upgrades, no migration: same versions, now sealed.

3,828
CVE fixes delivered by the first week of November 2026

100%
of fix requests for the bank's business-critical applications completed

57%
of fixes for critical or high-severity CVEs

16
fixes for vulnerabilities on CISA's KEV catalog, including Spring4Shell and Ghostcat

AT A GLANCE

Environment	Java applications on Maven and Gradle, GitHub Actions with shared pipeline templates, Aqua (Trivy) and Checkmarx One scanning
Challenge	Keep long-lived applications trusted as AI speeds up vulnerability discovery, without upgrade projects, inside a regulated change process
Deployment	One Seal CLI step in the shared pipeline templates, applying fix rules that the bank's central team sets on the Seal platform

The challenge

AI now finds vulnerabilities faster than teams can upgrade their way out of them. This bank's applications use old dependencies, some first released in 2005, and every upgrade means code changes, regression testing and a change-and-release process with at least six weeks of lead time.

EVERY UPGRADE MEANS

Code changes → Regression testing → Change and release → 6+ weeks lead time

THE BANK NEEDED FIXES THAT ARE

- In the versions it already runs
- Recognized by its own scanners
- Delivered through its shared pipelines
- Backed by the evidence its security review requires

The solution

Seal backports each upstream fix into the exact dependency versions the bank already runs: every fix is signed by Seal, sealed in the bank's own scanners and delivered through one pipeline step. The bank adopted it in three stages.

- **Signed** by Seal, traceable to the upstream fix
- **Sealed** in the bank's own scanners
- **Delivered** through one shared pipeline step

1 Proving it on the bank's own applications

The bank ran a technical evaluation on three of its own applications, using its own scanner findings and builds. **All three were completed within five weeks.** Along the way, the teams worked through outbound-proxy restrictions and added support for the bank's older Gradle tooling.

Application 1 ✓ | Application 2 ✓ | Application 3 ✓ — 3 of 3 completed within 5 weeks

The bank's security review then asked whether it could trust a patch it didn't write. Every sealed version starts from the upstream fix, is tested against the bank's exact version, and is reviewed and approved by a Seal security engineer before it is signed, so each one is traceable back to its source. For that review, Seal supplied signing and verification documentation.

HOW EVERY SEALED VERSION IS MADE

Starts from the upstream fix → Tested against the bank's exact version → Reviewed and approved by a Seal security engineer → **Signed by Seal, traceable to its source**

2 Getting the bank's scanners to agree

A sealed version only counts if the bank's scanners agree. Seal integrated with both of the bank's scanners, using a different mechanism for each, so security teams see remediation where they already look.

Aqua and Trivy

NATIVE VULNERABILITY FEED

Original version	CVE reported
Sealed version	Patched CVEs recognized as fixed
New CVE on a sealed version	Still reported

- The bank's Aqua findings set the remediation scope.
- Trivy reads Seal's feed natively and **shows sealed versions as fixed** for the CVEs Seal patched.
- No configuration, suppression lists or manual triage.
- New CVEs on a sealed version are **still reported**, so scans show the risk that remains.

Checkmarx One

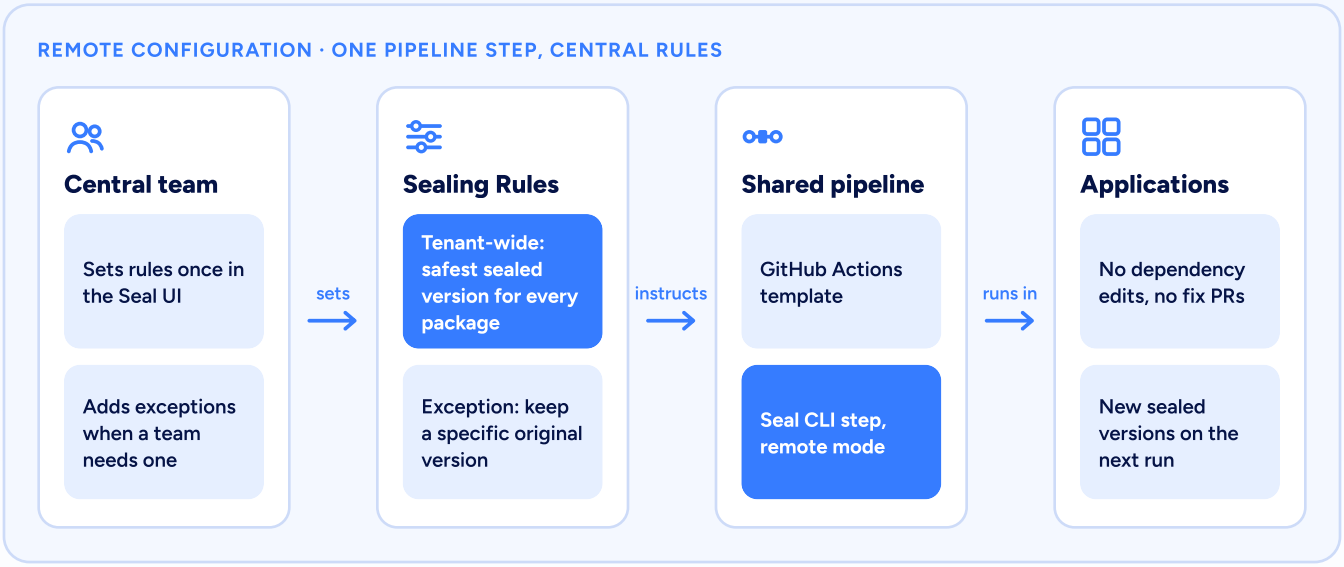
CUSTOM "SEALED" STATE VIA API

To Verify → (Seal CLI + API) → ✓ Sealed

- After each pipeline run, the Seal CLI moves every finding it fixed to a custom **"Sealed" state**.
- Built-in states like "Not Exploitable" don't mean patched. Seal added the new state **within a week**.
- Packages kept at their original version stay open, so the dashboard matches what ships.
- For auditors, "Sealed" is **evidence of remediation**, not risk acceptance.

3 Moving remediation into the shared pipeline

The evaluation ran Seal in fix-everything mode. For the enterprise rollout, the bank chose remote configuration to save developers time and streamline the process. The Seal CLI runs as one step in the shared GitHub Actions templates, in remote mode: it takes its instructions (which packages to replace, and with which sealed versions) from Sealing Rules that the bank sets centrally on the Seal platform, not from a file in each application's repository.



A single tenant-wide rule applies the safest sealed version to every package in every project. When a team needs an exception, such as keeping a specific original version, the bank's central team adds a rule in the Seal UI. Developers don't edit dependency files or open pull requests for fixes. When Seal ships a new sealed version, the next pipeline run picks it up automatically, with no pipeline change.

How the bank's deployment evolved

STEP 1

Manual package delivery

Seal-built packages published to the bank's internal registry, dependency references edited by hand

Used in: Early technical evaluation

Trade-off: Proved the fixes and scanner recognition, but every application needed hand edits

→

STEP 2

Automatic remediation

The Seal CLI in the pipeline fixes everything it can

Used in: Evaluation pipelines

Trade-off: No per-project setup, but no central way to set exceptions

→

STEP 3 CHOSEN

Remote configuration

The Seal CLI applies fix rules set centrally on the Seal platform

Used in: Enterprise rollout

Trade-off: One pipeline step for every team, with central control and exceptions and no changes to application repositories

> *"What makes this work at a bank is that nobody edits an application repository. One pipeline step, one central rule, and every team picks up new sealed versions on its next build. That's how you get to thousands of fixes without a thousand pull requests."*
>
> **Lev Pachmanov**, CTO, Seal Security

Results

6	3,828	2 days	0
business-critical applications went first	CVE fixes delivered by the first week of November	median to seal new requests since then	developer work to apply the fixes

In October, the bank chose six business-critical applications to go first. Instead of asking each team to pick fixes one by one, the bank requested every fix those applications needed at once, so all teams could run their pipelines together. By the first week of November, Seal had completed every request and **delivered 3,828 CVE fixes** across the bank's application portfolio. Since then, the bank's new requests have been **sealed in a median of two days**.

Because every one of those pipelines already ran Seal with central rules, the fixes needed no developer work: once the sealed versions shipped, teams re-ran their pipelines and the central rule applied them.

3,828 CVE fixes delivered — 57% critical or high



Critical **391** (10%) · High **1,796** (47%) · Other severities **1,641** (43%) · Known exploited (CISA KEV) **16**

The **16 known-exploited fixes** cover vulnerabilities attackers have used in the wild, including:

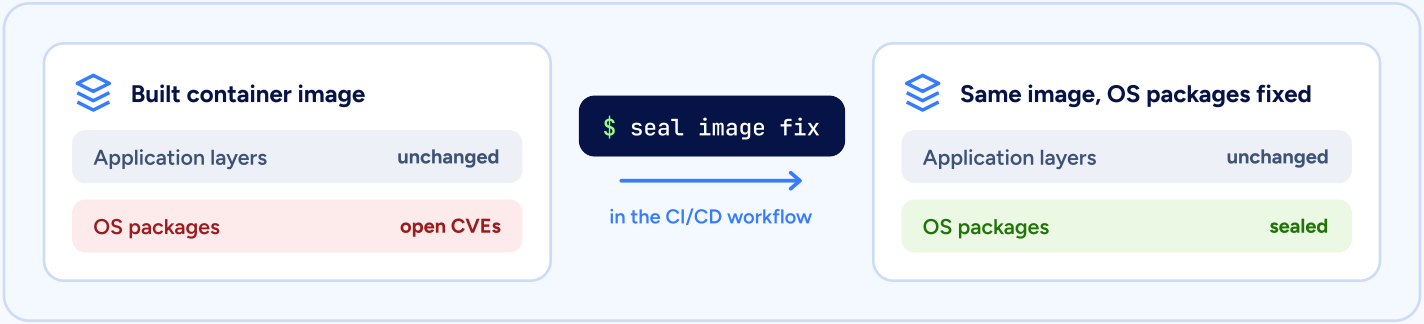
- **Spring4Shell** CVE-2022-22965
- **Ghostcat** CVE-2020-1938
- **HTTP/2 Rapid Reset** CVE-2023-44487
- **Apache Tomcat partial PUT** CVE-2025-24813
- **Apache Struts (2006)** CVE-2006-1547

From evaluation to enterprise launch

April 2026	Evaluation scoped. Mutual NDA, architecture review and vendor-risk assessment begin.
May 2026	Technical evaluation completes all three applications.
June 2026	Customer go-ahead and commercial close.
July to September 2026	Trivy and Checkmarx One integrated with the shared Maven and Gradle pipelines, including a custom state in Checkmarx. The bank moves to remote configuration, with the sealing rules managed centrally on the Seal platform.
October 2026	The bank chooses six business-critical applications to go first and submits their fix requests.
First week of November 2026	**All fix requests for the six applications complete: 3,828 CVE fixes delivered.**

What's next

A second business unit at the bank is now validating Seal on its container images, fixing OS packages inside images it has already built.



> *"We chose seal image fix because our current validation is centered on remediating vulnerabilities in existing container images as part of [our] CI/CD workflow. It lets us work directly with the built image while still exercising the OS package remediation flow."*
>
> **DevOps engineer**, Fortune 500 bank

Why it matters

AI has changed the pace of vulnerability discovery. The bottleneck is no longer finding vulnerabilities; it is fixing them without breaking applications or stalling release processes. This bank showed that a regulated institution can remediate at that pace: thousands of CVE fixes at Seal's regular cadence, signed by Seal, sealed in its own scanners and delivered through its own pipelines, into the versions it already runs.

For security and engineering teams, remediation stops being a queue of upgrade tickets. The open source they already run becomes something they can trust in place, through one pipeline step, set up once and managed centrally, that covers more applications as new vulnerabilities appear.

- **Signed** by Seal
- **Sealed** in your own scanners
- **Delivered** through your pipelines

If it's sealed, you can trust it.