TECHNICAL WHITE PAPER: **September 2026**



Continuous Security with VMware Tanzu Platform in the era of Frontier AI

A technical operating model for moving from vulnerability disclosure to verified remediation

vmware® by Broadcom

Table of contents

Table of contents . . . 2
Executive Summary . . . 3
Why this matters 3
What Tanzu Platform changes. . . . 3
1. Why Patch Latency Is a Systems Problem . . . 4
1.1 The exposure window is shaped by handoffs 4
1.2 Shielding is a bridge, not an end state 4
2. The Tanzu Continuous-Remediation Architecture . . . 5
2.1 Establish a current estate model 5
2.2 Add vendor triage and fix context 6
2.3 Build a compatible change path 7
2.4 Separate application intent from managed runtime layers 7
2.5 Extend the loop into the application 8
2.6 Repave from declared state 9
2.7 Treat credential rotation as a coordinated lane 10
2.8 Close the loop with verification 10
3. Worked Example: CVE-2026-22732 Across a Tanzu Estate . . . 11
From advisory to verified state 11
4. Comparative Operating Risk: Tickets Versus a Platform Control Loop . . . 14
5. Operating the Model and Measuring Control Effectiveness . . . 15

Abstract — Enterprise patching is often managed as a queue of findings, tickets, and change requests. That records activity, but it does not prove that the vulnerable state is gone. Continuous security patching treats remediation as a platform control loop: find the affected estate, choose a supported path, repair and rebuild the right layers, rotate credentials when exposure is plausible, and verify the result. This paper shows how Tanzu Hub, including Vulnerability Insights and Upgrade Planner, connects with Tanzu Operations Manager, BOSH, curated runtime layers, CredHub, and Spring application tooling to close that loop. A worked example follows CVE-2026-22732 across four Tanzu products, then extends the same model into Spring application dependencies.

Executive Summary

Why this matters

A vulnerability response succeeds only when the deployed state changes. Traditional programs often optimize one stage—scanning, ticket creation, or patch deployment—while leaving the expensive correlation work between stages to people. Security teams determine severity. Platform teams map where the component is used. Product owners research supported versions, and change managers assemble prerequisites. Auditors later rebuild evidence from several systems. Every handoff consumes part of the useful remediation window.

The exact disclosure-to-exploit interval varies by vulnerability, exposure, and adversary. The durable design assumption is nevertheless clear: automation lowers the cost of reconnaissance and analysis for both defenders and attackers. A quarterly or sprint-based rhythm is therefore not a reliable control for a critical, remotely reachable issue. Organizations need a path that can answer four questions quickly:

- Where is the vulnerable component?
- Which supported release removes it?
- How can the change be executed without creating avoidable instability?
- What evidence demonstrates that the finding is no longer present in the managed scope?

What Tanzu Platform changes

Tanzu Platform addresses those questions by connecting product-aware inventory with the mechanisms used to change the estate. Tanzu Hub relates vulnerabilities to foundations, product components, versions, support state, and application dependencies. Tanzu's vulnerability analysis adds product and release-line context to raw CVE data. Upgrade Planner uses the installed state to generate a compatible upgrade path. BOSH, buildpacks, stacks, stemcells, and product releases provide repeatable mechanisms for applying that path.

The result is not fully autonomous patching, nor is it a promise of zero downtime. It is a shorter, more observable loop: Detect and Correlate ⟶ Plan ⟶ Repair, Repave, and Rotate ⟶ Verify. Human approval, testing, backups, product guidance, and application ownership still matter. Tanzu Platform reduces the discovery and coordination work around them, then brings the workflow back to a current estate view for proof.

1. Why Patch Latency Is a Systems Problem

1.1 The exposure window is shaped by handoffs

A CVE publication creates two simultaneous tasks. The first is analytical: determine whether the issue is relevant, reachable, exploitable, and material in the organization's context. The second is operational: identify every affected deployment, choose a supported fix, schedule the change, and verify the result. These tasks are tightly coupled. A correct severity assessment without an accurate inventory produces uncertainty; an available patch without a compatible rollout path produces delay.

This is why patch latency should be measured as an end-to-end systems property rather than as the age of an individual ticket. The decisive interval begins when actionable information becomes available and ends when the managed estate has been changed and re-evaluated. Time spent reconciling scanner output with product packaging, reconstructing application relationships, or negotiating ownership is part of the exposure window even when no remediation ticket is formally overdue.

1.2 Shielding is a bridge, not an end state

Runtime shielding, web application firewall rules, network segmentation, and detection analytics can reduce immediate risk. They are especially useful while a fix is unavailable or being validated. Treat each temporary protection as a bridge: give it an owner, a way to confirm that it works, and a date or condition for removing it.

A temporary rule does not remove vulnerable bytecode, an outdated image layer, unknown host state, or credentials that may already have been copied. The job is finished only when the vulnerable component has moved to a supported version and the platform can confirm the new state.

A defensible remediation decision answers four questions from the same system model:

- **Where is it?** Identify the connected foundation, product component, installed version, related applications, environment, and accountable owner.
- **What fixes it?** Distinguish investigation status from fix availability and map the finding to a supported target release—not merely the numerically latest version.
- **How will it change safely?** Sequence prerequisites, backups, canaries, concurrency, maintenance windows, health gates, and recovery procedures before execution.
- **What proves closure?** Re-query the authoritative estate view after the change, retain execution evidence, and keep failed, deferred, or excepted items visible.

2. The Tanzu Continuous-Remediation Architecture

2.1 Establish a current estate model

The first architectural requirement is a current model of the objects the platform team can actually change. Tanzu Hub connects vulnerability information to foundations, product components, installed versions, support status, and application relationships. This attribution changes the first question in a critical response from "Do we think this exists somewhere?" to "Which managed objects currently report the affected component?"

In the example environment, the connected view covered two foundations and seven component types. The scale is modest, but the operating pattern is the point: the query starts from continuously collected platform inventory rather than a spreadsheet assembled after disclosure. When a buildpack, stemcell, service tile, or management component is implicated, the operator can move from the platform object to the workloads and owners that depend on it.

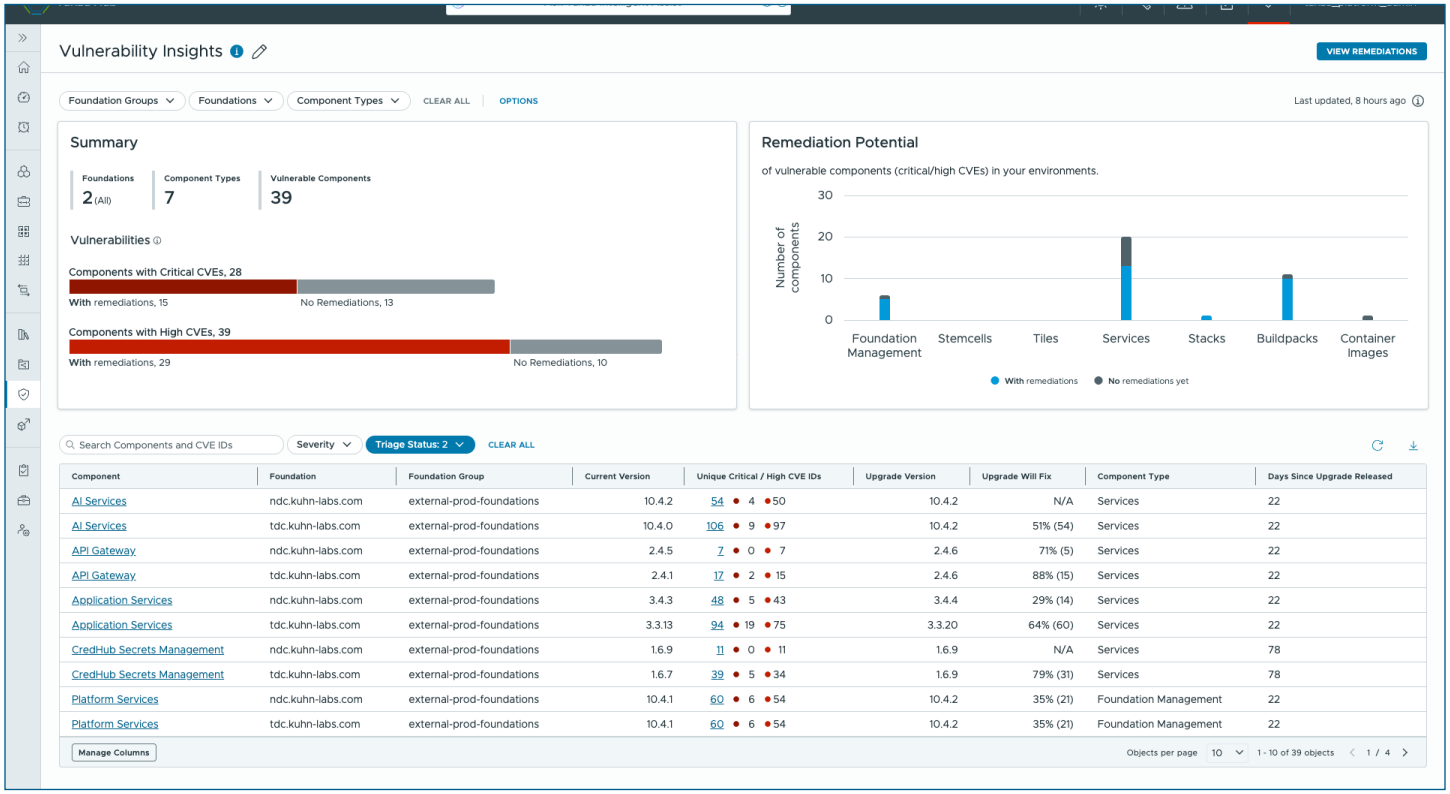


Figure 1. Hub's connected estate view: two foundations and seven component types in the example environment. Counts change as inventory refreshes.

Coverage gives the result its meaning. Connect every in-scope foundation and record the displayed Last updated time with the query. Tanzu Hub refreshes vulnerability data every 24 hours; internet-restricted environments require manual vulnerability and SBOM uploads. A later "no result" is meaningful only when coverage and ingestion are current.

2.2 Add vendor triage and fix context

Raw CVE feeds identify possible component-version matches. Product response requires more context: whether the component is actually affected, whether the vulnerable code is reachable, which release lines are supported, whether a corrective build exists, and what product package carries it. Broadcom's Tanzu vulnerability process tracks vulnerabilities across products, components, and release lines using component software bills of materials and CycloneDX analysis states.

Statuses such as in triage, resolved, exploitable, false positive, and not affected describe the Broadcom TVS and CycloneDX analysis process. Operators see a related but different vocabulary in Vulnerability Insights. The Triage column includes Affected, Not Affected, In Triage, and False Positive; patch status includes Fix available, Fix on next release, Code not present, Protected at runtime, and Can not fix. Broadly, Hub's Affected state maps to CycloneDX exploitable, while Fix available maps to CycloneDX resolved. When a corrected product build carries an updated dependency, its new SBOM can show that the vulnerable version has been replaced.

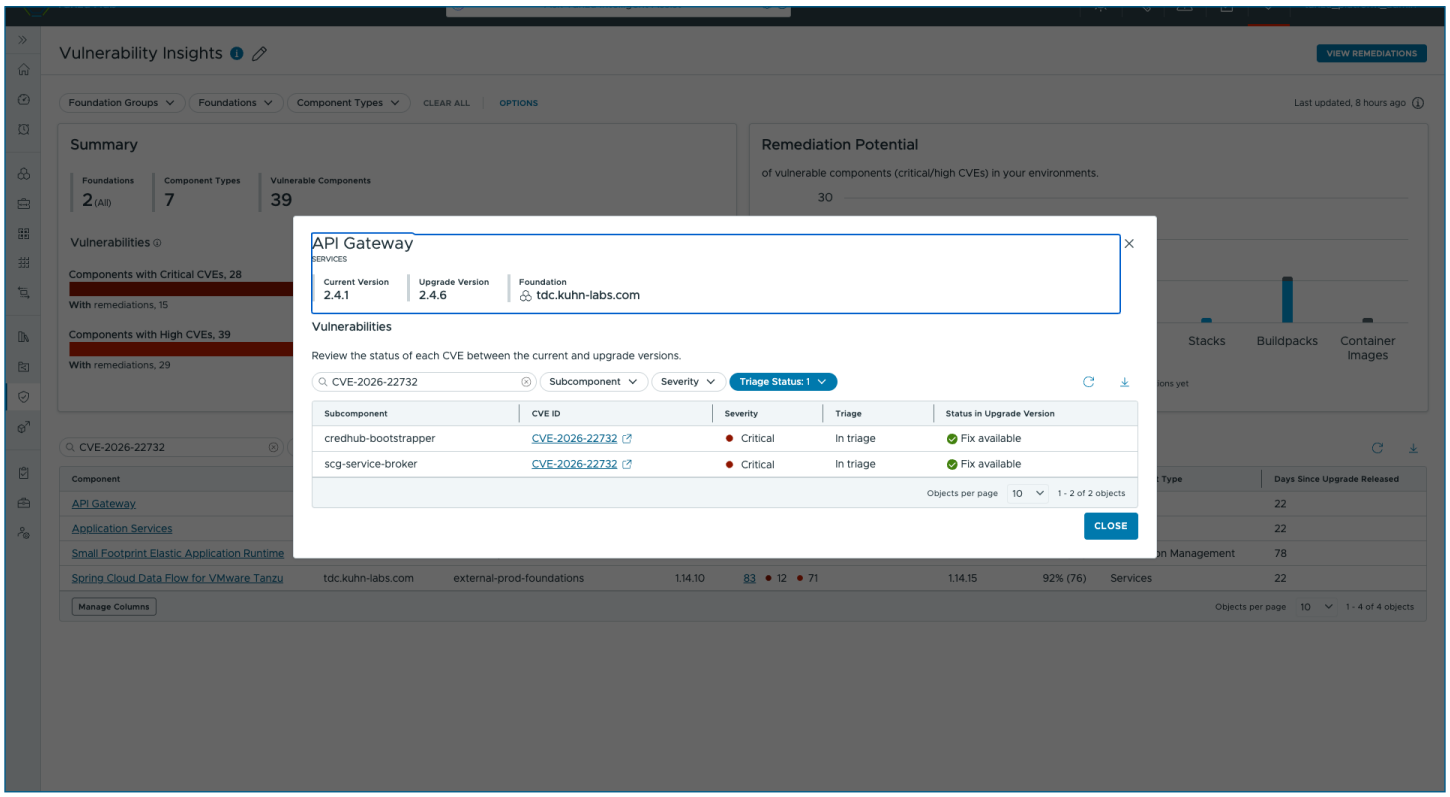


Figure 2. Product-aware drill-down separates triage state from fix availability for CVE-2026-22732.

The distinction matters because one open-source dependency may be packaged into several Tanzu products. Each product can have a different installed version, supported branch, target release, maintenance procedure, and application impact. Treating the CVE as a single generic record hides the very information needed to change the estate safely.

2.3 Build a compatible change path

Once the affected components and fixed releases are known, the practical question is: what can we change first without breaking what comes next? Foundation Core, tiles, services, stacks, and buildpacks have compatibility and support constraints. Upgrade Planner starts with the versions that are actually installed and builds a phased route through prerequisites, compatible product versions, CVEs resolved, and end-of-general-support dates.

The plan gives the change review a concrete starting point; it does not bypass approval. Platform and application owners still confirm product guidance, backups, capacity, maintenance windows, and recovery steps. If the estate or target changes, operators can regenerate the plan and export the updated path into the normal change-control process.

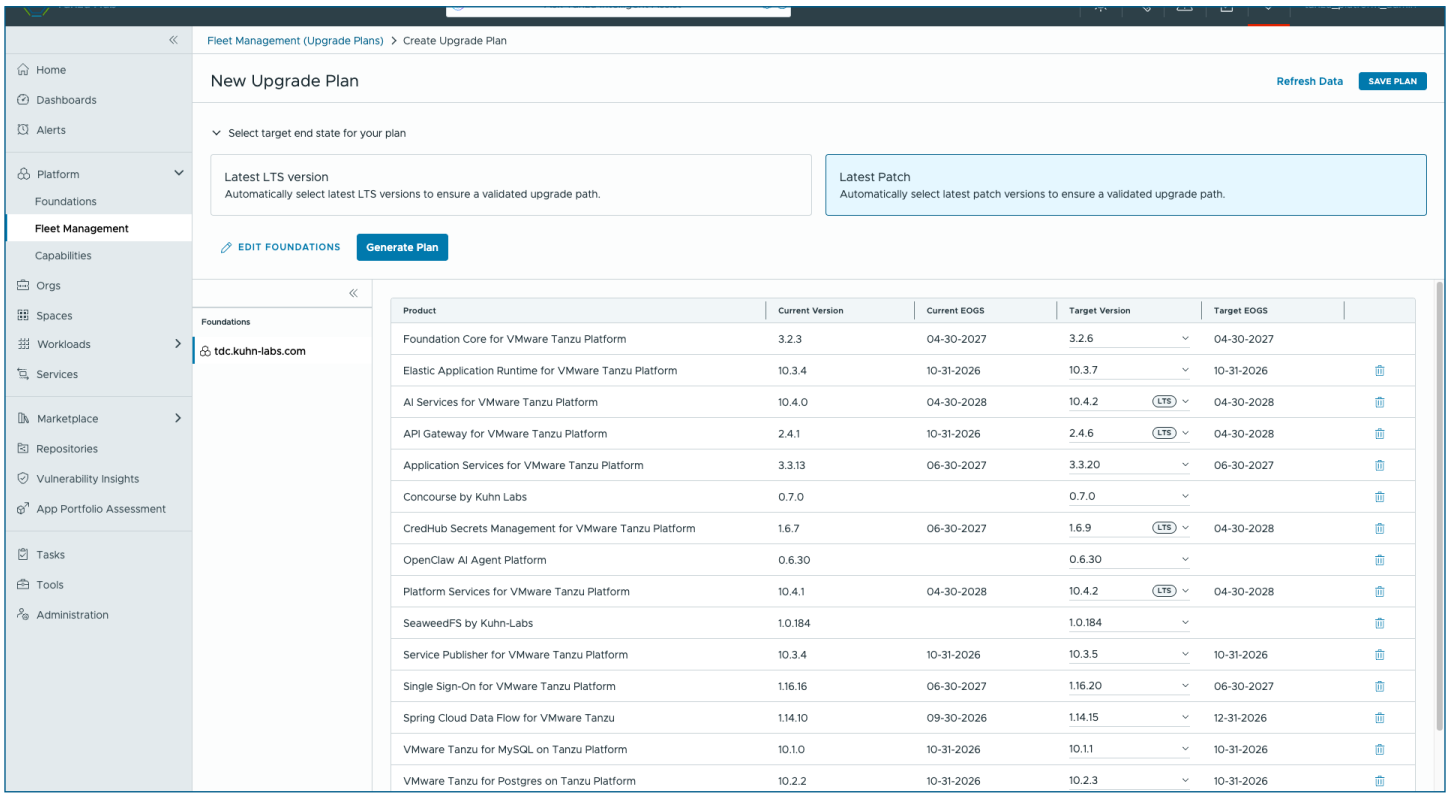


Figure 3. Upgrade Planner starts from installed products and proposes supported targets with compatibility and lifecycle context.

2.4 Separate application intent from managed runtime layers

Continuous remediation becomes practical when common dependency layers are owned and updated as platform products rather than rebuilt independently by every application team. Buildpacks, stacks, stemcells, and product releases separate much of the operating system, language runtime, and packaging lifecycle from application source. A platform team can therefore update many shared layers through a standardized path and identify which applications must be restaged to consume the new bits.

This separation creates a useful ownership boundary. A platform release can repair a platform-managed dependency, and a restage can move an application onto a corrected buildpack or stack. A vulnerability inside the application's declared dependencies usually needs a source change and compatibility testing. Tanzu Platform makes that boundary visible so platform and application teams can act from the same evidence.

2.5 Extend the loop into the application

The shared model should not stop at the platform boundary. Tanzu Hub's application-first view brings operational health together with buildpack CVEs, runtime stack status, and Spring library compliance. Platform operators and developers see the same evidence, but each can act at the right layer. A corrected buildpack or stack may require only a restage; a vulnerable declared dependency usually requires a source change.

From application evidence to a verified change

For Spring applications, the lifecycle is easy to describe: Observe ⟶ Target ⟶ Change ⟶ Verify. Hub inventories the libraries that are actually running, their support status, vulnerabilities, target versions, and estimated upgrade effort. Spring Application Advisor can then analyze the repository and produce incremental upgrade guidance or pull requests, including dependency and Java API changes. After review, testing, and deployment, Hub provides the running-state evidence for verification.

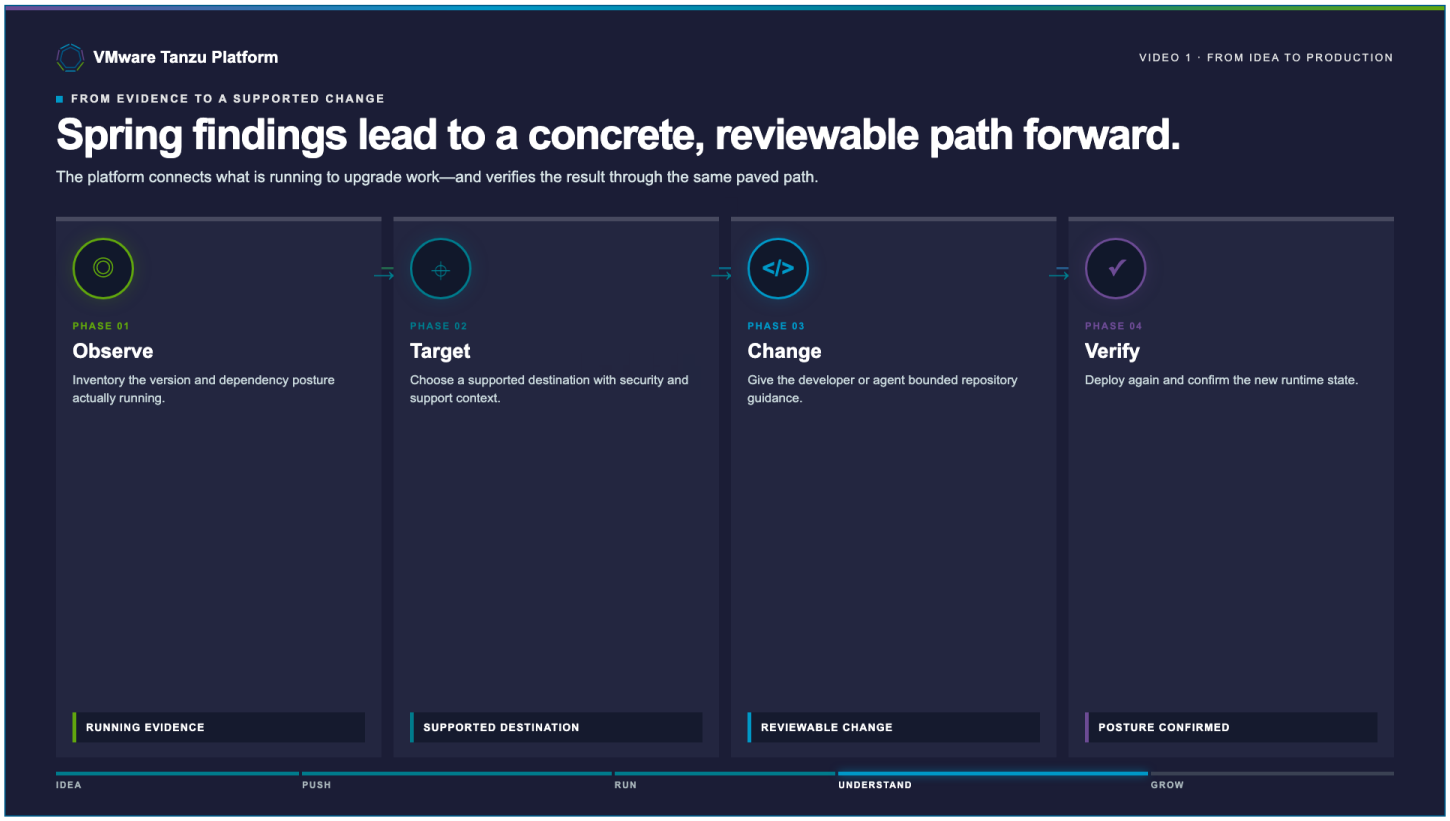


Figure 4. Application remediation follows the same loop: observe the running dependency, choose a supported target, make a reviewable change, and verify the deployed state.

That division of work is productive, not a gap. The platform team keeps shared runtime layers current and provides a repeatable delivery path. Application teams review and test code-level changes. Both return to the same observable deployment, so the remediation record connects repository work to production state.

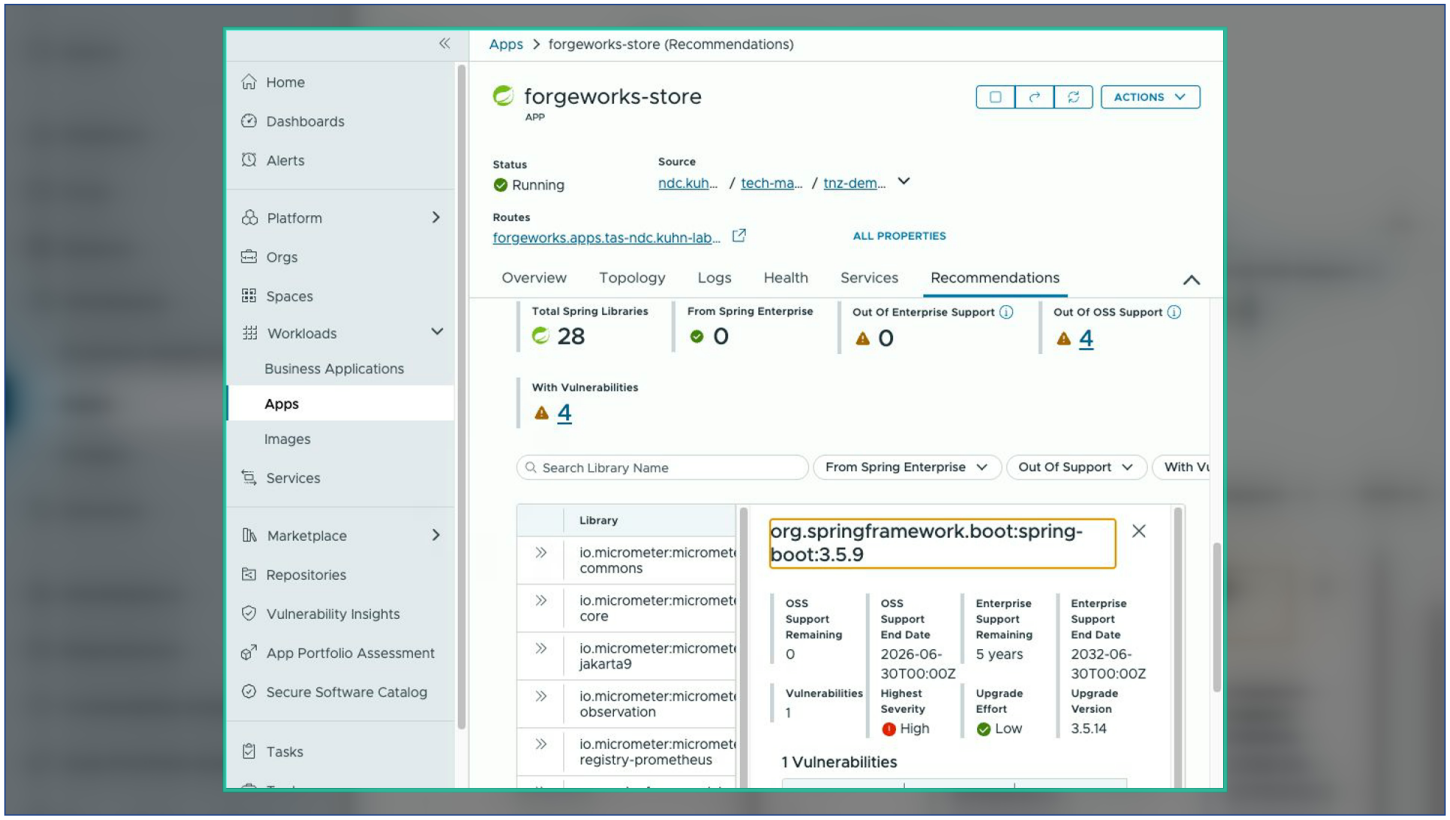


Figure 5. Real Hub evidence for a Spring application: 28 libraries inventoried, support status shown, and a high-severity finding mapped to a low-effort upgrade target.

2.6 Repave from declared state

Repairing a package closes the known vulnerability, but it does not prove that a long-lived host has no unauthorized changes or persistence. BOSH adds a declarative infrastructure layer. Deployment manifests describe releases, stemcells, instance groups, networks, and update behavior; reconciliation recreates managed instances from that declared state instead of preserving local mutations as part of the recovery path.

Canaries, max-in-flight controls, health-check windows, and product-specific procedures make repaving a controlled rollout rather than a blind rebuild. Capacity and observability still matter: a failed canary must stop the sequence, and the replacement strategy must preserve service objectives. The security value is structural—managed instances are replaced from known inputs, reducing the chance that unknown state survives indefinitely.

2.7 Treat credential rotation as a coordinated lane

A patch cannot invalidate a credential that was copied before remediation. When the vulnerability's exploit path or incident evidence makes credential exposure plausible, rotation should run as a parallel workstream. CredHub and BOSH provide mechanisms for managed credential generation and redeployment, but credential types have different consumers and different safe rotation patterns.

CredHub and BOSH automate much of the work by generating managed values, distributing them through deployment configuration, and redeploying consumers. Some credential types still need an ordered transition—for example, a period of overlapping certificate validity or a coordinated database-password change. Tanzu Platform makes that work repeatable and observable. The rotation record should show what changed, which consumers moved, when the old material was revoked, and which authentication checks passed.

2.8 Close the loop with verification

The final architectural requirement is to return to the same estate model used for detection. Hub should be queried again after the change so that installed versions, vulnerability state, and remaining exceptions can be evaluated in context. The evidence package should join that rescan with the generated plan, approved change, execution results, component versions, failed or deferred items, exception owners, and connector freshness.

Verification is scoped, not absolute. A clean result in Hub demonstrates what the connected foundations reported at that time. It does not prove that an unmanaged environment, an application-owned dependency, or a future data refresh cannot contain the issue. Stating that boundary makes the evidence stronger because reviewers can understand exactly what was tested.

3. Worked Example: CVE-2026-22732 Across a Tanzu Estate

From advisory to verified state

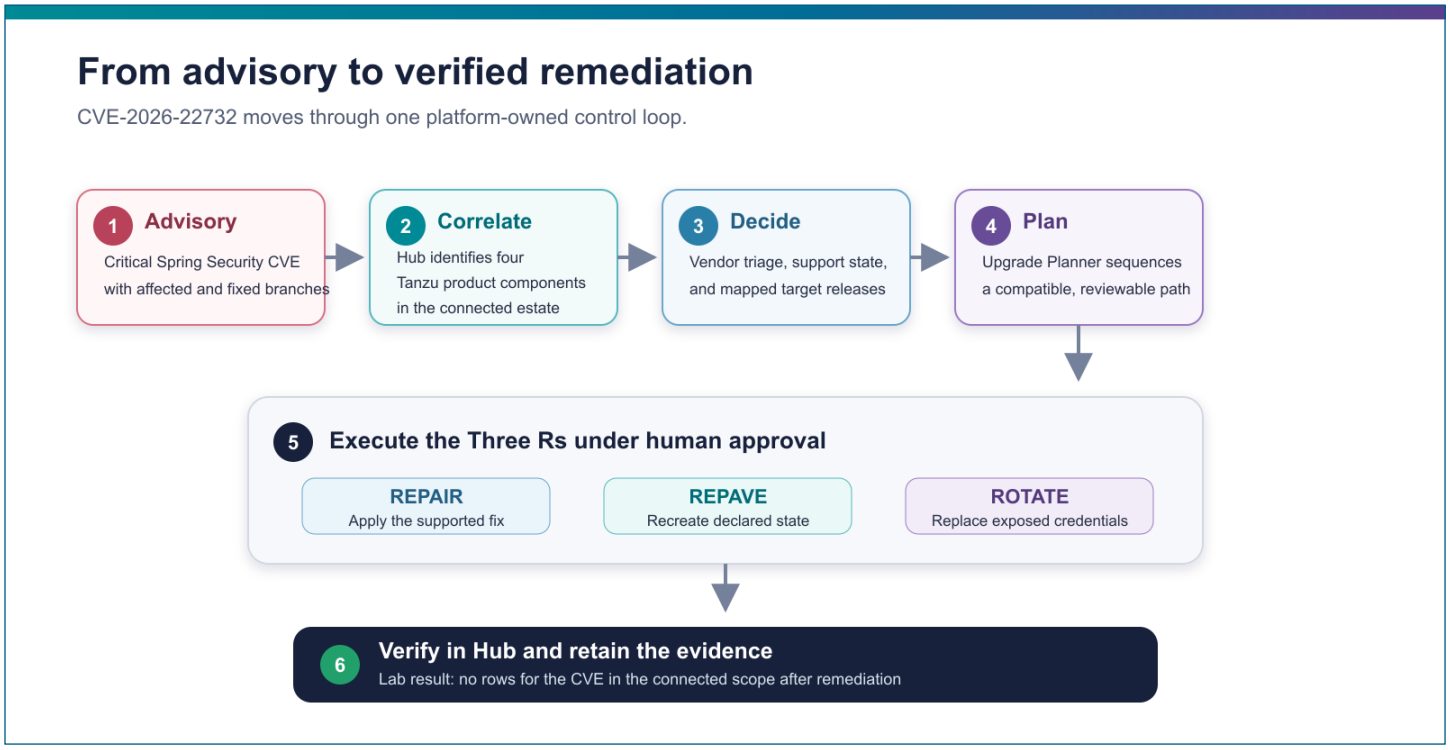


Figure 6. CVE-2026-22732 remediation journey across the connected Tanzu estate.

Step 1 — Detect and correlate

The Spring Security team published CVE-2026-22732 on Mar 19, 2026 and rated it Critical. Under specific conditions in affected servlet applications, Spring Security might not write configured HTTP response headers, potentially exposing sensitive data through caching mechanisms. The advisory identified affected branches from Spring Security 5.7 through 7.0 and listed fixed releases for each supported line. It also documented a workaround, while warning that the workaround changes application behavior. Hub turned the component advisory into an estate query. One search returned four Tanzu product components: API Gateway 2.4.1 ⟶ 2.4.6; Application Services 3.3.13 ⟶ 3.3.20; Spring Cloud Data Flow ⟶ 1.14.10 1.14.15; and Small Footprint Elastic Application Runtime 10.3.4 ⟶ 10.3.7. The useful result was not simply “four rows.” It was four owned platform objects, each tied to an installed version and a corrective destination.

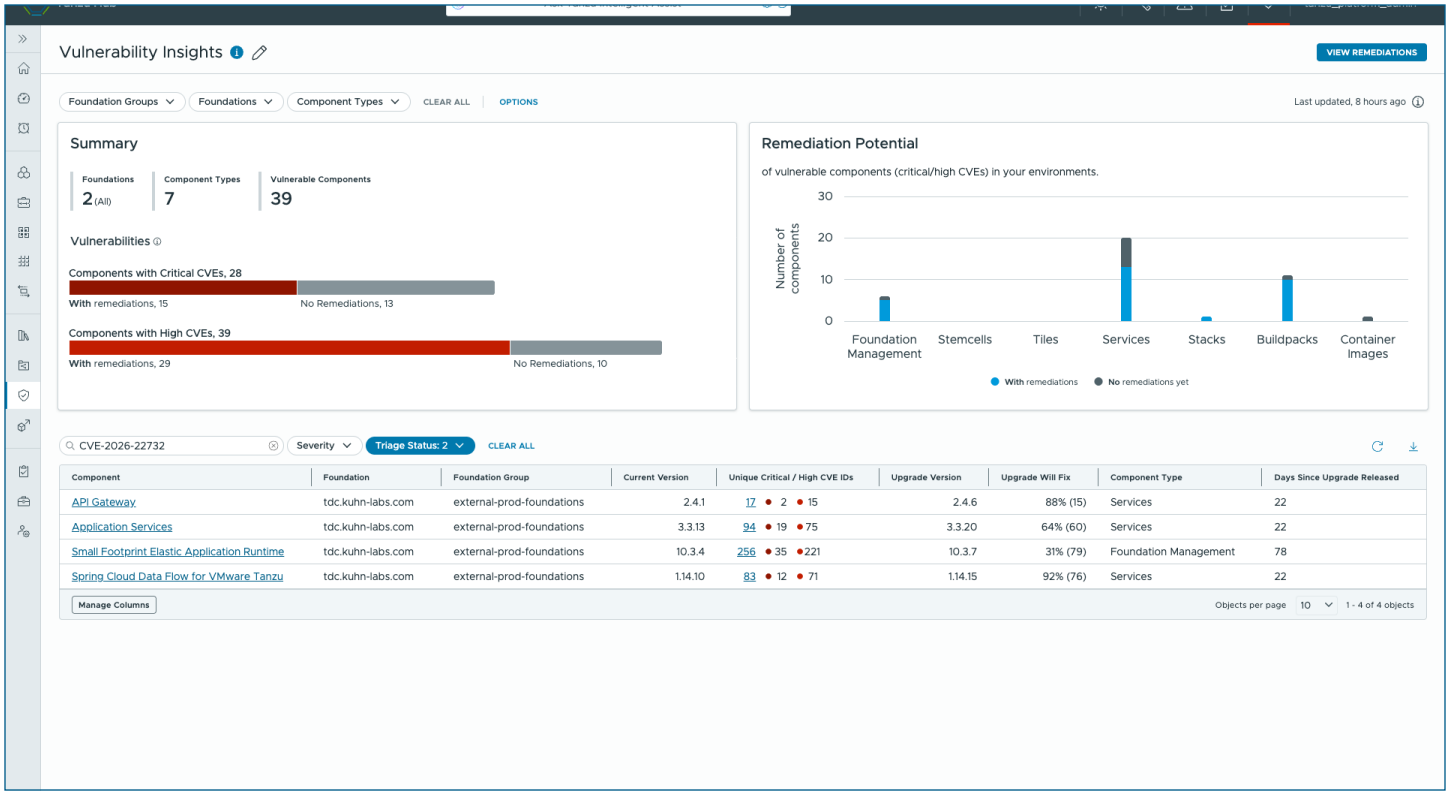


Figure 7. One CVE fans out across four installed Tanzu product components, each with its own corrective destination.

Step 2 — Plan the supported path

The example view contained 38 vulnerable components overall, 29 with a mapped fix. That snapshot shows why response planning cannot assume that every finding is immediately actionable. Some issues remain in triage, some need a product release, some are not affected in the deployed context, and some require a time-bound exception while the vendor or application team completes the work.

For CVE-2026-22732, the four affected products did not share one target version or one maintenance procedure. Upgrade Planner supplied compatibility and lifecycle context for the selected foundation and generated a phased path. Operators could compare a latest-patch goal with a long-term-support goal, regenerate the plan against refreshed foundation data, and export the result for change review. Human approval remained the gate between a technically valid plan and production execution.

Step 3 — Execute Repair, Repave, and Rotate

Repair applies the supported corrective release at the layer that owns the vulnerable dependency. Depending on the component, that can mean a product or tile upgrade, a buildpack or stack update followed by application restaging, or an application-level dependency change. The success criterion is not that the command completed; it is that the affected component now resolves to the fixed version and its dependent workloads are healthy.

Repave recreates managed infrastructure from the declared release and stemcell state. The rollout should use the product's canary, concurrency, health, backup, and recovery controls. Rotate addresses the separate question of credentials that may have been exposed during the vulnerability window. It should be triggered by exploitability and incident context, then executed according to the procedure for each credential type. The three lanes are coordinated because they remove different forms of attacker advantage: Repair removes the known flaw, Repave removes unknown host state, and Rotate removes the value of captured secrets.

Step 4 — Verify and retain evidence

After the updates, the same CVE search returned "No data available." For the connected foundations and that data refresh, the four previously returned platform components no longer appeared. That is useful closure evidence for a defined scope—not a claim that every unmanaged environment or application dependency everywhere is permanently clean.

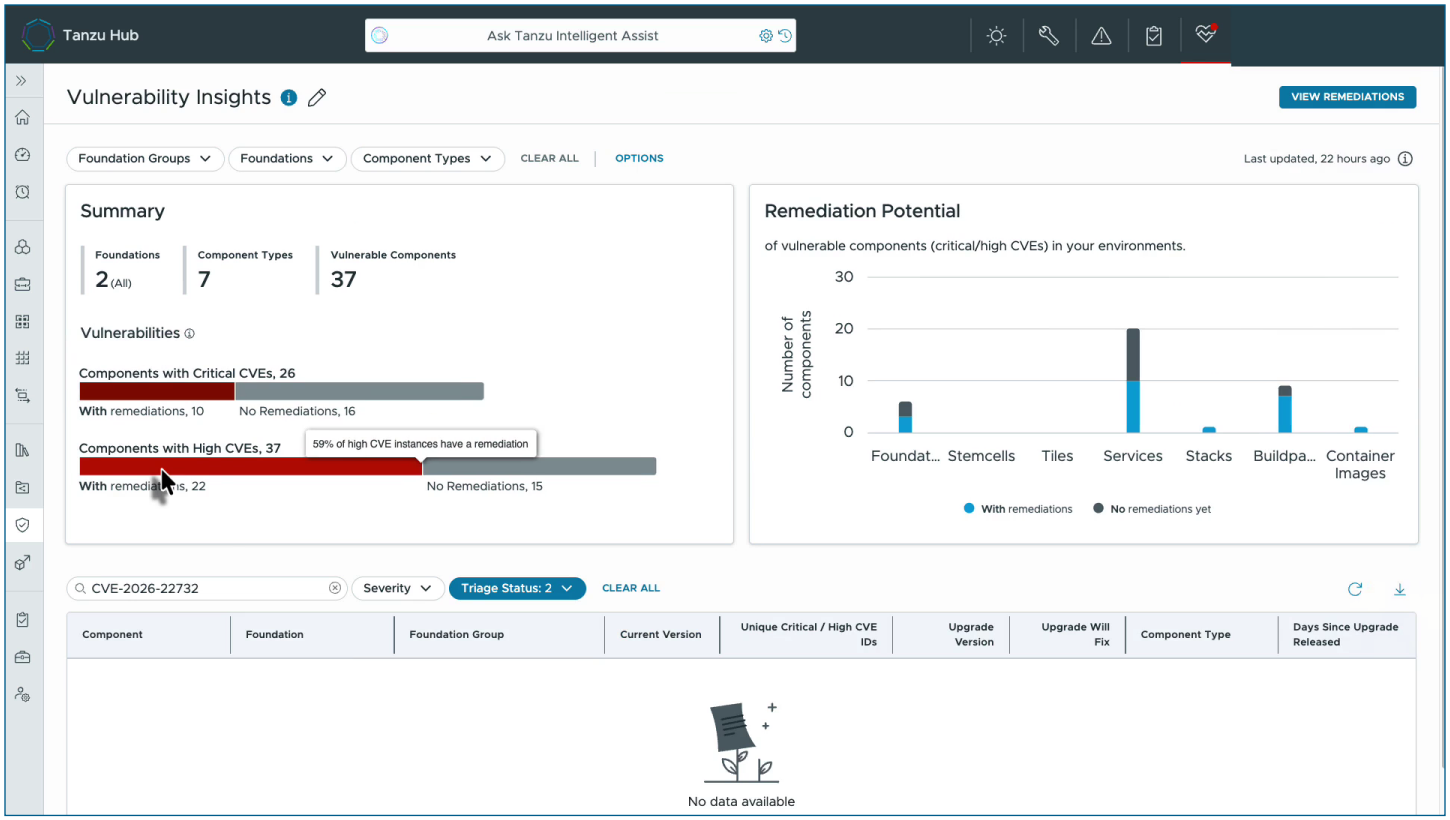


Figure 8. Post-change verification: the same CVE query returns no rows for the connected scope and current data refresh.

A production evidence package should record the pre-change finding, mapped targets, generated plan, approvals, completed product versions, rollout health, rotation results where applicable, the post-change Hub query, and any failures or exceptions. This turns the demonstration into a repeatable control. The process ends with evidence about system state, and the next data refresh begins the loop again.

4. Comparative Operating Risk: Tickets Versus a Platform Control Loop

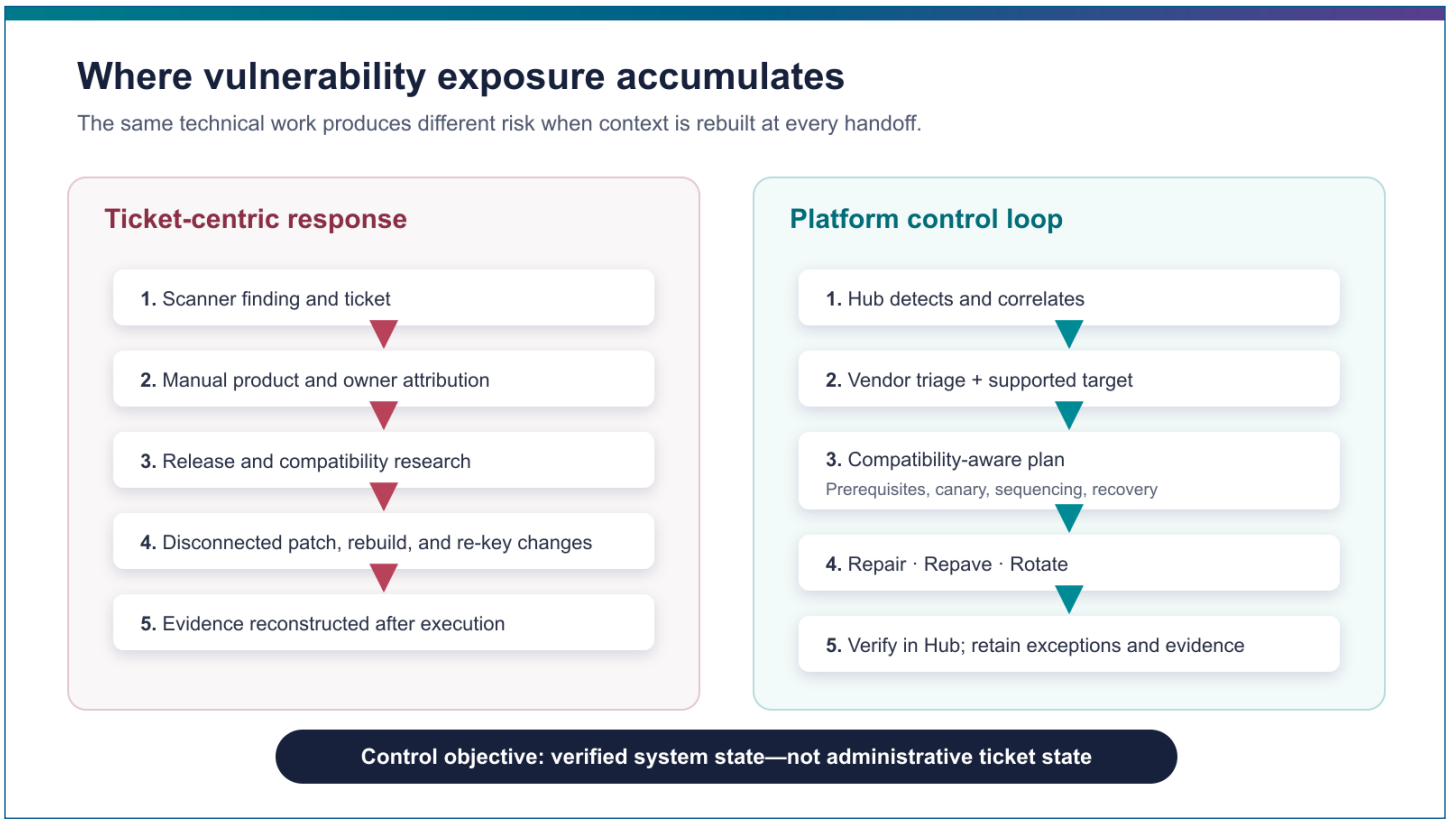


Figure 9. Ticket-centric response versus a platform control loop.

In a ticket-centric baseline, the scanner finding and the production change live in different systems. Security creates a record. Platform engineers reconstruct which product package contains the dependency. Application teams decide whether code changes are required, while release engineering researches supported targets. Change managers assemble the prerequisites, and separate tools execute infrastructure, runtime, and credential work. The process can be diligent and still move slowly because each team rebuilds context at its boundary.

The most dangerous ambiguity appears at closure. A ticket may be closed because a change was scheduled, a compensating control was installed, or one affected component was upgraded. Without a return to the estate model, the organization cannot easily distinguish administrative completion from verified remediation.

In the Tanzu Platform control loop, the same stages remain, but the state moves with the workflow. Hub provides connected inventory and product attribution. Vendor triage and SBOM context identify affected and fixed release lines. Upgrade Planner produces a compatible path from the installed state. BOSH and curated platform dependencies supply repeatable Repair and Repave mechanisms, while CredHub-supported procedures make Rotate a visible parallel lane. The workflow returns to Hub for post-change verification.

This design reduces ambiguous handoffs; it does not eliminate operational risk. A plan can still be wrong for a local integration, a canary can fail, capacity can be insufficient, or an unmanaged environment can remain outside the view. The architectural improvement is that those conditions become explicit inputs, gates, or exceptions in one observable loop rather than hidden reconstruction work during a crisis.

5. Operating the Model and Measuring Control Effectiveness

Continuous security patching requires operating lanes that match the situation. Routine remediation follows the normal platform release cadence. Urgent remediation compresses planning and approval around a high-priority issue with an available fix. An exception lane records the reason a supported fix cannot yet be applied, the compensating controls, owner, review date, and exit criteria. When exploitation or compromise is suspected, the work becomes incident response; containment, forensics, eradication, and credential rotation may take precedence over the standard maintenance path.

Responsibility should be explicit. Security owns threat context and risk interpretation. Platform engineering owns connected-foundation coverage, product lifecycle, rollout mechanisms, and verification data. Application teams own source-level dependencies and compatibility testing. Change authorities govern production risk, and incident responders govern containment and evidence when compromise is plausible.

The most useful measures describe transitions in the control loop: percentage of foundations connected to Hub; inventory and vulnerability-data freshness; time from advisory to affected-estate determination; percentage of findings with a mapped fix; time from fix availability to approved plan; time from approval to completed rollout; verification latency; failed-change rate; exception age; and the proportion of routine fixes completed through the standard platform path. These measures reveal where the organization is consuming the exposure window. Ticket volume alone does not.

Conclusion — The central security advantage of a platform is not that it makes every patch automatic. It is that the facts, ownership boundaries, and change mechanisms are ready before the next critical advisory arrives. Tanzu Platform connects estate inventory, product-aware vulnerability context, compatibility planning, declarative infrastructure, curated runtime layers, credential management, and Spring application intelligence. CVE-2026-22732 shows the loop working across platform products. The Spring example shows it continuing into application dependencies: observe what is running, choose a supported target, make a reviewable change, and verify the deployed state.

The remaining disciplines still matter: threat intelligence, exploitability analysis, testing, backups, recovery, application changes, exception governance, and incident response. A mature program uses Tanzu Platform to make those disciplines operate against the same current system model. That is the shift from patching the ticket to patching—and proving—the estate.

References

Spring Security Advisory: CVE-2026-22732 · Tanzu: Smarter Patching at Scale · Tanzu: Faster Is Still Safer — The Three Rs · Tanzu: Preparing for AI-Driven Exploits

Broadcom: Tanzu External Vulnerability Response and Remediation Policy · VMware Tanzu Upgrade Planner Documentation · BOSH: Deployment Configuration and Credential Rotation

Application security lifecycle: Tanzu Platform 10.4 · Tanzu Hub unified interface · Spring Application Advisor · Spring Application Advisor 1.4

vmware® by Broadcom

Copyright © 2026 Broadcom. All rights reserved.
The term "Broadcom" refers to Broadcom Inc. and/or its subsidiaries. For more information, go to www.broadcom.com. All trademarks, trade names, service marks, and logos referenced herein belong to their respective companies. Broadcom reserves the right to make changes without further notice to any products or data herein to improve reliability, function, or design. Information furnished by Broadcom is believed to be accurate and reliable. However, Broadcom does not assume any liability arising out of the application or use of this information, nor the application or use of any product or circuit described herein, neither does it convey any license under its patent rights nor the rights of others.
Item No: Patch the Estate 9/26