Centralized Scan Factory for Staic Code Scanning

Rohit Chouhan
Application Security Engineer
PepsiCo, TX, US
Email:rchouhan@cs.stonybrook.edu

Sera Kim
Application Security Engineer
PepsiCo, TX, US
Email:serakim1105@gmail.com

Chad Haley
Application Security Engineer
PepsiCo, TX, US
Email:chad.haley443@gmail.com

Abstract—

1. Introduction

Application Security (AppSec) in the modern software landscape spans a broad range of domains—including source code analysis, API and edge security, container security, AI security, and web application firewalls (WAFs). Today's applications are complex, distributed systems composed of numerous interconnected components and services. Securing each of these layers is increasingly challenging, making AppSec a foundational pillar of enterprise security strategy and a critical component of Attack Surface Management (ASM).

Among the core responsibilities of an AppSec program is source code analysis: the process of examining application code to detect potential security vulnerabilities early in the development lifecycle. Since source code forms the basis of all software, identifying and remediating weaknesses at this stage is one of the most effective ways to reduce downstream risk. Traditionally, organizations rely on Commercial Off-The-Shelf (COTS) scanners to perform Static Application Security Testing (SAST), uncovering flaws such as SQL injection, cross-site scripting (XSS), insecure deserialization, and other high-impact vulnerabilities. These scanners also provide Software Composition Analysis (SCA) to detect vulnerabilities and license issues in third-party dependencies, and increasingly include capabilities to identify hardcoded secrets such as API keys, client credentials, and passwords.

While the market offers a wide variety of SAST, SCA, and secrets-scanning tools—each with integration options for source code managers, CI/CD systems, and findings platforms—large organizations face significant challenges in operationalizing them. In environments with diverse development teams, heterogeneous tooling, and thousands of repositories, ensuring that scanners consistently target the correct assets and maintain adequate coverage is non-trivial. Continuous scanning, correlating results, routing findings to the right code owners, and preventing configuration drift are persistent bottlenecks. Despite the prevalence of these challenges, there is limited systematized knowledge or research on effective enterprise-scale integration patterns. This paper aims to bridge that gap by presenting common approaches, lessons learned, and practical considerations drawn from implementing these scanning systems in a Fortune 500 enterprise. Our focus is limited to SAST, SCA, secrets scanning.

We will be analyzing a few case studies to understand a couple of commonly used stategies and why they are unable to provide a complete integration and coverage for a large enterprise with diverse and complex workflows. To address these challenges, we will propose the Centralized AppSec Scan Factory design–a cloud-native framework designed to automate, orchestrate, and scale application security scanning across complex enterprise environments. The system consolidates SAST, SCA, and secrets scanning into a unified, centrally managed workflow. Leveraging event-driven orchestration through message queues and on-demand containerized scan agents, the platform dynamically allocates compute resources, maintains consistent configuration, and provides unified visibility and reporting. This architecture ensures reliable, scalable, and cost-efficient scanning coverage with minimal manual intervention, enabling organizations to operationalize AppSec at enterprise scale while addressing the gaps of current approaches and makes our systems ready for the evolving attack vectors on the supply chain front. The idea for a centralized scan factory was also presented in SANS Sec 540.

We will conclude this paper by proposing a few new approaches for achieving and doing static code scanning, and container scanning in larger organizations.

2. Background

All applications begin with source code. Any vulnerabilities present in the source code will propagate downstream into the application built from it. Source code is typically stored and managed in Source Code Managers (SCMs), which modern enterprises use to centralize code and support collaborative development. A codebase begins its journey on a developer's workstation within an Code Editor or an Integrated Development Environment (IDE), after which it is committed to an SCM. From there, CI/CD pipelines retrieve the code, run automated tests and checks, and ultimately deploy the built application. Because of this lifecycle, IDEs and CI/CD pipelines are the two primary points where static code analysis scans can be introduced.

Static code analysis [1] does not require code compilation and can be performed before packaging or building the application. Although certain SCA checks require a build step, these cases are exceptions rather than the norm. Consequently, a widely adopted strategy is embedding security scans directly into CI/CD pipelines. This is commonly done by adding a scan task before the build stage, or by triggering scans on pull-request events to do scan on the new code that is being added, providing early detection of vulnerabilities. Similarily, static code scan plugin can be setup on developers's IDEs to enable scanning on workstations.

However, while intuitive, this approach introduces significant operational constraints. Setting up CI/CD scanning or IDE scan plugin on thousands on pipelines and workstations demands extensive coordination with DevOps teams and often results in inconsistent coverage and configuration drift. Additionally, this approach delegates the responsibility and controls of scanning to the teams - relying on individual teams to setup, run, maintain, and remediate security findings. This reduces enterprise-wide visibility, and ties scanning reliability to the stability and configuration of each CI/CD workflow or IDE plugin configuration. In smaller environments, AppSec responsibilities can be delegated to developers or DevOps engineers with manageable overhead, even when multiple tools are used for different scan types. This decentralized model begins to break down as the scale of the engineering organization increases. As the number of repositories, teams, and toolchains grows, ensuring consistent and reliable scanning becomes significantly more challenging. Common problems include:

1) Managing access and permissions across numerous teams
2) Enforcing consistent scanning policies and remediation standards
3) Increasing operational burden on developers and DevOps teams
4) Creating confusion when multiple scanning tools coexist
5) Ensuring full and accurate scan coverage across all code assets
6) Prioritizing scans based on risk or business impact

Another common approach relies on the native integrations offered by COTS scanners. These integrations typically allow connections to SCMs, container registries, findings-management systems, and Security Information and Event Management (SIEM) platforms. In this model, the scanner automatically clones repositories and analyzes them for vulnerabilities. While effective in theory, COTS platforms often impose rigid opinions on scan frequency, which code branch gets scanned, parallelization, on-demand execution, and findings lifecycle workflows. In large enterprises with diverse development patterns and complex DevOps processes, these limitations frequently prevent the scanners from achieving complete coverage against new supply chain threats. We identified the following limitations with them:

1) Ability to select what code branches to scan
2) Scan frequency adjustment
3) Auto-import newly added repositories
4) On demand scanning
5) Custom scan triggers based on business rules
6) Customizations for security findings

3. Design: Centralized Static Code Scanning

In the previous section, we outlined two common approaches to implementing static code scanning and discussed their inherent limitations, particularly within large enterprises that operate diverse DevOps workflows, toolchains, and development practices. In this section, we introduce the design of our proposed solution, a centralized scanning architecture, and in the following section, we present case studies demonstrating how this design effectively resolves the shortcomings of the earlier approaches.

A security scanning system fundamentally consists of three stages: identifying assets to be scanned, executing the scans, and generating findings. Driving the remediation of those findings is the next most important step, although not directly part of the scanning system but this will what drives the risk enterprise risk down as the true purpose of scanning is to reduce risk by enabling timely and effective remediation. With this in mind, we begin by defining the key features that should be should be offered by our system:

3.1. Core Features

1) Automated scanning
2) Deterministic scan coverage
3) Self-service access for stakeholders
4) Tool-agnostic and extensible architecture
5) Customizable findings management workflows
6) Comprehensive findings-management dashboard

Our proposed system will be composed of multiple components, where each components will be responsible for one logical thing and there will be seperation of concerns between different components or subsystems. This architecture ensures that faults in one system are isolated from other components. We will be going over each component of our system and explain its responsibilities in details below.

3.2. Inventory Service

For any scan service the first step is to get an inventory of all the assets that needs to be scanned. In case of static code scanning the, the inventory service should have access to the SCMs and all the repositories. Knowing the complete inventory is very essential for any kind of security scanning, the total number of scannable assets will give you what is commonly known as denominator and the number of assets that you are able to scan is the numerator and the ratio of these two will give us our scan coverage. Obviously there are more nuances to this, as you would not want to scan everything that exists in your SCM, as some of the repositories might be old, never deployed, or deprecated, in which case you will subtract that from the denominator. The denominator should give us the total number of scannable assets, and the numerator should tell us the number of assets from the denominator that got scanned successfully.

This service should have be able to run on a periodic schedule so that it can keep an updated record of all the repositories, branches, commits etc. This information can be useful in doing a priority based scanning.

3.3. Scan Orchestration Service

The scan orchestration service schedules and initiates scans for assets managed by the inventory service. The scans can be scheduled based on the repository metadata - last commit time, number of committer, last scan time. The orchestration service can be designed in a way that is capable of creating scan messages and putting them on a queue - which will be consumed by the scan service.

In addition, the orchestration service enforces enterprise scanning policies, including rules that determine which assets should be scanned based on factors such as update recency, business criticality, or risk classification. This ensures consistent and policy-driven coverage across the organization.

This service is also to supposed to run on some schedule - as it will be our consumer of scan requests.

3.4. Scan Service

The scan service consumes queued scan requests and performs the actual scanning using the organization's chosen COTS scanning tools. Its architecture is intentionally scanner-agnostic and modular, enabling seamless replacement or integration of different scanning vendors as organizational needs evolve. This design reduces vendor lock-in and future-proofs the system.

This service needs to be scalable since it is the consumer of scan requests. As the number of requests in the queue increases the service should be able to scale up to run multiple scans in parallel to process the scheduled scan in a reasonable amount of time.

3.5. Security Findings Processing

Upon completion of a scan, the resulting findings are processed, normalized, and enriched before being forwarded to the centralized findings-management platform. Standardization ensures uniform formatting, severity classification, and prioritization across different scanners and asset types. Enriched findings can then be delivered to asset or code owners with actionable context, supporting effective remediation and enabling meaningful enterprise-wide risk reduction.

This service should be a modular enough to support sending findings to various platforms.

3.6. Findings Management Dashboard

The findings management dashboard serves as a role-based, self-service interface for developers, DevOps teams, security engineers, managers, and leadership. It provides a single pane of glass into all aspects of the centralized scanning ecosystem - including asset inventories, scan schedules, security findings, risk trends, and remediation progress.

Developers and DevOps teams can easily view the security posture of their repositories and track remediation work, while security teams gain visibility into systemic risks and scanning coverage. Leadership benefits from high-level insights into organizational risk, compliance status, and overall program performance.

This dashboard can be implemented in a variety of different ways and depends on the organization's preference. It can be a bespoke dashboard make using front-end technologies or it could be made using platforms like ServiceNow, PowerBI, Quicksight etc.

4. Implementation

The centralized scan factory can be implemented as a collection of modular services, each responsible for one of the core functions described in the section above. These services will be bespoke to the organization's environment and must be designed to integrate with its specific source code management platforms, DevOps platoforms, container registries, cloud providers, and centralized findings management systems. Together, these components abstract the complexity of scanning away from developers and other stakeholders, enabling full centralization while still allowing complete flexibility and customization.

In this model, the COTS scanning tool is used solely for executing scans, while all supporting workflow-scheduling, orchestration, prioritization, normalization, and reporting are fully managed by the scan factory. Application security engineers and tool administrators will continue to use the COTS scanner's native portal for tool configuration, license management, and operational settings.

These services can be deployed on any cloud provider depending on organizational preference. The specific cloud architecture, infrastructure components, and deployment models are beyond the scope of this paper, as these implementation details will naturally vary across organizations and should be tailored to meet their unique operational and security requirements.

5. Evaluation

In this section, we compare three distinct approaches to implementing an application security scanning program -

1) The traditional decentralized model, where different types of scans are performed by separate COTS tools integrated into CI/CD pipelines and maintained by developers or DevOps engineers;

2) The default-integration model, where organizations rely solely on the built-in connectors and automation features provided by each COTS tool; and
3) Our proposed centralized scan factory architecture.

The traditional pipeline-based approach places significant operational responsibility on development teams, who must configure, maintain, and troubleshoot scanner integrations while also interpreting scan results. The default-integration approach reduces this burden but is limited by the features and constraints of the vendor-provided integrations, often lacking flexibility in scheduling, scan-trigger mechanisms, prioritization logic, and multi-tool coordination.

In contrast, the centralized scan factory aims to provide full coverage, consistency, and flexibility by abstracting all scanning workflows into a unified system independent of development teams and individual tool limitations.

To evaluate these three models, we present a qualitative comparison using common industry workflows and real-world use cases encountered in large enterprises. We analyze how each approach performs in scenarios involving scan coverage, operational overhead, prioritization, scalability, and findings management. Additionally, we provide case studies drawn from a Fortune 500 environment to illustrate how each approach behaves in practice and to highlight the specific advantages of the centralized scan factory design.

5.1. Case Study: Distributed CI/CD scanning

CI/CD pipelines often feel like the most natural place to run security scans—whether static code analysis or container image scanning. The idea makes intuitive sense: shift left, catch issues early, and stop vulnerable code before it ever reaches the main branch or before an insecure image is pushed to a registry.

While this approach works well in smaller organizations, we ran into several challenges at enterprise scale. In an environment with tens of thousands of repositories, hundreds of thousands of branches, and thousands of pipelines, it quickly became difficult to maintain consistent visibility into what was actually being scanned. Relying on individual teams to correctly configure scan tasks, apply them to the right pipelines and branches, and enforce "break-the-build" rules for critical findings proved unreliable.

These issues become even more pronounced when every team works differently. Some teams have dedicated DevOps support, others do not; some development is handled entirely by contractors. Scan tasks frequently failed due to configuration issues, and in many cases developers simply disabled the tasks to keep their pipelines running. This distributed, self-implemented model also pushed the burden of triaging and remediating findings entirely onto developers, leaving the AppSec team with limited control and minimal oversight.

Access management to the various scanning tools added an additional layer of complexity. Each development team needed access, creating ongoing administrative overhead and introducing new questions around authentication, authorization, and data access boundaries. Developers also had to learn, manage, and interact with multiple different security tools—adding friction without guaranteeing consistent security coverage.

5.2. Case Study: xz-utils

In February 2024, a malicious backdoor was discovered in the Linux build of the xz utility bundled within the liblzma library in versions 5.6.0 and 5.6.1. This backdoor allowed an attacker possessing a specific Ed448 private key to gain remote code execution through OpenSSH on affected Linux systems. Because many Linux distributions ship xz-utils as part of their core utilities and because containerization commonly uses these distributions as base images, the vulnerability had the potential to propagate into container images and, ultimately, running containers.

When the incident surfaced, we needed to quickly determine whether any container images running in our environment relied on vulnerable base images. However, within our organization, container image scanning was performed in a fully decentralized manner using Snyk pipeline tasks. Development and DevOps teams were responsible for configuring and running their own scans, deciding what to scan and when, leaving the AppSec team with no centralized control or visibility.

During zero-day events like this, the organization's ability to assess exposure depends on two fundamental factors:

1) Whether we have a complete and accurate inventory of all deployed assets.
2) Whether all potentially vulnerable assets have been scanned.

Without a reliable asset inventory, we cannot confidently assert that we are not exposed. This creates a false sense of security, which is especially dangerous in a rapidly evolving zero-day scenario. In a decentralized model, ensuring that every team has configured their pipeline scans correctly is difficult even in smaller organizations. At enterprise scale, it becomes nearly impossible.

Zero-day incidents typically require ad-hoc rescans of all potentially affected assets because many commercial scanning tools add detection capabilities only after the vulnerability is disclosed. If scans completed before the tool's vulnerability database was updated, the earlier results would be incomplete. In a decentralized setup, the only way to address this would be to communicate with every DevOps team and request them to re-run their pipelines-a slow, unreliable, and operationally heavy process. During the xz-utils incident, we had no firm understanding of how many production images were actually being scanned across the organization.

Organizations that rely on registry-level integrations with commercial tools (COTS) are in a better position because they can automatically detect and scan images stored in the registries. If all production images reliably pass through those registries, the tool can provide a near-complete inventory and allow targeted scans for newly disclosed vulnerabilities. However, this approach still has significant limitations.

In our experience with several leading container security tools, we encountered issues such as:

1) No automatic import of new images pushed to repositories.
2) Scanning restricted only to the top N most recent or most used images.
3) No capability to trigger an ad-hoc scan across all images in use.

CNAPP and CSPM platforms that integrate directly with cloud tenants offer improved visibility by identifying container images running as part of cloud workloads and scanning them for vulnerabilities. However, these tools can only scan what they can see. Shadow IT, on-prem environments, and unmanaged deployments frequently fall outside their visibility. Additionally, AppSec programs are inherently limited by the feature sets and opinionated workflows of these tools. Customizing scan frequency, discovery intervals, or integrating findings with external systems such as the ServiceNow Vulnerability Response module can be rigid, inconsistent, or operationally cumbersome.

5.3. Case Study: NPM supply chain attacks

The recent surge in software supply chain attacks, particularly the ongoing Shai-Hulud NPM attacks highlights the growing prevalence of malicious packages distributed through public package managers. These attacks reinforce the need for organizations to maintain complete visibility into all third-party dependencies across their entire software ecosystem. Unlike traditional vulnerabilities, supply chain attacks can compromise any machine that clones a repository and installs dependencies, meaning exposure is not limited to deployed applications. As a result, scanning only the default branch, as most COTS scanners do - is insufficient.

This is where our centralized AppSec Scan Factory demonstrates significant strength. Leveraging this system, we were able to scan every branch across all repositories within 24 hours, giving us a comprehensive inventory of all packages in use across the organization. This visibility is critical because the threat is no longer isolated to production builds; even installing vulnerable dependencies from non-default branches can lead to compromised developer machines, leaked secrets, and downstream breaches.

With a centralized scanning model, the Inventory Service maintains a complete catalog of all organizational assets. The Scan Orchestration Service, fully controlled by the AppSec team, can be configured to scan every branch of every repository, something not feasible in a decentralized approach or with scanner-hosted compute. The actual scanning is still performed using the COTS scanner, but instead of relying on the vendor's limited cloud capacity, we run scans on our own scalable infrastructure, enabling us to execute hundreds of scans in parallel and bypass the constraints of standard SAST offerings.

Once scans are completed, the centralized pipeline also ensures that findings flow directly into the Centralized Findings Management System (CFMS). Since we control the ingestion process, we can tag these findings distinctly to support rapid identification, improved reporting, and streamlined remediation workflows. This greatly accelerates the organization's ability to assess and respond to zero-day supply chain threats.

6. Conclusion

Why should we trust them with our data, who is scanning the scanner, closed-source, scan as a library.

7. Scan as a Library: SaaL

References

[1] OWASP Foundation, "Owasp static code analysis," https://owasp.org/www-community/controls/Static_Code_Analysis.