Designing a Centralized Scan Factory for Enterprise-scale Staic Code Scanning

Rohit Chouhan
Application Security Engineer
Email:rchouhan@cs.stonybrook.edu

***Abstract*—Modern application security programs increasingly struggle to maintain effective static code scanning coverage at enterprise scale. Conventional approaches—such as pipeline-integrated scans and vendor provided integrations are poorly suited for large organizations with thousands of repositories, heterogeneous DevOps workflows, and rapidly evolving supply chain threats. Recent software supply chain attacks have further exposed the limitations of these models, demonstrating that restricting scans to default branches or production-bound code is insufficient to mitigate risk.**

We start by understanding the limitations of convetional approaches by assesing how they perform is real-world application security use-cases. After understanding the limitations of these approaches, we will present centralized static code scanning architecture—Centralized Scan Factory. Our proposed system decouples scan execution from individual development pipelines to centralize asset inventory, scan orchestration, execution, and findings management under the application security team. By leveraging organization-owned infrastructure and scanner-agnostic abstractions, the system enables deterministic scan coverage across all repositories and branches while avoiding the capacity and flexibility constraints of conventional commercial off-the-shelf (COTS) solutions.

We evaluate the system in a large and diverse enterprise environment comprising over 20,000 repositories. Analyzing the system through a real-world case study demonstrates the ability to achieve comprehensive scanning under 24 hours, significantly improving asset visibility and exposure detection, particularly for supply chain threats that can impact non-production code paths. We also evaluate the proposed system to understand how is overcomes the limitations of conventional approaches. Finally, we discuss how our proposed architecture serves as a foundation for building extensible, policy-driven application security platforms tailored to organizational needs. In scope of future work, we introduce Scanning-as-a-Library (SaaL)—an abstraction to modularize application security components in form of a library.

1. Introduction

Application Security (AppSec) spans a broad range of domains—including source code analysis, API and edge security, container security, AI security, and web application firewalls (WAFs). Modern applications are complex distributed systems composed of many interconnected software components. Securing each component is essential in securing a modern software application. This makes AppSec a very important pillar of of enterprise security strategy and Attack Surface Management (ASM).

A core responsibility of an AppSec program is static code analysis [1]—the process of analysing source code without running it. Since all the software is either compiled or interpreted from source code, identifying and remediating weaknesses at the source code level is one of the most effective ways to reduce downstream risk. Because of this reason static code scanning will be our main focus in this paper, it involves three types of scanning:

Static Application Security Testing (SAST)

Detects flaws such as SQL injection, cross-site scripting (XSS), insecure deserialization, and other code snippets that could lead to a vulnerability.

Software Composition Analysis (SCA)

Detects vulnerabilities and license issues in third-party dependencies (libraries) used in the source code.

Secret Scanning

Detects hardcoded secrets such as API keys, client credentials, and passwords.

Market offers a wide variety of SAST, SCA, and Secret Scanning tools, each with various integration options with source code managers, Continuous Integration/Continuous Deployment (CI/CD) systems, and findings platforms. Despite having a large number of AppSec scanning tools, we found very limited systematized knowledge or research on effective enterprise-scale integration and implementation of AppSec scanning. In this paper we will be focusing on two things - (a) Conventional approaches used in enterprises and its limitations. (b) Proposed centralized scan factory design that overcomes the limitations of conventional approaches.

We begin this paper analyzing a few case studies to understand some commonly used approaches and explore the reasons why they are unable to provide a complete coverage for a large enterprise with diverse and complex workflows. After summarizing the biggest limitation of these approaches, we will propose our Centralized Scan Factory—a cloud-native framework designed to automate, orchestrate, and scale application security scanning across complex enterprise environments. Our design is built upon an idea presented in the SANS course SEC 540: Cloud Native Security and DevSecOps Automation [2], where a centralized scanning is used instead of decentralized scanning in CI/CD pipelines.

2. Background

Source code is the origin of every software, it is the human-readable text of instructions that get translated to an executable computer program. Hence, a vulnerability present in the source code will propagate downstream into the built software. Source code is typically stored and managed using Source Code Managers (SCMs)—which is used to centralize code and support collaborative development. An application starts its journey on a developer's workstation inside a code editor or an Integrated Development Environment (IDE) as instructions written in a programming language. A SCM is then used to store, manage, and do collaborative development. In contemporary software development, code is then automatically tested, built, and deployed using CI/CD pipelines. Piplines are underpinned by cloud based runners or compute units that are capable of retrieving the code, running automated tests, buiding the application, and ultimately deploying built application. The source code lifecycle is shown in Figure 1.

Based on the source code lifecycle, IDEs and CI/CD pipelines are the two stages where static code analysis scans can be performed. Static code analysis does not require code compilation and thus can be performed before building an application. Although few specific SCA scans, mostly for Java Virtual Machine (JVM) languages using Maven build system require a build to generate a dependency tree, these may require a build before scanning. Consequently, a widely adopted strategy is embedding security scans directly into CI/CD pipelines. This is done by adding a scan task before the build stage, or by triggering scans on pull-request events to scan the new code that is being added, providing early detection of vulnerabilities.

Another commonly adopted approach is the use of static code scanning plugins integrated directly into developers' IDEs, enabling source code to be scanned locally on individual workstations. While this model appears intuitive and developer-friendly, it introduces significant operational, governance, and compliance challenges, particularly in large enterprises with heterogeneous development environments. At scale, deploying and maintaining IDE plugins or CI/CD scanning integrations across thousands of repositories, pipelines, and developer workstations requires extensive coordination with numerous DevOps and application teams. In practice, this level of coordination is difficult to achieve and sustain, often resulting in inconsistent adoption, configuration drift, and uneven scan coverage across the organization. As a result, security enforcement becomes fragmented, with no reliable mechanism to ensure that all assets are scanned consistently and according to enterprise policy.

More critically, this decentralized model delegates ownership and control of security scanning to individual development teams. Teams are responsible for configuring scanners, determining scan frequency, managing updates, interpreting results, and driving remediation. This creates a significant governance and compliance risk: security controls are no longer centrally enforced, scan coverage cannot be deterministically verified, and compliance reporting becomes dependent on self-attestation rather than measurable guarantees. In regulated or audit-driven environments, this lack of centralized visibility and policy enforcement undermines the organization's ability to demonstrate compliance and respond effectively to emerging threats.

Another popular approach by organization is enabled by using integrations offered by COTS tools. 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 scans them for SAST, SCA, and secret related vulnerabilities. While effective in theory, COTS platforms often impose prescriptive models on scan frequency, scan target branches, number of parallel scans, on-demand execution, and findings lifecycle workflows.

In the next section, we will be discussing two case studies to explore the limitations of conventional static scanning approaches when implemented for a large and diverse enterprise with thousands of code repositories.

3. Conventional scanning approaches

In this section, we examine conventional approaches to implementing static code scanning through two real-world scenarios encountered during our work on an AppSec team within a Fortune 500 enterprise operating more than 20,000 repositories and highly diverse DevOps workflows. In this environment, we had the opportunity to implement, operate, and evaluate coventional application security scanning approaches discussed in the previous section.

Through a set of representative case studies, we analyze the limitations of these approaches in detail and illustrate how they manifest at scale in large, complex enterprise environments. These examples highlight the operational, governance, and coverage challenges that arise when conventional scanning models are applied beyond small or homogeneous development ecosystems.

3.1. Case Study: CI/CD scanning

Integrating static code scanning into CI/CD pipelines can help identify vulnerabilities early in the development lifecycle and prevent vulnerable code from reaching production. While this approach can be effective for small or relatively homogeneous organizations, our experience shows that it introduces significant challenges at enterprise scale. In an environment with tens of thousands of repositories, hundreds of thousands of active branches, and thousands of CI/CD pipelines, maintaining consistent visibility into what is being scanned becomes increasingly difficult.

This model relies heavily on individual development teams to correctly configure scanning tasks, attach them to the appropriate pipelines and branches, and enforce build-breaking policies. In practice, this dependence on decentralized configuration resulted in substantial operational and

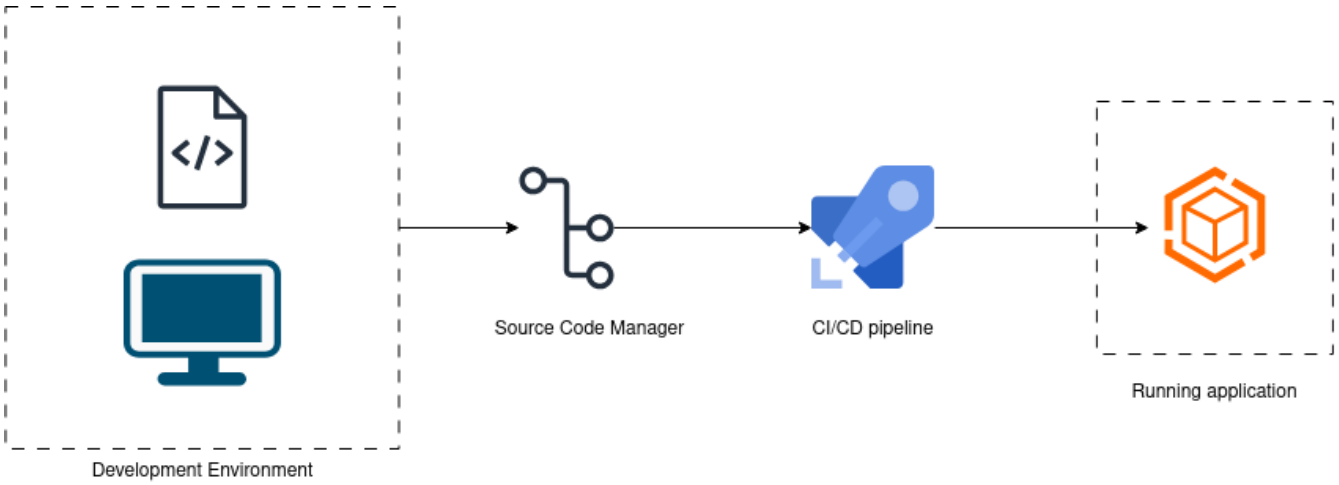

Figure 1. Source code lifecycle

governance challenges. We observed widespread misconfigurations, inconsistent or incorrectly enforced build breakers, and failures caused by expired credentials or mismanaged pipeline secrets. As a result, accurately tracking scan coverage proved extremely difficult—there was no reliable or centralized mechanism to determine how many pipelines required scanning, how many were correctly configured, or which assets were effectively covered. This lack of observability significantly undermined governance and compliance efforts.

Furthermore, when pipeline-based scan tasks failed due to configuration or infrastructure issues, developers often disabled the scanning steps to keep pipelines operational. This behavior, while understandable from a delivery perspective, effectively bypassed security controls and created silent coverage gaps. In this decentralized model, responsibility for triaging and remediating findings is also pushed entirely onto development teams, leaving the AppSec team with limited visibility, control, and enforcement capabilities. Key metrics such as scan coverage, scan success rates, and the volume and age of unresolved findings could not be reliably captured or centralized. Consequently, the security posture of the organization's source code became a fragmented, shared responsibility with minimal oversight and weak governance guarantees.

In the next section, we analyze a zero-day incident to further highlight the limitations of conventional scanning approaches. Even when vendor-provided integrations are used, significant gaps become evident during zero-day events, when rapid, comprehensive, and accurate assessment of exposure is critical to securing the environment.

3.2. Case Study: xz-utils backdoor

In February 2024, a malicious backdoor was discovered in the Linux build of the `xz-utils` bundled within the `liblzma` library in versions 5.6.0 and 5.6.1 [3]. The backdoor enabled an attacker possessing a specific Ed448 private key to achieve remote code execution through OpenSSH on affected Linux systems. Because many Linux distributions ship `xz-utils` as a core system component, and container images frequently use these distributions as base images, the vulnerability had the potential to propagate widely-extending from base operating system images into application containers and ultimately into running workloads.

Following public disclosure of the incident, it became critical to rapidly determine whether any container images in our environment were derived from vulnerable base images. This assessment required scanning all container images and doing static analysis of Dockerfiles to identify references to affected base images.

During a zero-day incident like this, an organization's ability to accurately assess exposure and drive timely remediation depends on the following:

1) A complete and authoritative inventory of all scannable assets
2) The ability to execute scans across all relevant assets without delay
3) Clear ownership and accountability for findings to enable efficient remediation

Without a reliable asset inventory, organizations cannot confidently assert that they are not exposed, resulting in a false sense of security, particularly dangerous in rapidly evolving zero-day scenarios. In a decentralized scanning model, ensuring that every team has correctly configured and consistently executed pipeline-based scans is challenging even in small organizations; at enterprise scale, it becomes virtually impossible.

Zero-day incidents often require ad-hoc rescanning of all potentially affected assets, as commercial scanners typically gain detection capabilities only after vulnerability signatures or databases are updated post-disclosure. Consequently, any scans completed prior to such updates may fail to detect the newly disclosed vulnerability. In a decentralized model, addressing this gap necessitates manual coordination with

each DevOps team to re-run pipelines—a slow, unreliable, and operationally intensive process that significantly delays accurate exposure assessment and remediation.

Organizations relying on out of the box integrations provided by COTS tools are marginally better positioned, as these integrations can automatically scan images in registries or source code from repositories in SCM. However, based on our experience with multiple leading COTS security platforms these integrations are not enough to assess complete risk profile. For example, the results will not reflect the current risk as they were done using older vulnerability database, no option to re-scan all assets in our environment, applications may be deployed from non-default branches, or developers workstations may be exposed by vulnerable dependencies introduced during local builds. Recent supply chain attacks have demonstrated the feasibility of compromising developer environments by installing backdoors when vulnerable library versions are pulled during development. These scenarios underscore the necessity of scanning all branches and all relevant artifacts to provide meaningful protection against modern supply chain threats.

3.3. Limitations

Based on the two case studies, we identify the following systemic limitations in conventional static code scanning models:

Lack of Authoritative Asset Inventory
Conventional approaches lack a centralized, continuously updated inventory of repositories, branches, pipelines, and artifacts. Without a reliable denominator of scannable assets, organizations cannot accurately measure scan coverage or confidently assess exposure during incidents.

Non-Deterministic and Incomplete Scan Coverage
Scan execution depends on correct pipeline configuration by individual teams, leading to inconsistent coverage across repositories and branches. Default-branch-only scanning and selective artifact scanning leave large portions of the codebase unexamined.

Decentralized Configuration and Governance Risk
Reliance on development teams to configure, maintain, and enforce scan policies introduces governance and compliance risks. Misconfigurations, disabled scan tasks, expired credentials, and inconsistent enforcement are common and difficult to detect centrally.

Poor Visibility and Observability at Enterprise Scale
There is no reliable mechanism to centrally track which pipelines are scanned, scan success rates, or coverage gaps. Key security metrics such as scan completeness, failure rates, and remediation progress—cannot be consistently captured or audited.

Security Controls Easily Bypassed Under Operational Pressure
When scan tasks fail due to infrastructure or configuration issues, developers often disable them to restore pipeline functionality. This behavior creates silent coverage gaps and undermines security controls without alerting security teams.

Inability to Rapidly Respond to Zero-Day Events
Decentralized scanning models are poorly suited for zero-day response, which often requires immediate, organization-wide rescanning. Coordinating pipeline re-execution across thousands of teams is slow, unreliable, and operationally prohibitive.

Delayed Detection Due to Scanner Database Updates
Commercial scanners typically detect vulnerabilities only after their databases are updated post-disclosure. Scans completed before the update provide a false sense of safety, necessitating rapid rescans which they do not efficiently support.

Limited Control Over Scan Scope and Prioritization
COTS integrations often restrict scan scope to a subset of branches or artifacts and offer limited flexibility in prioritization. This prevents organizations from tailoring scans based on asset criticality, business impact, or evolving threat intelligence.

Insufficient Coverage of Non-Production Risk
Conventional approaches focus primarily on deployed artifacts and default branches, overlooking risks in non-default branches and developer workstations. Modern supply chain attacks demonstrate that vulnerabilities in development environments can lead to credential theft, lateral movement, and downstream compromise.

Fragmented Ownership and Accountability
Findings are surfaced at the pipeline level with ownership implicitly assigned to individual teams, leaving AppSec teams with limited oversight and enforcement capability. This fragmentation complicates coordinated remediation and enterprise-wide risk management.

4. Design: Centralized Static Code Scanning

The limitations of conventional AppSec scanning approaches discussed in the previous section—particularly the lack of deterministic coverage, centralized governance, and operational scalability motivate the need for a fundamentally different model. In this section, we present the design of the proposed Centralized Scan Factory, an architecture that decouples scan execution from individual development pipelines by centralizing asset inventory, policy enforcement, scan orchestration, and findings management. The design aims to provide consistent, policy-driven scan coverage across large, heterogeneous enterprise environments while remaining extensible and vendor-agnostic.

At a high level, a security scanning system comprises three phases—identification of assets to be scanned, running scans on those assets, and generating security findings. Although remediation of findings is not the responsibility of the scanning system, it represents the most critical downstream outcome. The objective of security scanning is not just the discovery of findings, but the reduction of organizational risk through timely and effective remediation. With this in mind, we define the core capabilities that the proposed system must provide:

1) Complete asset inventory
2) Full automated scanning

3) Deterministic scan coverage
4) Role-based and self-service access
5) Tool-agnostic and extensible architecture
6) Scan prioritization and zero-day scanning support
7) Policy-driven, customizable findings workflow
8) Comprehensive findings-management dashboard

Our proposed system is composed of multiple loosely coupled components, each responsible for a distinct logical function, with a clear separation of concerns between subsystems. This modular architecture improves maintainability by ensuring that failures in one component are isolated from others. Next, we will define different components of the proposed Centralized Scan Factory and outline thier function and responsibilities.

4.1. Inventory Service

The inventory service is responsible for maintaining a complete and accurate record of all assets that require scanning. In the context of static code scanning, this service must integrate with SCM systems to discover and track all repositories under the organization's control. A complete asset inventory will help us calculate the scan coverage by expressing it in terms of ratio of scanned assets to the total number of scannable assets.

$$\text{Scan coverage } (\%) = \frac{\text{Scanned assets}}{\text{Total scannable assets}} \times 100 \quad (1)$$

The denominator represents the total set of relevant assets, while the numerator reflects the subset of those assets that have been scanned successfully. To remain accurate over time, the inventory service must execute on a periodic schedule and continuously synchronize with SCMs. The collected metadata such as repository activity, branch updates, and commit history can also be leveraged to support priority-based scanning, enabling the system to focus resources on the most recently modified.

4.2. Scan Orchestration Service

The scan orchestration service is responsible for scheduling and initiating scans for assets collected by the inventory service. Scan execution can be determined using repository metadata such as last commit timestamp, number of contributors, and the time elapsed since the previous scan. Based on these criteria, the orchestration service generates scan requests and enqueues them as messages for consumption by the scan service. This service enables us to have a deterministic scan coverage with control to trigger ad-hoc scans or full inventory scans.

In addition, the orchestration service enforces enterprise-wide scanning policies by applying rules that determine which assets should be scanned and when. These rules may be based on factors such as update recency, business criticality, or asset risk classification. By centralizing policy enforcement, the orchestration service ensures consistent, deterministic, and policy-driven scan coverage across the organization.

4.3. Scan Service

The scan service consumes queued scan requests and executes the scans using a COTS security scanning tools. Its architecture is modular and scanner agnostic, allowing different COTS scanners to be integrated with minimal system changes as organizational requirements evolve. This approach reduces vendor lock-in and improves the long-term adaptability of the system.

The scan service is designed to scale horizontally in response to workload demand—as the volume of scan requests in the queue increases, the service can dynamically provision additional workers to execute multiple scans in parallel. This elastic scaling capability enables the system to process large numbers of scheduled scans within acceptable time bounds, even in large and highly active environments.

4.4. Security Findings Processing

Upon completion of a scan the identified security findings are processed, normalized, and enriched before being forwarded to the centralized findings-management platform. Normalization ensures consistent formatting, severity classification, and prioritization across different scanners and asset types. Enrichment augments findings with additional contextual information such as asset metadata, ownership, and risk signals—enabling findings to be routed to the appropriate code or asset owners with clear, actionable guidance. This standardized and enriched ingestion process supports efficient remediation and facilitates meaningful, enterprise-wide risk reduction.

4.5. 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 should be able to view the security posture of their repositories and track remediation progress. Security teams gain visibility into systemic risks, scan coverage, and trends across the environment. Leadership benefits from high-level insights into organizational risk, compliance posture, and overall program effectiveness. The findings-management dashboard can be implemented using a variety of approaches, depending on organizational preferences and existing tooling. It may take the form of a custom dashboard built with custom front-end technologies, or be implemented using established enterprise platforms such as ServiceNow, Power BI, or Amazon QuickSight.

5. Implementation

The centralized scan factory can be implemented as a collection of modular services, each responsible for a

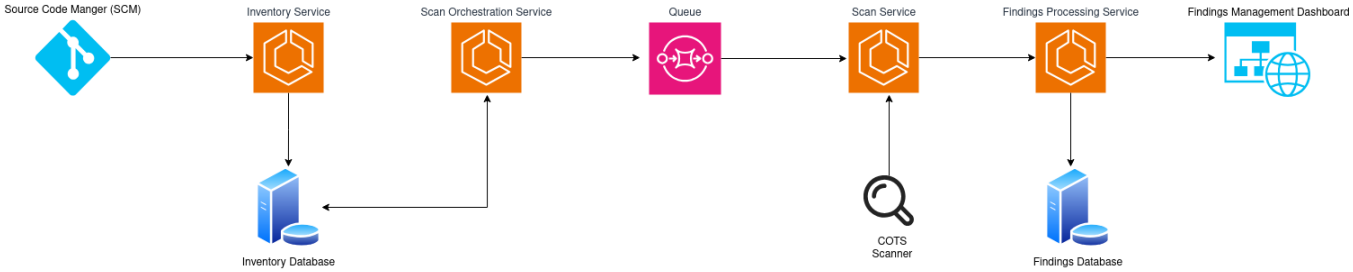

Figure 2. Centralized Scan Factory: High-level implementation

distinct core function described in the previous section. These services are intentionally bespoke to the organization's environment and are designed to integrate with its specific source code management platforms, DevOps tooling, business requirements, remediation and governance policies, cloud providers, and centralized findings-management systems. Collectively, these components form a centralized scan factory that abstracts the operational complexity of security scanning away from developers and other stakeholders, while enabling full centralization, flexibility, and customization for the AppSec team.

In this model, COTS scanning tools are used exclusively for executing scans. All supporting workflows—including scheduling, orchestration, prioritization, findings normalization, and reporting—are fully managed by the centralized scan factory. Application security engineers and tool administrators continue to interact with the COTS scanner's native interface for configuration, license management, and operational settings. Developers and DevOps engineers, however, are insulated from the complexity of configuring and maintaining scanner integrations, reducing operational overhead and inconsistency across teams.

The services comprising the scan factory can be deployed on any cloud provider, with the choice driven by organizational preference and existing infrastructure. Detailed cloud architecture, infrastructure components, and deployment models are intentionally out of scope for this paper, as these implementation details vary widely across organizations and must be tailored to meet unique operational, scalability, and security requirements.

Based on our experience implementing this model within a Fortune 500 enterprise, we present a high-level deployment prescription. The inventory service can be deployed on virtual machines or managed compute platforms such as serverless functions or container services. It executes on a periodic schedule to ensure that the inventory database accurately reflects the organization's current asset landscape.

The scan orchestration service acts as the producer within the system, generating scan requests and placing them onto a managed queue. A queuing system introduces an additional layer of flexibility and resilience, enabling prioritization strategies, retry mechanisms, and dead-letter queues for failed scans. First-in, first-out (FIFO) queues may be used, although alternative prioritization strategies can be applied depending on organizational requirements. The use of a producer–consumer pattern also allows the system to absorb spikes in scan demand, as queued messages persist even when downstream services are temporarily constrained.

The scan service functions as the consumer and is responsible for executing scans using the selected COTS tool. We recommend deploying this service on a scalable container platform, such as a managed container service or a Kubernetes cluster. The cluster should scale dynamically based on the depth of the scan queue, ensuring reasonable scan turnaround times while supporting high degrees of parallelism. This elasticity enables both scheduled and ad-hoc scanning at enterprise scale, accommodating thousands of scan targets without manual intervention.

Once scans are completed, vulnerability findings are normalized and processed in accordance with organizational policies before being stored in a database or forwarded to a centralized findings-management system. These findings can then be surfaced to asset owners through a dedicated findings dashboard, implemented using custom front-end technologies or enterprise reporting platforms.

This prescription reflects our practical experience, but the centralized scan factory itself is intentionally unopinionated. Its primary objective is to provide the flexibility and extensibility required to achieve comprehensive, last-mile coverage—addressing gaps that are difficult or impossible to close using CI/CD-embedded scanning or out of the box integrations offered by COTS tools. The reference architecture is shown in Figure 2.

6. Evaluation

In Section 3 we discussed limitations of coventional AppSec scanning approaches through a two enterprise-scale case studies. In this section, we evaluate the proposed Centralized Scan Factory when it was put through its paces by a recent software supply chain attack. After that we will see how the centralized scan factory compares to conventional approaches and different core AppSec functionality. We will also see if our proposed design is able to overcome some of the limitations of conventional approached.

6.1. Case Study: NPM supply chain attacks

The recent software supply chain attack - Shai-Hulud NPM supply chain attack [4] (September 15, 2025), highlighted the use of package manager to spread malware.

TABLE 1. COMPARISON BETWEEN CENTRALIZED SCAN FACTORY AND COVENTIONAL APPROACH

AppSec component	Conventional approaches	Centralized AppSec factory
Operational burden	On developers and DevOps team in case of CI/CD scans	No operational burden on other stakeholders
Inventory management	Dependent on what the COTS scanner offers	Complete control with the AppSec team
Scan frequency customization	Dependent on what the COTS scanner offers	Complete customization with scan orchestration and scan service
Scan target customization for zero-day scanning	Dependent on what the COTS scanner offers and how the CI/CD tasks are configured	Can be customized
Findings management	Dependent on what the COTS scanner offers	Finding workflows can be customized by the AppSec team
Vulnerability and posture management dashboards	Dependent on what the COTS scanner offers	Custom dashboards can be created

Attacks like this underlines the need for organizations to maintain comprehensive visibility into all third-party dependencies across their entire software stack. Unlike traditional vulnerabilities, modern supply chain attacks like this can compromise any system that installs the compromised dependency, meaning exposure is not just limited to deployed application.

This is where the centralized AppSec Scan Factory demonstrates a significant advantage. Our designed scan factory was able to scan every branch across all repositories (more than 500K branches) within 24 hours, producing a complete inventory of all third-party packages in use across the organization. This level of visibility is critical because supply chain threats are no longer confined to production builds; installing compromised dependencies from non-default branches can lead to developer workstation compromise, credential leakage, and downstream breaches.

The parallelism offered by our design enabled scanning of 500K targets in 24 hours. The scan service was deployed as a horizontally scalable cluster, which can scale up based on the length of the scan queue. By adjusting the scale limits of the scan service one can change how much time it takes to scan on the assets in an environment. For example, if we assume that each scan takes an average of 3 minutes to complete and if we are only running 1 scan at a time, then we will be able to scan only 20 targets in 1 hour and 480 targets in 24 hours. Let's assume 500 scans per day for easier calculation. We intend to scan 500K targets in 24 hrs. With the current rate it would take 1000 days to complete scanning all the assets. Meaning we have to multiply the scan speed by 1000. If we can run 1000 scans in parallel this can be achieved since we will then be scanning 20K scans in 1 hrs and approximately 500K scans in 24 hours.

In the centralized model, the inventory service maintains an authoritative catalog of all organizational assets, and the scan orchestration service fully controlled by the AppSec team can be configured to scan every branch is each code repository. This capability is difficult to achieve with decentralized, pipeline-based scanning or with vendor-hosted scanner compute, both of which impose limits on scale and flexibility. Although scanning itself is still performed using COTS tools, execution is shifted to organization-controlled, scalable infrastructure. This enables hundreds of scans to run in parallel and removes the throughput and concurrency constraints commonly imposed by standard scanning tools.

Once scans complete, findings are ingested into a centralized findings-management system through a controlled pipeline. Because the ingestion process is fully managed, findings can be enriched and tagged to enable rapid identification, improved reporting, and streamlined remediation workflows. This centralized handling significantly accelerates the organization's ability to assess and respond to any zero-day threats.

In contrast, traditional pipeline-embedded scanning places substantial operational responsibility on development teams, who must configure, maintain, and troubleshoot scanner integrations while also interpreting results. Vendor-provided integrations reduce some of this burden but remain constrained by rigid assumptions around scheduling, trigger mechanisms, prioritization logic, and multi-tool coordination. The centralized scan factory addresses these limitations by abstracting scanning workflows into a unified, policy-driven system that is independent of individual development pipelines and vendor-specific constraints. This approach enables consistent coverage, improved scalability, and greater operational resilience at enterprise scale.

6.2. Centralized scan factory vs conventional approaches

In the table 1 we summarize how our proposed Centralized scan factory compares to the conventional approach on various aspects of the AppSec scanning. Some of the limitations were discussed in the previous sections. It is clear that proposed design overcomes the limitation of the two convetional approaches discussed throughout the paper.

7. Conclusion

The centralized AppSec Scan Factory presented in this work explores a decentralized approach to application security scanning. We saw that conventional scanning models such as pipeline-embedded CI/CD scans or vendor offered out of the box integrations are insufficient in addressing the scale, complexity, and evolving threat landscape faced by large enterprises. Modern attacks, particularly software supply chain compromises like Shai-Hulud, require security architectures that provide comprehensive visibility, flexibility, and centralized control capabilities that are difficult to achieve using off-the-shelf solutions alone.

Our proposed approach represents a viable design for AppSec static code scanning architecture. Rather than positioning the scan factory as a single, monolithic solution, we view it as a reference design for building bespoke security platforms tailored to organizational needs. In practice, achieving robust and effective application security often requires a degree of customization that COTS tools are unable to provide either due to inherent product limitations or gaps in how these tools integrate into complex enterprise environments. The scan factory design builds on top of COTS scanner but overcomes its limitation by using custom components and integrations as described in the design.

8. Future Work: Scanning as a Library (SaaL)

Building on the centralized scan factory concept, we propose Scanning as a Library (SaaL) as a future direction for enterprise AppSec tooling. The core idea is to abstract the foundational components of the scan factory into a reusable library that organizations can extend to implement their own centralized scanning solutions.

Such a library would encapsulate common functionality including parallel scan orchestration, scalable infrastructure constructs, asset inventory management, findings normalization, and integration with findings-management systems. It would expose well-defined interfaces for interacting with COTS scanners - such as triggering scans, retrieving results, and querying repository inventories while allowing organizations to retain full control over execution and data handling.

Our experience suggests that many elements of large-scale scanning systems (e.g., inventory system, scan orchestration) are highly repeatable across organizations. Modularizing these components into a reusable library would significantly lower the barrier for security engineers to build custom solutions tailored to their environments. This approach is particularly compelling when combined with AI-assisted development workflows, which further reduce implementation effort while enabling greater flexibility than rigid vendor-provided integrations.

An additional benefit of the SaaL model is improved data sovereignty. Organizations that wish to prevent source code, scan targets, or findings from leaving their organizational boundary can fully control data flow by running scanners and orchestration logic on internally managed infrastructure. This is often difficult or impossible with vendor-hosted scanning models.

Overall, Scanning as a Library represents a promising evolution of the centralized scan factory concept, one that balances the strengths of COTS scanning engines with the flexibility, scalability, and control required for enterprise-scale application security. Future work will focus on formalizing this library, evaluating its applicability across different organizational contexts, and exploring how it can support additional security domains beyond static code analysis.

References

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

[2] SANS, "Sec540: Cloud native security and devsecops automation," https://www.sans.org/cyber-security-courses/cloud-native-security-devsecops-automation.

[3] Wikipedia, "Xz utils backdoor," https://en.wikipedia.org/wiki/XZ_Utils_backdoor.

[4] Datadog, "Shai-hulud," https://securitylabs.datadoghq.com/articles/shai-hulud-2.0-npm-worm/.