# Centralized Scan Factory for 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-hosted scanning platforms 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.**

**In this paper, we present the design and implementation of a centralized static code scanning architecture, referred to as a Centralized Scan Factory. The proposed system decouples scan execution from individual development pipelines and centralizes asset inventory, scan orchestration, execution, and findings management under the control of the application security team. By leveraging scalable, 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 enterprise environment comprising over 20,000 repositories and diverse development practices. Our results demonstrate the ability to achieve comprehensive branch-level scanning within operationally feasible timeframes, significantly improving asset visibility and exposure detection—particularly for supply chain threats that impact non-production code paths. Finally, we discuss how this architecture serves as a foundation for building extensible, policy-driven application security platforms tailored to organizational needs, and talk about another novel idea - scanning-as-a-library (SaaL). An abstraction providing components in form of a library to the application security engineers, enabling them to create a scanning system based on the needs of the organization.**

## 1. Introduction

Application Security or 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). Modern applications are complex, distributed systems composed of numerous interconnected components and services. Securing each of these components is essential in securing a modern application, 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 static code analysis [1]- the process of analysing source code without running it. Since all software is either compiled or interpreted source code, identifying and remediating weaknesses at the source code level is one of the most effective ways to reduce downstream risk. Static code analysis usually involves three components - Static Application Security Testing (SAST), Software Composition Analysis (SCA), and secret scannig. Typically, enterprises rely on COTS scanners to perform static code analysis. SAST scans helps in uncovering flaws such as SQL injection, cross-site scripting (XSS), insecure deserialization, and other code snippets that cloud lead to a vulnerability. SCA scan detect vulnerabilities and license issues in third-party dependencies used in the code. An increasing number of scanners also include capabilities to identify hardcoded secrets such as API keys, client credentials, and password - this capability is called as secret scanning.

While the market offers a wide variety of SAST, SCA, and secrets-scanning tools, each with various integration options with source code managers, CI/CD systems, and findings platforms; large organizations still face significant challenges in operationalizing them. At enterprises 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.

Lack of knowledge and discussion aroung enterprise-level AppSec scanning motivated us to systemize our experience - which was gained by doing AppSec in a Fortune 500 company. Through this paper we aim to answer the following quesitons - (a) What are some popular approaches used in enterprises and why they are not enough? (b) Alternate approach that overcomes the limitations of the discussed popular approaches?

To answer the first question we will analyze a few case studies to understand some commonly used stategies and 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; this is inspired by the idea presented in the SANS course SEC 540 [2].

## 2. Background

Source code is human-readable text of instructions that get translated to executable computer program. Any vulnerability present in the source code will propagate downstream into the build software. Source code is typically stored and managed by using Source Code Managers (SCMs) - which are used to centralize code and support collaborative development. Any application starts its journey on a developer's workstation within an code editor or an Integrated Development Environment (IDE) as instructions written in a programming language, after which it is commit to a SCM. After that the code needs to be built or packaged so that it can run as part of an application. In contemporary software development world this is done through a system called as Continuous Integration/Continuous Deployment (CI/CD) pipelines. These piplines are cloud based runners or compute units that are capable of retrieving the code, running automated tests, and ultimately deploying the built application. The lifecycle of the source code is shown in Figure 1.

Based on source code lifecycle, IDEs and CI/CD pipelines are the two points where static code analysis scans could 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 scan 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 the source code directly on workstations. However, while intuitive, this approach introduces significant operational constraints especially for enterprises wih a large and diverse software development environment. Setting up CI/CD scanning or IDE scan plugin on thousands on pipelines and workstations demands extensive coordination with DevOps teams which is very hard to acheive in a large organization, this can lead to lack of coverage and configuration drift. Additionally, this approach delegates the responsibility and controls of scanning to the development 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 enterprises with homogeneous tooling, development practices, and limited repositories, 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 for numerous teams
2) Enforcing scanning policies and remediation standards
3) Increased operational burden on development teams
4) Increased confusion with mutiple scan tools
5) Ensuring full and accurate scan coverage across all code assets
6) Inability to prioritize 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 SAST, SCA, and secret related vulnerabilities. While effective in theory, COTS platforms often impose rigid opinions on scan frequency, scan target branches, parallelization, on-demand execution, and findings lifecycle workflows. In large enterprises with diverse development patterns and complex DevOps processes, these limitations prevent the scanners from achieving complete coverage in terms of securing the enterprise from modern attacks that can be carried out by exploiting source code vulnerabilities. COTS offered native integrations are not a panacea, connected these scanners to an enterpise environment still leaves a lot of gaps. We identified the following limitations some popular COTS scanners in the market:

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

Next, we will look at some case studies to understand the limitation of these two approaches when implemented for a large and diverse enterprise with thousands or code repositories.

## 3. Limitations: Conventional Approaches

Based on our experience operating within a large enterprise environment comprising over 20,000 repositories and diverse DevOps workflows, we implemented and evaluated both conventional application security scanning approaches. Over time, the limitations of these approaches became increasingly evident. In the following case studies, we examine these limitations in detail and demonstrate how they manifest at enterprise scale.

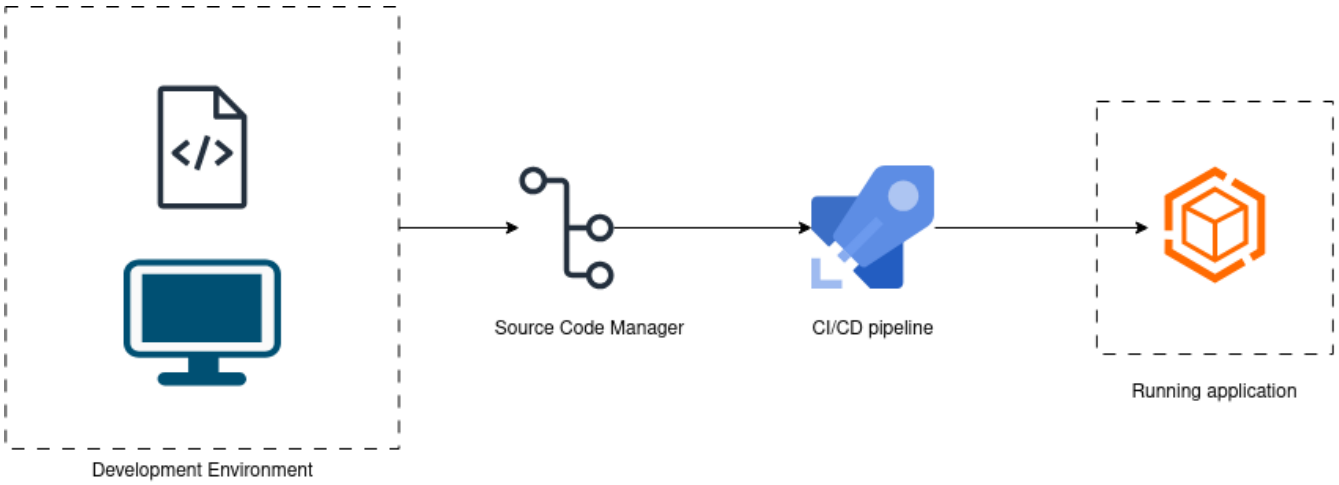


Figure 1. Lifecycle of source code

### 3.1. Case Study: Distributed CI/CD scanning

CI/CD pipelines often feel like the most natural place to run SAST, SCA, and secrets scans, assuming that all development teams at an enterprise are using CI/CD pipelines in their software development lifecycle process. Putting a scan in the pipeline can help identify vulnerabilities early, and stop vulnerable code before it ever reaches the main branch for deployment. 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 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 if not unmanageable.

In our experience we found that the pipeline scan tasks frequently failed due to configuration issues, and in many cases developers simply disabled the tasks to keep their pipelines in a runnable state, circumventing the scan process entirely. 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. With this approach, the AppSec team cannot assess key metrics like - scan coverage, scan success rate, and number of findings. This makes the security of the source code a distributed and shared responsibility with very limited oversight.

### 3.2. Case Study: xz-utils backdoor

In February 2024, a malicious backdoor was discovered in the Linux build of the xz utility [3] 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 application containers 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, there was a need to quickly determine whether any container images running in our environment were derived from a vulnerable base image. This was achieved by scanning all the container images and looking for dockerfiles using vulnerable image as the base image by doing static code scanning. However if static code scanning and contianer image scanning is performed in a fully decentralized manner using pipeline tasks the development and DevOps teams are responsible for configuring and running their own scans. Leaving the AppSec team with no centralized control or visibility. During zero-day events like this, the organization's ability to assess exposure and remediate the vulnerabilities depends on the following:

1) Complete and accurate inventory of all assets
2) Scans on all assets
3) Assignment of finding to the right owner for remediation

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 scanning 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 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.

Organizations that rely on native commercial tools (COTS) integrations are in a better position because they can automatically detect and scan images stored in registries and repositories in SCM. However, this approach still has significant limitations. In our experience with several leading COTS security tools, we identified the following limitations:

1) No automatic import of new images or repositories pushed to the registry or the SCM
2) Scanning is often restricted only to N most recent images or only the default branch of a repository
3) No capability to trigger an ad-hoc scan across all assets

While scanning the top N images or the default (main) branch in case of a repository seems like a good approach, it fails to capture the complete risk present in the environment. For instance, a running image might not be the latest images in a registry, an application might be deployed form a non-default branch, there could be a vulnerability that allows a threat actor to compromise a developer's workstation, and many more. In the recent time we have seen supply chain attacks capable of installing a backdoor on workstations when a vulnerable version on library is intalled on the developer's machine - this means that scanning all the branches, default and non-default is essential to ensure that there are no instances of those library being installed.

# 4. Design: Centralized Static Code Scanning

In the previous section, we outlined two common approaches for implementing static code scanning and discussed their limitations, particularly in large enterprises operating with diverse DevOps workflows, toolchains, and development practices. In this section, we present the design of our proposed solution: the Centralized Scan Factory.

At a high level, a security scanning system consists of three primary stages: identifying assets to be scanned, executing scans against those assets, and generating security findings. While remediation is not a direct function of the scanning system itself, it is the most critical downstream outcome. The ultimate goal of scanning is not merely to surface findings, but to reduce organizational risk by enabling timely and effective remediation. With this objective in mind, we first define the core features that the proposed system must provide.

The key features guiding our design are as follows:

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) A comprehensive findings-management dashboard

The 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 and ensures that failures in one component are isolated from others.

In the remainder of this section, we describe each component of the Centralized Scan Factory and outline its responsibilities. The following section then details the implementation of the system and discusses practical considerations observed during deployment.

## 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.

Maintaining a comprehensive inventory is essential for effective security scanning. The total number of scannable assets constitutes the denominator, while the number of assets that are successfully scanned forms the numerator. The ratio of these two values provides a quantitative measure of scan coverage. However, not all repositories present in an SCM should necessarily be included in the denominator. Some repositories may be deprecated, experimental, never deployed, or otherwise out of scope. Such repositories must be explicitly excluded to ensure that coverage metrics accurately reflect the organization's active attack surface.

The inventory service should therefore apply filtering and classification logic to determine which assets are considered scannable. The resulting 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 or highest-risk assets.

## 4.2. Scan Orchestration Service

The scan orchestration service is responsible for scheduling and initiating scans for assets managed 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.

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,

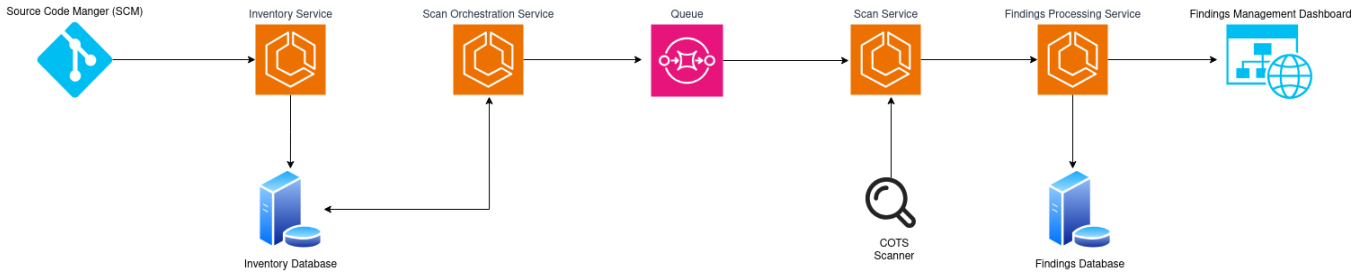


Figure 2. Centralized Scan Factory: High-level implementation example

deterministic, and policy-driven scan coverage across the organization.

### 4.3. Scan Service

The scan service consumes queued scan requests and executes the actual scans using the organization's selected COTS security scanning tools. Its architecture is intentionally scanner-agnostic and modular, allowing different scanning vendors to be integrated or replaced 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 resulting 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 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 created on platforms like ServiceNow, PowerBI, Quicksight etc.

## 5. 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 previous section. These services will be bespoke to the organization's environment and should be created to integrate with its specific source code management platforms, DevOps platoforms, container registries, cloud providers, and centralized findings management systems. Together, these components will create a centralized scan factory to abstract the complexity of scanning 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. However we are presenting a generic architecture diagram of the setup which can serve as a starting point.

## 6. Evaluation

In Section 3 we discussed two conventional approaches for AppSec scanning and discussed their limitations through

some case studies. Now that we have proposed our centralized scan factory, we will be going through another case study to analyze how this system performs and see whether or not it is able to fix some of the limitations of traditional approaches.

### 6.1. Case Study: NPM supply chain attacks

The recent surge in software supply chain attacks - most notably the ongoing Shai-Hulud NPM campaign, highlights the increasing prevalence of malicious packages distributed through widely used package managers. These attacks underscore 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 can compromise any system that clones a repository and installs dependencies, meaning exposure is not limited to deployed applications or default branches. As a result, scanning only the default branch as is common with many Commercial Off-The-Shelf (COTS) scanners is fundamentally insufficient.

This is where the centralized AppSec Scan Factory demonstrates a significant advantage. Using this architecture, we were able to scan every branch across all repositories 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.

In the centralized model, the inventory service maintains an authoritative catalog of all organizational assets, while the scan orchestration service fully controlled by the AppSec team can be configured to scan every branch of every repository. This capability is difficult to achieve with decentralized, pipeline-based scanning or with vendor-hosted scanner compute, both of which impose practical 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 zero-day supply chain 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.

## 7. Discussion & Future Work

The centralized AppSec Scan Factory presented in this work explores a non-traditional approach to application security scanning. As discussed earlier, conventional scanning models such as pipeline-embedded scans or vendor-managed 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 an initial step toward a more adaptable and resilient AppSec 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 Commercial Off-The-Shelf (COTS) tools are unable to provide either due to inherent product limitations or gaps in how these tools integrate into complex enterprise environments.

### 7.1. 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 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 enterprise 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.