Centralized Scan Factory for Staic Code Scanning

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

***Abstract*—**

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]: which is the process of analysing source code without running it, hence static. Since all software is starts from source code, identifying and remediating weaknesses at the source code level is one of the most effective ways to reduce downstream risk. Typically, enterprises rely on Commercial Off-The-Shelf (COTS) scanners to perform static code analysis, specifically Static Application Security Testing (SAST). SAST scanning helps in uncovering flaws such as SQL injection, cross-site scripting (XSS), insecure deserialization, and other code snippets that lead to a vulnerability. These COTS scanners also provide Software Composition Analysis (SCA), which adds the capability to detect vulnerabilities and license issues in third-party dependencies used in the applications. 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 in the world on AppSec scanning tools.

While the market offers a wide variety of SAST, SCA, and secrets-scanning tools, each with various integration options for 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. This motivated us to look into AppSec scanning integration, relying on our experience of working closely on AppSec team in a Fortune 50 company. Through this paper we aim to answer the following quesitons - (a) What are some popular approaches used in enterprises and what are their limitations (b) Is there an 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 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 AppSec Scan Factory design - a cloud-native framework designed to automate, orchestrate, and scale application security scanning across complex enterprise environments; this is built on the idea presented in the SANS course Sec 540 [2]. The system consolidates SAST, SCA, and secrets scanning into a unified, centrally managed workflow by 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. Finally, we will conclude this paper by proposing another new idea that offers a very different perspective on AppSec scanning.

2. Background

All applications are built from their source code, hence any vulnerability 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 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 code written in any programming language, after which it is committed to an 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 pipelines 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 showing in Figure 1 Based on this lifecycle, IDEs and CI/CD pipelines are the two primary 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 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 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 and often results in inconsistent 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 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 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 day 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 in an enterprise setup.

3. Limitations: Conventional Approaches

Based on our experience in a relatively large enterprise with more than 20K repositories, and diverse DevOps process, we had experience with implementing both the conventional approaches. We will try to understand the limitations of them through the following case studies.

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 thier 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 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 if not unmanageable.

In out 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,

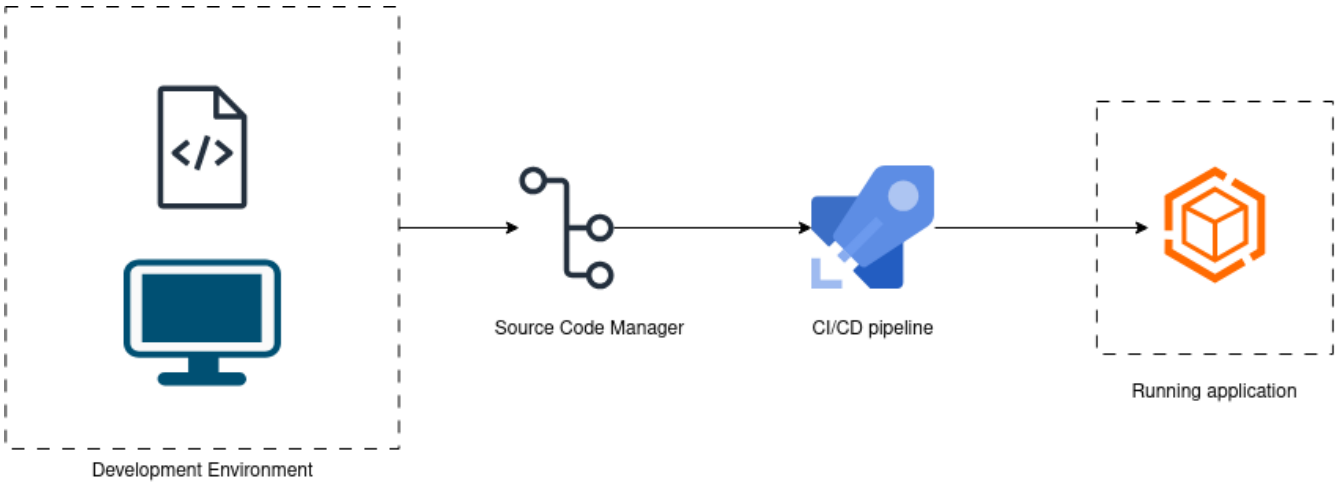

Figure 1. Lifecycle of source code.

the AppSec team cannot assess the scan coverage, scan efficacy, remediation rate, number of findings, etc. making 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 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, there was a need to quickly determine whether any container images running was derived from a vulnerable base image - this can be achieved by scanning all the container images and also 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 deployed assets.
2) Scanning all the assets.
3) Assign the findings to the 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 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.

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 branch (production branch) in case of a repository seems like a good approach, real world risks are often missed by these scans. Many times a running image might not be the latest images in a registry. There are instances where a development environment deploying source code form non-default branches have vulnerabilities that allows a threat actor to compromise production instance of an application. Moreover, 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 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 - Centralized Scan Factory.

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. The core features on which we want to build our system are:

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 is built using multiple components, where each components will be responsible for one logical thing with seperation of concerns between different components or subsystems. This architecture ensures that faults in one system are isolated from other components. Next, we will be going over each component of our system and explain its responsibilities in details below, and in the following section we will be explaining the implementation of the system.

4.1. Inventory Service

This service will be responsible for getting 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 the denominator and the number of assets that you are able to scan is the numerator - the ratio of numerator and denominator will give us 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.

4.2. Scan Orchestration Service

The scan orchestration is responsible for scheduling and initiating 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.

4.3. 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 will be horizontally scalable to accomodate the number 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.

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. 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.

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.

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.

6.1. 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.

7. Conclusion

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

8. Scan as a Library: SaaL

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.