Scribe

A Use Case of Using Scribe Trust Hub

How can Scribe's tools be used to manage supply chain risks in an on-prem use case?

TABLE OF CONTENTS

INTRODUCTION

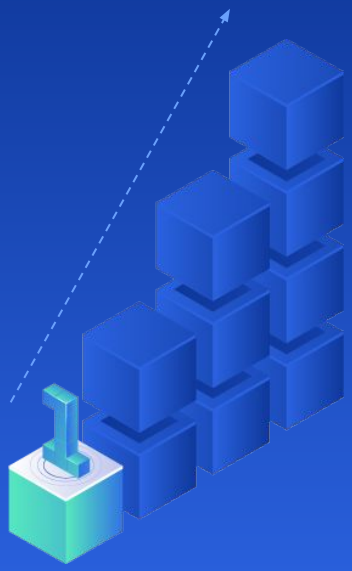
Scribe is a security platform designed to minimize users' risk from their software supply chain.

As a software producer, you can use Scribe to apply security guardrails to your SDLC based on security posture, risk analysis of open-source dependencies, and enforcement of development practices. You can also use Scribe to share select attestations about your compliance and software's security with your customers. Previous posts on this blog provided broad reviews of how these goals can be achieved.

Typical use cases for using the Scribe Security suite of tools usually deal with a cloud-native organization, with access to Scribe's own SaaS platform from their developers' environments, as well as access to other relevant SaaS platforms (e.g., SCA tools, static analysis tools) and unlimited capability to update known vulnerabilities publications, AV signatures, etc. However, along with these 'mainstream' use cases, we see a steady demand to achieve the same level of supply chain assurance from organizations with a segregated development environment - either with limited connectivity to public SaaS services via security gateways or completely air-gapped. This post describes how Scribe's suite of tools can be used in such use cases.

STAGE 1

Collecting, signing, and storing the evidence



Scribe's approach to software supply chain assurance is evidence-based. We generate and collect a variety of evidence types covering your entire SDLC and use the evidence to evaluate risk, identify tampering and discover violations of security practices required in your organizational policy (AKA "Guardrails"). For an on-prem deployment we provide a set of tools for evidence generation and collection to accommodate many types of software build pipelines. The most basic type of evidence collected is the Software Bill of Materials (SBOM). Scribe's SBOM creation tool, **Valint,** is provided as a stand-alone command line tool that can be integrated into a build-script or executed from a Jenkinsfile. Valint is also available as an Azure DevOps task that can be downloaded from the Azure marketplace and deployed on a private Azure Cloud or an on-prem Azure Stack deployment. Other CI platform-integrations are also available (Git, Bitbucket, Circle CI, to name just a few).

The Valint tool is capable of generating several types of evidence: SBOMs, provenance objects and generic evidence. The generic evidence capability allows you to use almost any type of artifact as evidence - practical examples are: an SBOM provided by a 3rd party (not generated by Scribe), a SARIF file representing a static analysis report, an externall SCA report.

Other types of evidence could be collected by the Scribes policy engine directly by accessing API endpoints of developers' tools providing important insights regarding their security posture, e.g. querying Azure Pipelines to verify if branch protection is activated or querying a Git repo to check if signed commits requirement is enforced.

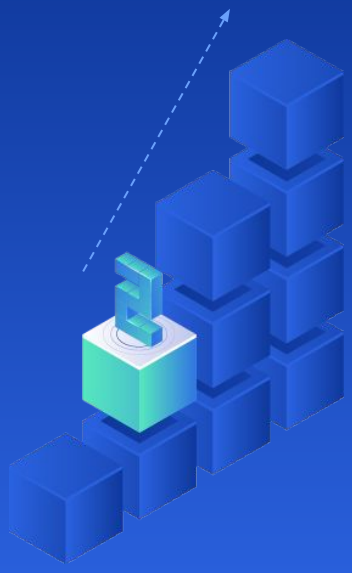
All the evidence generated or collected by valint need to be stored in an accessible location. This evidence store will be later accessed at the policy verification phase. The stored evidence will be used as input parameters for the policy evaluation.

The two main options to implement an evidence store for on-prem setups are either to use a folder in a file system accessible through the local network or using a local OCI registry.

All the evidence collected by Valint can be signed using locally deployed PKI keys. The signature can be later verified as part of the policy evaluation process.

STAGE 2

External OSINT sources - bridging the (air) gap



Once you generate and collect SBOMs from your software build pipeline, one of the most valuable security controls to implement is to check the dependencies listed in the SBOM against repositories of known vulnerabilities (e.g., NVD), and against reputation repositories (e.g. listings of CVSS, EPSS, KEV or Reputation Scorecard). Such checks could produce reports of vulnerabilities present in your product through direct or indirect dependencies, and also detect dubious components such as components no longer maintained.

These reports could be used as input parameters to policy rules such as:

"Fail the build if at least one component is discovered in any of the SBOM containing a vulnerability with CVSS score of 8 or above."

"Generate a warning if a dependency is discovered with Reputation Scorecard rating of 3.5 or below"

"Stop an artifact in the Admission Controller if it contains a component that is blocklisted"

Scribe provides an extensive set of policy-as-code rules facilitating the implementation of security controls required to comply with the best practices of secure SDLC (Software Development Life Cycle), as well as specific software supply chain assurance frameworks, such as SSDF and SLSA. Each policy implementation contains a Rego language file implementing the policy logic and a .yaml configuration file. The required policy files can be checked out from the repo and transferred to the closed development environment.

However, having only the policy-as-code files and the evidence generated inside the closed environment is not enough. Certain policies such as in the examples above, requires access to relevant repositories, such as known vulnerabilities repositories or package reputation repositories.

There could be several possible solutions to make such repositories accessible inside a closed environment - Some organizations have already provisioned vulnerabilities scanning inside the closed environment using tools such as Nexus IQ or JFrog XRay with a local copy of vulnerability data with periodic updates delivered through a security gateway. In this case Scribe can utilize the vulnerabilities reports generated by these tools as an input.

In case your organization does not have such tools already deployed, Scribe provides you with our own tools enabling vulnerability scans in a closed environment, and the proper procedures to provide periodic updates of vulnerability data, suitable for use with security gateways or cross-domain-solutions.

STAGE 3

Evaluating policy compliance

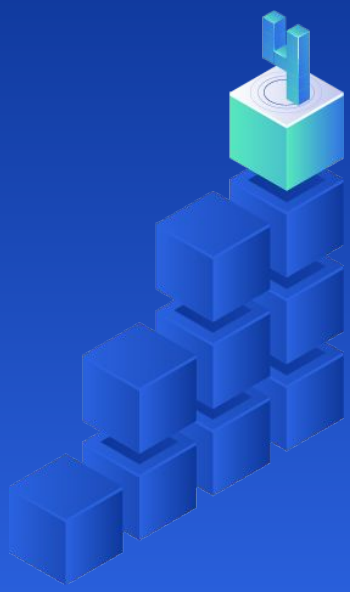
Once you have all the required components in place and accessible from the closed environment (i.e., the evidence collected, the policy-as-code files and the external data sources) Valint can be used to verify or enforce any type of policy.

Examples of simple policies are verifying that all SBOMs are signed and the signature is verified, or checking that a specific product component has a SLSA provenance object which is properly signed.

More complex policies may include sets of security controls to comply with the SSDF standard, like component version pining, or Git branch protection. Another example is verifying that all commits in a Git repository are properly signed by authorized developers, according to an allow-list, where all developers are identified by their PKI certificates.

STAGE 4

Judgment day - enforce or alert



The deployment of the Valint tool at the policy evaluation stage determines how the policy result is treated. A policy 'fail' result could be used to enforce the stopping of a build pipeline, stop the code at the admission controller gateway, generate and send an alert to the relevant stakeholders or just write an error message to a log file to be viewed if and when needed.

CONCLUSION

The stages described in this article provide a practical implementation of an on-prem deployment of Scribe's tool set. Such a deployment facilitates supply chain risk management through continuous collection and signing of security-related evidence throughout all stages of the SDLC. The collected and signed evidence is used to create, verify and enforce SDLC guardrails into the development process to ensure the delivery of a trusted product.

Each on-prem deployment might be slightly different. Scribe provides a versatile set of tools that can be tailored to a specific use case's requirements.