Microsoft

NCCoE – DevSecOps practices Industry Workshop Session

Agenda

- Microsoft’s internal DevSecOps practices
  - Security Development Lifecycle
  - Secure OSS Consumption
- Distributing Supply Chain Conformance Data
  - Today: OCI Registries
  - Future: SCITT

Microsoft Security Development Lifecycle (SDL)

SDL defines best practices for secure design and developing secure code throughout the lifecycle of software development.

SDL has been extended to provide guidance on securing the software supply chain

- SDL created in 2004 focused on securing products and services we ship, then published in 2006
- Initially SDL was updated annually
- Moved to agile release process in 2015
- Incorporate new requirements to address technology changes
  - Cloud Services
  - Open Source Software
  - ML/AI
- Cross-company SDL Working Group
  - Most engineering teams represented
  - Team proposes and reviews changes/updates
  - Often in response to incidents
- SDL has been extended over the years to address Secure Software Supply Chain requirements such as the SSDF (our approach to EO14028 compliance)

Implementing the SDL

Security Requirements

- Strong preference for controls being built into the platform (no need to “opt in”) and automation (zero-config tools)
- Clear, actionable security guidance for software engineers (security patterns, remediation)

Secure Design

- Threat modeling, architecture reviews, specialists in areas like cryptography

Tooling & Automation *(essential for scale and coverage)*

- SAST (CodeQL, secret/credential scanning, etc.) and DAST (web-based scanners, fuzzing)
- Supply chain security (open source project detection, scanning, alerting)
- Inserted at various points (IDE, push, PR build, periodic, etc.) where they make sense
- Compliance measurement, claim generation, and evidence management (Liquid, SCITT, etc.)

Open Source Software (OSS) Secure Supply Chain (SSC) Framework

A recent extension of the SDL that defines how to securely consume OSS dependencies into the developer's workflow

We've been implementing this since 2019

- We published the OSS SSC Framework in August 2022 [OSS Secure Supply Chain Framework (microsoft.com)]
- It's a threat-based risk-reduction approach toward secure consumption

Real World OSS Supply Chain Threats

Threats	Real examples	Mitigation via OSS SSC Framework	Framework requirement reference
Accidental vulnerabilities in OSS code or Containers that we inherit	SaltStack	Automated patching, display OSS vulnerabilities as pull requests	UPD-2, UPD-3
Intentional vulnerabilities/backdoors added to an OSS code base	phpMyAdmin	Perform proactive security review of OSS	SCA-5
A malicious actor compromises a known good OSS component and adds malicious code into the repo	ESLint incident	Ability to block ingestion via malware scan, single feed, all packages are scanned for malware prior to download	ING-3, ENF-2, SCA-4
A malicious actor creates a malicious package that is similar in name to a popular OSS component to trick developers into downloading it	Typosquatting	OSS provenance analysis, single feed, all packages are scanned for malware prior to download	AUD-1, ENF-2, SCA-4
A malicious actor compromises the compiler used by the OSS during build, adding backdoors	CCleaner	Rebuilding OSS on trusted build infrastructure ensures that packages don't have anything injected at build time	REB-1
Dependency confusion, package substitution attacks	Dependency Confusion	Single feed, securely configure your package source mapping	ENF-1, ENF-2
An OSS component adds new dependencies that are malicious	Event-Stream incident	All packages are scanned for malware prior to download, single feed	SCA-4, ENF-2
The integrity of an OSS package is tampered after build, but before consumption	How to tamper with Electron apps	Digital signature or hash verification, SBOM validation	AUD-3, AUD-4
Upstream source can be removed or taken down which can then break builds that depend on that OSS component or container	left-pad	Use package-caching solutions, mirror a copy of OSS source code to an internal location for Business Continuity and Disaster Recovery (BCDR) scenarios	ING-2, ING-4
OSS components reach end-of-support/end-of-life and therefore don't patch vulnerabilities	log4net CVE-2018-1285	Scan OSS to determine if it is at end-of-life	SCA-3
Vulnerability not fixed by upstream maintainer in desired timeframe	Prototype Pollution in lodash	Implement a change in the code to address a zero-day vulnerability, rebuild, deploy to your organization, and confidentially contribute the fix to the upstream maintainer.	FIX-1
Bad actor compromises a package manager account (e.g. npm), with no change to the corresponding open source repo, and uploads a new malicious version of a package	Ua-parser-js	OSS provenance analysis, single feed, scan OSS for malware	AUD-1, ENF-2, SCA-4

OSS SSC Framework Maturity Model

- The guide lists out the requirements and organizes it into a maturity model, where each level has different themes

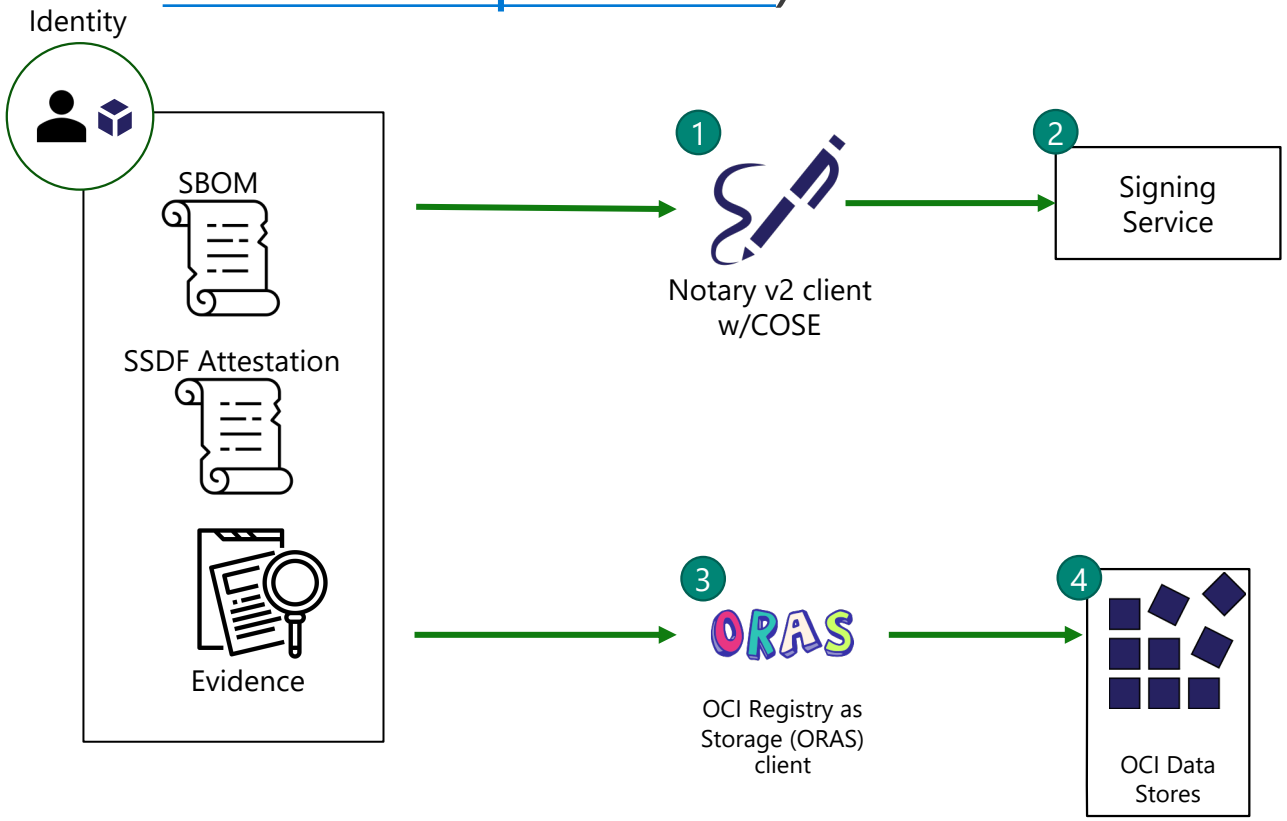
Level 1	Level 2	Level 3	Level 4
Minimum OSS Governance Program • Use package managers • Local copy of artifact • Scan with known vulns • Scan for software licenses • Inventory OSS • Manual OSS updates	**Secure Consumption and Improved MTTR** • Deny list capability • Scan for end life • Have an incident response plan • Auto OSS updates • Alert on vulns at PR time • Audit that consumption is through the approved ingestion method • Validate integrity of OSS • Secure package source file configuration	**Malware Defense and Zero-Day Detection** • Clone OSS source • Scan for malware • Proactive security reviews • Enforce OSS provenance • Enforce consumption from curated feed	**Advanced Threat Defense** • Validate the SBOMs of OSS consumed • Rebuild OSS on trusted infrastructure • Digitally sign rebuilt OSS • Generate SBOM for rebuilt OSS • Digitally sign protected SBOMs • Implement fixes

Our guide is published here: [GitHub - microsoft/oss-ssc-framework: Open Source Software Secure Supply Chain Framework]

Supply Chain Conformance Data Exchange - Today

Industry specifications – CBOR Object Signing and Encryption (COSE) and Open Container Initiative (OCI) Registries

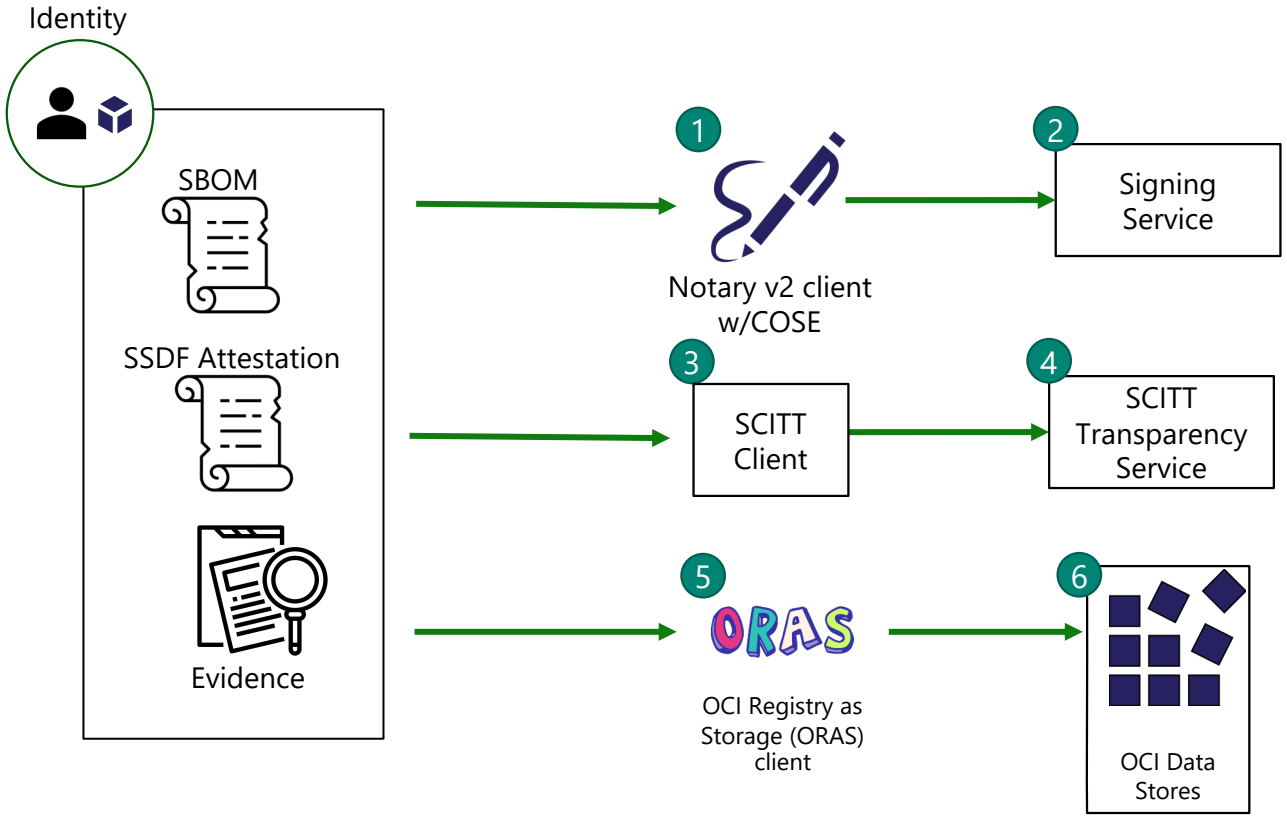
- Conformance data can be exchanged today using OSS client implementations (Notary, ORAS) based on industry specifications (COSE, OCI Distribution Specification)



Supply Chain Conformance Data Exchange - Future

Industry Standards - Supply Chain Integrity Transparency and Trust (SCITT)

- SCITT standards being developed within the Internet Engineering Task Force (IETF) add support for immutable record of activity using decentralized transparency services



Resources

- Evidence Signing
  - COSE Specification RFC 8152 - CBOR Object Signing and Encryption (COSE) (ietf.org)
  - Notary Client https://notaryproject.dev/
- Evidence Store
  - OCI Distribution Specification https://github.com/opencontainers/distribution-spec/blob/main/spec.md
  - ORAS Client https://oras.land/
- SCITT
  - IETF-SCITT GitHub Repository
  - IETF-SCITT Mailing List

Microsoft

Thank you