Security Best Practices

2023-06-21-TAC-LfnProjectSecurity.pptx

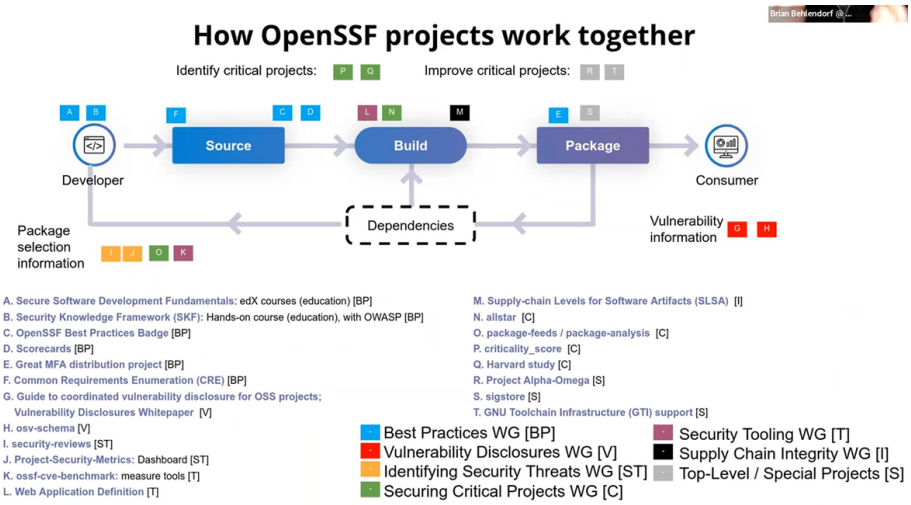
Share your community's best practices that might be applicable to other projects.

Vulnerability Reporting

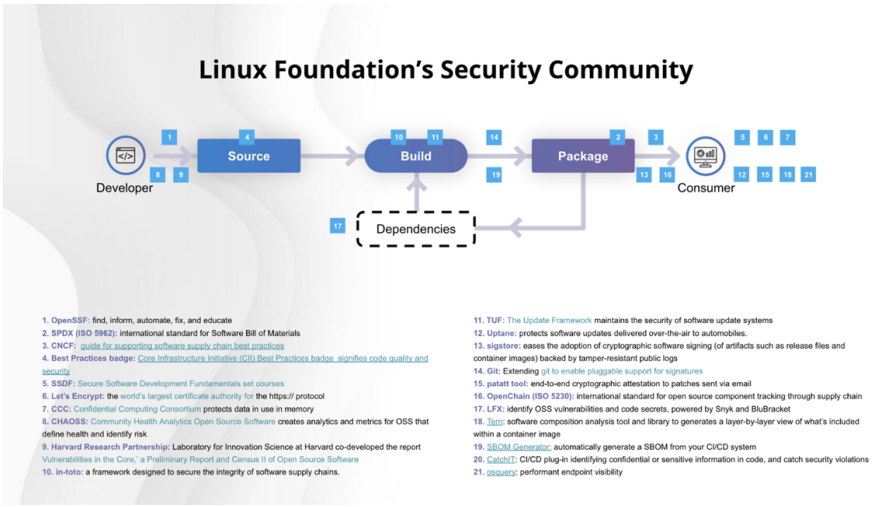
- Create formal bug reporting process for the project.

Demonstrate Security Awareness

- Adopt OpenSSF badging and show progress to Gold badging.
  - https://github.com/coreinfrastructure/best-practices-badge
  - https://github.com/ossf/scorecard



- Linux Foundation security community



Practice Secure Lifecycle Management (per release)

- Retire technical debt.
- Include vulnerability management.
- Identify and remove unmaintained code from release package.
- Upgrade dependencies (libraries, databases, language versions.)
- Interface security (APIs, GUIs, Portals.)
- Remove all secrets from code.
- Goal for LFN projects: create an LFN security cookbook that documents how security best practices and tools can be implemented and used across LFN project.
- Starting point: ONAP implementation of Security Best Practices, LFX security.

Documentation

- Provide security transparency to the users of the open source code.
- Known open vulnerabilities in project code and dependent packages.
- Vulnerabilities closed in the release by fixing code and upgrading packages.
- Secure and resilient configuration settings
- Integration points with external security system
    - Certificate Authority (CA)
    - Certificate management protocol support
    - LDAP
    - OAuth Authorization Server
    - Log management systems
- Language version dependencies
- Third party component and version dependences
    - databases such as Cassandra
    - messaging such as Kafka

CI/CD best practices

- Automate rejection of insecure merges (planned PoC in ONAP)

Architecture

- Include logging of security events in the project.
- For stand-alone projects: integrate authentication and authorization into web interfaces and APIs.
- Support confidentiality on all interfaces. For HTTP, this typically means supporting TLS 1.2+.
- Where containers are used: follow the CIS benchmarks, such as the Docker benchmarks.
- Where K8S is used: follow the CIS Kubernetes benchmarks.

Supply Chain Security

- Secure the commit process to prevent unauthorized code from being included in an open source project
- Digitally sign all code produced by the project using an X.509 code signing cert issued by a public certificate authority (CA).
- The Linux Foundation has a secure signing process.
- Create an SBOM for each application produced by a project team.
- Use SPDX, CycloneDX or SWID for SBOM format.
- Digitally sign the SBOM with an x.509 signing certification issued by a public CA.
- SBOMs can be automatically generated in the CI/CD pipeline using a software composition analysis tool.

Software Bill Of Materials (SBOM)

- ONAP work - https://wiki.onap.org/display/DW/Software+Bill+of+Materials
- ONAP presentation - SBOM_DBOM.pptx
- Other references
    - https://github.com/opensbom-generator/spdx-sbom-generator
    - https://github.com/CycloneDX/cyclonedx-maven-plugin
- Scripts for automated SBOM generation by Maven
    - Robert Varga, February-24-2021:

      As per my AI, I have reached out to Jessica. The LFN-side of the build tools is tracked here, the corresponding patch here .

      For the purposes of OpenDaylight, I think using a tool outside of our build system (Maven) is less than optimal. Since OpenDaylight has a project managing default build system policy, I have filed blocked URLODLPARENT-280 - Generate an SBOM for artifacts RESOLVED to track this effort. There are maven plugins for both SPDX and CycloneDX. The former is under development and it seems to have a number of issues, while the latter seems to be a breeze to integrate.

      So the initial test is to add the plugin execution via a trivial patch and then let the normal build pipeline treat SBOMs just as any other maven artifact. This results in metadata being correctly propagated to properly propagate to Nexus even for snapshots (https://nexus.opendaylight.org/content/repositories/opendaylight.snapshot/org/opendaylight/odlparent/opendaylight-karaf-empty/10.0.0-SNAPSHOT/, scroll down to see latest artifacts) and also to staging repositories, for example here: https://nexus.opendaylight.org/content/repositories/odlparent-2204/org/opendaylight/odlparent/opendaylight-karaf-empty/10.0.0/ .

      From what I can tell the SBOM is reasonably complete, but it would be nice if someone could validate it to see whether we need to provide more metadata

Integrate security testing in CI/CD

Test Type	Description	Example Tools	LF Provided Tools

Static Application Security Testing (SAST)	Detects vulnerabilities in the code written by the project team. Some SAST tools provide autofix capabilities.	Snyk Veracode SonarCloud	Snyk Blubracket
Software Composition Analysis (SCA)	Detects known CVEs in third party package used by the project team in their code.	Sonatype NexusIQ Veracode Mend	NexusIQ (ONAP only) Snyk
Dynamic Application Security Testing (DAST)	Detects vulnerabilities in a running application by simulating attacks to all interfaces and examining its running state, and its responses to the simulated attacks. Requires the project team to create a traffic file that can be replayed in the pipeline.	AppScan	??
Container Scanning	Detects vulnerabilities in container base images and open source dependencies used in base images and Dockerfile commands. Some products include autofix capabilities.	Snyk Aqua/Trivy JFrog Xray StackRox	Snyk
Code Coverage Testing	Verifies and validates code quality by evaluating the amount of code executed by automated tests.	SonarCloud	??
Code Quality	Measures the quality of the code produced by the project team. Code quality measures include maintainability, clarity, testability, portability, robustness, reusability, complexity, safety and security. Never hardcode secrets in code.	SonarCloud	??

Other tools

- Remote attestation (SEDIMENT)
- 3rd party API checks

Managing dependencies

- Direct dependencies are straightforward.
  - Upgrade direct dependencies to the latest supported version with each release.
  - Prioritize upgrading direct dependencies with effective vulnerabilities.
    - An effective vulnerability is one that can be executed by the application containing it.
    - Not all vulnerabilities in a package are in code that is used by the application.
  - Prioritize upgrading direct dependencies on deprecated versions.
  - Prioritize upgrading direct dependencies containing vulnerable dependencies.
  - Dependencies with zero-day critical vulnerabilities may require upgrades and emergency releases. A good example is Log4J.
- Transitive dependencies are more difficult.
  - May be resolved by upgrading to a newer version of the direct dependency containing it.
  - Some transitive dependencies can be upgraded independent, but this requires more testing.
- How to automate the mitigation?
  - Some container scanning tools can automatically upgrade open source dependencies used in container base images.