Open Source Software Paradigms

Lecture - 09

Security of Open source Software

Lecturer: Biniam Behailu
Addis Ababa Science and Technology University

Learning Objectives

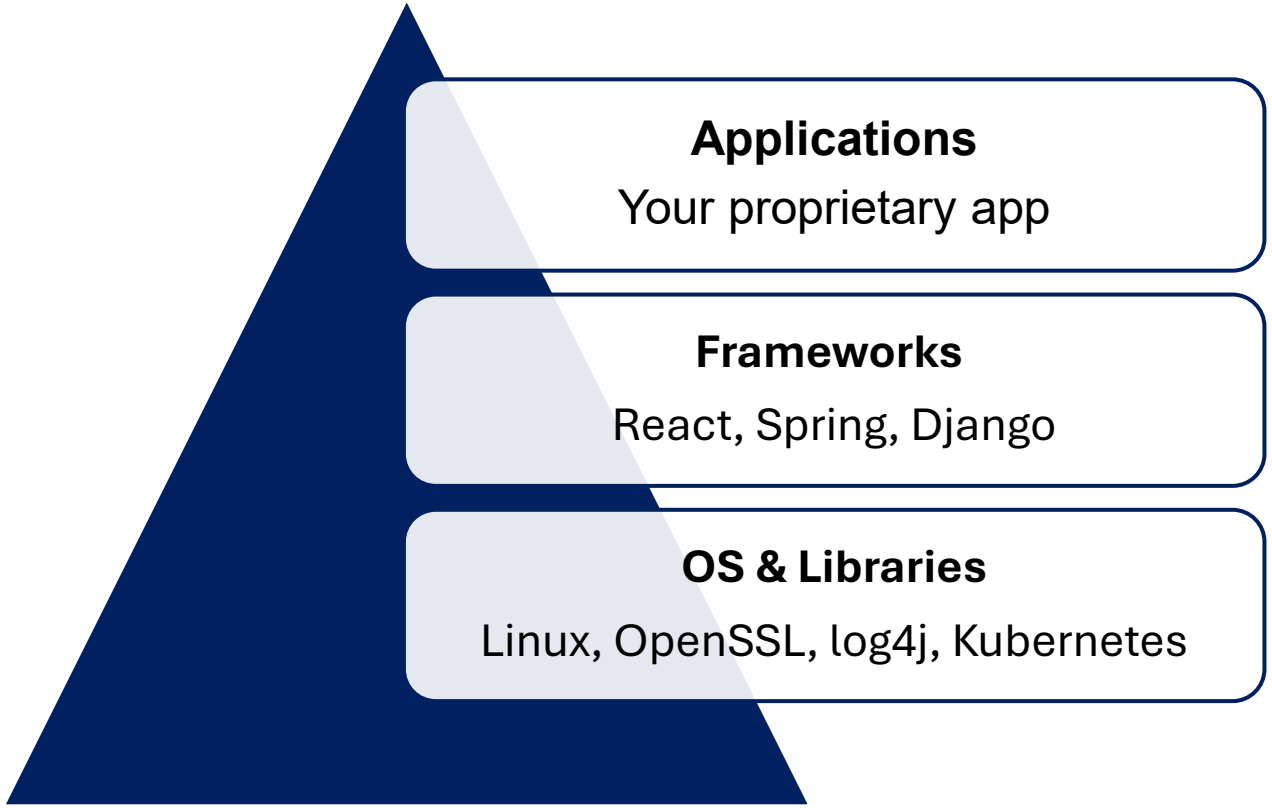
By the end of this lecture, you will be able to:

- Analyze the unique security advantages and challenges inherent in open source software.
- Identify common security risks in the modern software supply chain, such as typosquatting and maintainer burnout.
- Explain the key phases of the open source security lifecycle, from discovery to remediation.
- Evaluate and prioritize open source vulnerabilities using metrics like CVSS and EPSS.
- Recommend practical best practices and tools for securing open source usage within an organization.

Contents

The Pervasiveness of OSS

- Over 90% of all software codebases contain open source components.



" Open source software is the foundation of the digital world. "

The Open Source Paradox

- Open source software powers modern technology from cloud infrastructure to mobile applications.
- Yet its transparency, which enables security auditing, also exposes potential vulnerabilities to attackers.
- This paradox demands a nuanced understanding of both the benefits and risks inherent in open source development.

***Image source**: M. Bedadi, "What are the benefits of Open Source Software?," Digital Laoban, Dec. 7, 2020. [Online]. Available: https://www.digitalaoban.com/benefits-of-open-source-software/. [Accessed: 07-Oct-2025].*

The Open Source Paradox

Organizations today face a critical challenge

- How to harness the innovation and cost-effectiveness of open source while maintaining robust security standards across their entire software supply chain.

Transparency

Code is publicly auditable by the community

Vulnerability

Vulnerabilities are visible to malicious actors

Speed

Rapid patches when issues are discovered

The Security Advantages of OSS

Transparency

- **No "Security through Obscurity"** - When a vulnerability is found, patches can be developed and released quickly by the community.
- **Independent Verification** - Security researchers, academics, and companies can audit the code.
- **Example**: The OpenSSL codebase was extensively reviewed post-Heartbleed.

Rapid Patching & Community Response

- **Speed -** You can see the code. No hidden backdoors (in theory).
- **Collaboration -** Multiple experts can collaborate on a fix.
- **Example**: The Linux kernel team is renowned for its rapid response to critical vulnerabilities.

Flexibility and Control

- **No Vendor Lock-in** - You are not dependent on a single vendor for security updates.
- **Custom Patches** - You can create and apply your own security patches if the upstream project is slow.
- **Example**: A company can harden a Linux distribution for its specific high-security needs.

Standardization & Auditability

- **Common Codebase** - A single, well-audited OSS library (e.g., OpenSSL) is used by millions, leading to collective hardening.
- **Compliance** - Source code availability can help with certain regulatory audits (e.g., for government use).

The Security Challenges & Risks of OSS

The Modern Software Supply Chain

- Before evaluating any open source solution, establish clear technical and business criteria to guide your decision.

" Every step is a potential attack vector "

Software Composition Analysis (SCA)

- **Software Composition Analysis (SCA)** is the process of identifying all open source components in your codebase.
- Most organizations don't have a complete Bill of Materials (BOM) for their software.

Do you know every OSS library and its version in your application right now?

***Image source**: "Little White Man Question Mark Thinking Three-Dimensional Illustration," PNGTree, [Online]. Available: https://pngtree.com/freepng/little-white-man-question-mark-thinking-three-dimensional-illustration_18163648.html. [Accessed: 07-Oct-2025].*

Vulnerability Management

- Thousands of new OSS vulnerabilities are published every year (CVEs).

  - ☞ Time from vulnerability introduction to discovery.
  - ☞ Time from discovery to patch availability.
  - ☞ Time from patch availability to your deployment.

The "Typosquatting" & Dependency Confusion Attack

- **Typosquatting** - Malicious packages with names similar to popular ones (e.g., react vs reakt).
- **Dependency Confusion** - Attackers publish a malicious package with a higher version number to a public registry than the one used in your private repository.
- **Result** - Your build system automatically downloads and executes malicious code.

***Image source**: "Fayil:Hacker behind PC.svg," Wikipedia, 7 Oktoba 2017. [Online]. Available: https://ha.wikipedia.org/wiki/Fayil:Hacker_behind_PC.svg. [Accessed: 07-Oct-2025].*

Maintainer Burnout & Abandoned Projects

The Human Factor

- Many critical OSS projects are maintained by one or a few volunteers.
- Consequences
  - ☞ Slow response to security issues.
  - ☞ No security patches.
  - ☞ Project becomes "abandonware."

Example: The event-stream npm incident where a maintainer handed over control to a malicious actor.

License Compliance Risks

Not a Security Risk?

Incorrect OSS license compliance can lead to:

- ☞ Slow response to security issues.
- ☞ No security patches.
- ☞ Project becomes "abandonware."

Case Study - Heartbleed (OpenSSL) - 2014

- **Component**: OpenSSL cryptographic library.
- **Vulnerability**: Buffer over-read in the TLS heartbeat extension.
- **Impact**: Leak of up to 64KB of server memory, potentially containing private keys, passwords, and user data.
- **Root Cause**: A simple programming error in a critical, widely used library that lacked dedicated resources.

***Image source**: "Heartbleed Logo," Heartbleed, [Online]. Available: https://www.heartbleed.com/heartbleed.png. [Accessed: 07-Oct-2025].*

The OSS Security Lifecycle & Tools

The Three Pillars of Application Security (AppSec)

"White-box" testing of your source code

"Black-box" testing of your running application

Analyzing your dependencies for vulnerabilities and licenses

SAST (Static Application Security Testing)

DAST (Dynamic Application Security Testing)

SCA (Software Composition Analysis)

Phase 1: Discover (Create a Software Bill of Materials - SBOM)

What is an SBOM?

- A nested inventory of all software components.
- **Formats**: SPDX, CycloneDX, SWID.

Tools

- Dependency Track: For analyzing SBOMs.
- Built-in Scanners: npm audit, snyk test, docker scout.

Phase 2: Detect & Analyze Vulnerabilities

Process

- SCA tool scans your SBOM.
- Cross-references with vulnerability databases (NVD, GitHub Advisories).
- Provides a report with CVSS scores.

Key Metrics

- **CVSS Score**: Severity (Low, Medium, High, Critical).
- **EPSS Score**: Exploit Prediction Score (likelihood of active exploitation).

Phase 2: Detect & Analyze Vulnerabilities

Tools

- **Commercial**: Snyk, Mend (formerly WhiteSource), Black Duck, GitHub Advanced Security.
- **Open Source**: OWASP Dependency-Check, Trivy, Grype.
- **Integrated**: GitLab Dependency Scanning, GitHub Dependabot.

Phase 3: Prioritize & Triage

- You will have hundreds or thousands of vulnerabilities. You cannot fix them all at once.

- Prioritization Factors:
  - ☞ CVSS/EPSS Score.
  - ☞ Is the vulnerable function actually called? (Reachability Analysis).
  - ☞ Is there a public exploit (PoC)?
  - ☞ What is the asset's business criticality?

Phase 4: Remediate

- **Option 1 -** Update the Dependency. The best and simplest solution.
- **Option 2 -** Apply a Patch. If no update is available, consider back-porting a community patch.
- **Option 3 -** Mitigate. Configure the software to disable the vulnerable feature (e.g., a Log4J mitigation).
- **Option 4 -** Accept the Risk. Document the business reason for not fixing it.
- **Option 5 -** Replace/Remove. Find a more secure alternative.

Phase 5: Prevent & Harden the Pipeline

- Integrate security checks earlier in the SDLC.

Practice

- ☞ Block PRs that introduce new Critical vulnerabilities.
- ☞ Use signed commits and verified dependencies.
- ☞ Harden your CI/CD pipeline (e.g., use isolated, ephemeral builders).

Best Practices & Governance

Pillars of OSS Security Governance

People

Training, clear roles (who owns OSS risk?).

Process

Policies, approval workflows, exception handling.

Technology

The toolchain (SCA, SAST, DAST).

Best Practice 1: Establish an OSS Policy

- What it should include:

  - ☞ Approved and banned licenses.
  - ☞ Allowed/Prohibited package sources.
  - ☞ Maximum allowed vulnerability severity for new imports.
  - ☞ Mandatory SBOM generation.
  - ☞ Patching SLAs (e.g., Critical vulns must be patched in 72 hours).

Best Practice 2: Centralized Dependency Management

- Use a Proxy Repository Manager: JFrog Artifactory, Sonatype Nexus.
- Benefits:

  ☞ Cache dependencies for speed and availability.

  ☞ Control which external packages can be downloaded.

  ☞ Scan components upon ingestion.

  ☞ Block malicious or non-compliant components.

Best Practice 3: Automate Security in CI/CD

- A CI/CD pipeline diagram with security gates:

1. Pre-commit: Secret scanning.
2. Build: SCA, SAST.
3. Test: DAST.
4. Deploy: Final security scan; fail the build if Critical vulns are present.

Best Practice 4: Cultivate a Security-Aware Culture

- **Train Developers**: Secure coding, OSS risks, how to use security tools.
- **Incentivize Security**: Don't punish teams for finding vulns; reward them for fixing them quickly.
- **Share Responsibility**: Security is everyone's job, not just the "Security Team."
- **Speaker Notes**: "If developers see security as a blocker, they will work around it. If they see it as a helper, they will embrace it."

Best Practice 5: Participate in the OSS Community

- Give Back: Contribute security patches upstream. This fixes the problem for everyone, including you in the long run.
- Report Vulnerabilities Responsibly: Follow responsible disclosure practices.
- Sponsor Critical Projects: Fund the infrastructure or maintainers of projects you rely on.

Summary

- Open Source is Everywhere: It's the foundation of modern software, making its security a universal concern.
- Embrace the Paradox: Transparency is both a strength (community auditing) and a weakness (visible to attackers).
- Know What You Have: You can't secure what you don't know. Use SCA tools to create a Software Bill of Materials (SBOM).
- Manage the Lifecycle: Follow a continuous process: Discover -> Detect -> Prioritize -> Remediate -> Prevent.
- Build a Security Culture: People and processes are as important as technology. Train developers and make security a shared responsibility.
- Give Back to the Community: Contribute patches, report vulnerabilities responsibly, and sponsor projects you rely on.

Brain Teaser

1. The first and most critical phase in the OSS security lifecycle is:

A. Prioritize

B. Detect

C. Discover

D. Remediate.

Brain Teaser

1. The first and most critical phase in the OSS security lifecycle is:

A. Prioritize

B. Detect

C. Discover

D. Remediate.

Brain Teaser

2. A malicious package with the name "requwest" is uploaded to a public repository, hoping a developer will mistype the name of the popular "request" library. This is an example of:

A. Dependency Confusion

B. Typosquatting

C. Maintainer Burnout

D. Vendor Lock-in

Brain Teaser

2. A malicious package with the name "requwest" is uploaded to a public repository, hoping a developer will mistype the name of the popular "request" library. This is an example of:

A. Dependency Confusion

B. Typosquatting

C. Maintainer Burnout

D. Vendor Lock-in

Thank you!

"Open source software is a testament to the power of collaboration; it transforms ideas into innovations, empowering individuals and communities to build a better future together."

Lecturer: Biniam Behailu
Addis Ababa Science and Technology University