

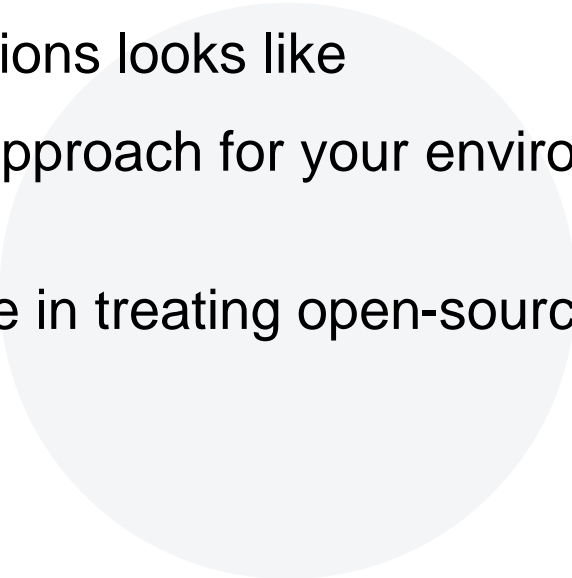


Navigating through the open-source security risks

Stanislav Sivák
May 27th, 2025



Goals

- Understand the open-source software risk complexity
 - Understand how solutions looks like
 - Review your current approach for your environment or prepare questions to find out more
 - Share your experience in treating open-source risks with us!
- 

Agenda

- Introduction
- Motivation
- Solution
- Approaches
- Q&A

Disclaimer:

“This is my personal presentation and represents neither my current employer nor any other organization.”

Introduction

- I started working in cybersecurity in 2007 and later switched to software security
- Worked in various organizations in Luxembourg and Germany before coming to Czech Republic
- Acted in various security roles: developer, engineer, tester, consultant, manager
- Currently being responsible for a software security champions program at a Czech bank

Open-source risks in 2024

1,658 projects scanned by Black Duck audits

97% of the codebases contained open source

70% of scanned code had its origin in open source

Vulnerabilities and Security



86%
of risk-assessed
codebases contained
vulnerable open source



81%
of risk-assessed
codebases contained
high- or critical-risk
vulnerabilities

**8 of the
top 10**
high-risk vulnerabilities
were found in jQuery

Components	Percentage of codebases containing the component
jQuery	32%
jQuery UI	16%
Bootstrap (Twitter)	15%
Spring Framework	12%
Lodash	12%
Netty Project	11%
jackson-databind	9%
Apache Tomcat	8%
Python programming language	5%
TensorFlow	1%

Source: [Black Duck OSSRA Report 2025](#)

Have you heard yet?

SECURITY NEWS

60 Malicious npm Packages Leak Network and Host Data in Active Malware Campaign

Socket's Threat Research Team has uncovered 60 npm packages using post-install scripts to silently exfiltrate hostnames, IP addresses, DNS servers, and user directories to a Discord-controlled endpoint.

May 2025

SECURITY NEWS

Typosquatted Go Packages Deliver Malware Loader Targeting Linux and macOS Systems

Malicious Go packages are impersonating popular libraries to install hidden loader malware on Linux and macOS, targeting developers with obfuscated payloads.

March 2025

Malicious npm Packages Attacking Linux Developers to Install SSH Backdoors

By Tushar Subhra Dutta - April 22, 2025

April 2025

Critical Flaw in Apache Parquet Allows Remote Attackers to Execute Arbitrary Code

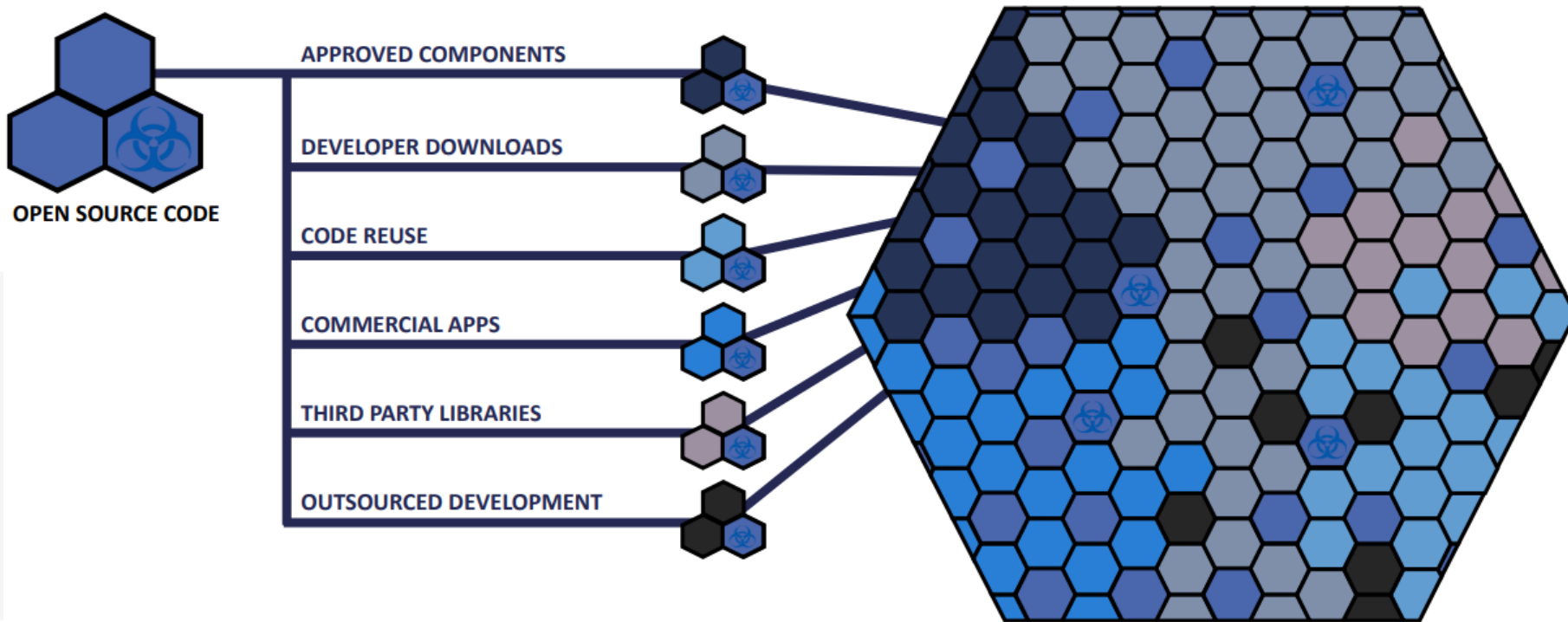
Apr 04, 2025 Ravie Lakshmanan

Vulnerability / Cloud Security

April 2025

Where it comes from?

Open source enters your code through many channels...



... and it also brings its specific risks

Challenges

1. Know my assets

- I need further information to our application inventory...
- How much open-source software do we use?
- How is the use of open-source software governed in our organisation?

JQuery 1.5	Angular v16	Struts 2.5.3
RubyZip 2.3.2	OpenSSL 3.0.0	RubyZip

Software Bill of Materials



CIO

2. Know my security risks

- Which our products have open-source vulnerabilities?
- Do we have any components with critical and high vulnerabilities?
- Which our projects have the XXX vulnerable component?
- How do we recognize **vulnerable** and/or **malicious** components?

OWASP TOP 10

A06:2021 Vulnerable and Outdated Components

CWE-937 OWASP Top 10 2013: Using Components with Known Vulnerabilities

CWE-1035 2017 Top 10 A9: Using Components with Known Vulnerabilities

CWE-1104 Use of Unmaintained Third Party Components



CISO/Security
Manager

3. Know my legal risks

- Are we legally allowed to deploy/distribute our software with its current open source?
- Do we have any open-source licenses non-compliant with our internal policy?
- Are any open-source licenses contradictory?

2017 - Artifex Software, Inc. versus Hancom, Inc.

Artifex Software

Hancom Inc.

- 1. Developed open-source PDF interpreter
- 2. The interpreter has a dual license: either GPL or commercial

- 3. Used the interpreter in the commercial Office software
- 4. Hancom neither paid for the commercial license nor published the custom software as open-source -> license infringement



US District Court

5. GPL can be treated like a legal contract



Compliance team

Vizio sued by nonprofit to share code for open-source software

By Blake Brittain

October 20, 2021 6:06 PM GMT+2 · Updated 3 years ago

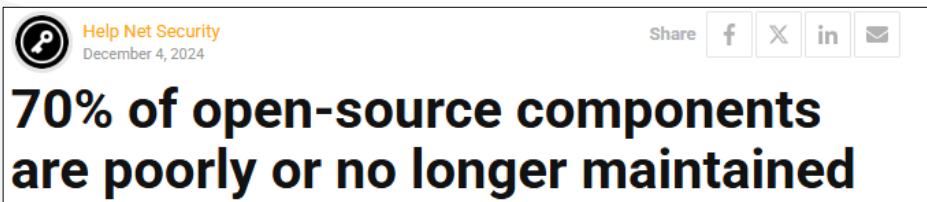


3. Know my legal risks

Licensing Scheme	License Family	Examples	Risk Level
Copyleft	Affero General Public License (AGPL)	<ul style="list-style-type: none">• GNU Affero General Public License v3 or later	Very High
Copyleft	Reciprocal	<ul style="list-style-type: none">• GNU General Public License (GPL) 2.0 or 3.0• Sun GPL with Classpath Exception v2.0	High
Copyleft	Weak reciprocal	<ul style="list-style-type: none">• Code Project Open License 1.02• Common Development and Distribution License (CDDL) 1.0 or 1.1• Eclipse Public License• GNU Lesser General Public License (LGPL) 2.1 or 3.0• Microsoft Reciprocal License Mozilla	Medium
Non-commercial use	Non-commercial	<ul style="list-style-type: none">• Aladdin Free Public License (AFPL)• Java Research License (JRL)	Very High
Copyright (©)	N/A		Very High

4. Operational risks

- How well is the component maintained by the community?
- Are security vulnerabilities/bugs fixed within tolerable time?
- What is our plan B if there is no new update?



Engineers

Solution

4 steps to the rescue



**1. Know Your
Software**

2. Identify Risks


3. Treat Risks

**4. Control Software
Release**



Know Your Software

Software Bill of Materials (SBoM)

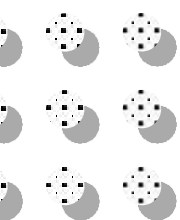
- Contains **vendor, component/dependency/module names** and **versions**
 - Provides relations between the components (direct vs transitive)
 - License information
 - Vulnerability metadata (optional)
- 



Know Your Software

Software Bill of Materials (SBoM) - Standards

- [CycloneDX](#) by OWASP and [SPDX](#) by Linux Foundation are two most acclaimed SBoM open standards
- They still evolve and have multiple flavours depending on the BoM purpose (security, compliance, disclosure)
- Machine-readable formats, commonly XML and JSON



Know Your Software

Software Bill of Materials (SBoM) – Minimal common fields

The following attributes represent the essential elements required in an SBOM:

- **SBOM meta-information:** Core data of the SBOM document itself, including author of the SBOM data, timestamp of its creation and the primary component being documented
- **Supplier name:** The entity that creates, defines and manufactures the software component
- **Component name:** The designated name of software component, assigned by the original supplier
- **Version of component:** Captures version-specific changes to accurately track updates and modifications
- **Other unique identifiers:** Includes reference types such as Common Platform Enumeration (CPE), SWID tags, or Package URLs (PURL) for precise component tracking
- **Cryptographic hash:** A unique fingerprint for each software component that enables verification of integrity and precise identification
- **Dependency relationship:** A structured map showing how components are interconnected, covering both direct and transitive dependencies
- **License information:** Documentation of legal terms for supplied software components
- **Copyright notice:** Entity holding exclusive and legal rights to the listed components

pkg:maven/org.apache.commons/io@1.3.4

Know Your Software

Software Bill of Materials (SBoM) - Standards

```
{
  "$schema": "https://cyclonedx.org/schema/bom-1.6.schema.json",
  "bomFormat": "CycloneDX",
  "specVersion": "1.6",
  "serialNumber": "urn:uuid:3e671687-395b-41f5-a30f-a58921a69b79",
  "version": 1,
  "metadata": {
    "timestamp": "2025-01-21T12:00:00Z",
    "component": {
      "bom-ref": "internal-web-app",
      "manufacturer": {
        "name": "Acme Inc"
      },
      "type": "application",
      "name": "Internal Web App",
      "version": "2.4.1",
      "description": "This is an example and serves as the affected first-party application in this example."
    }
  },
  "compositions": [
    {
      "aggregate": "complete",
      "vulnerabilities": [ "internal-web-app" ]
    }
  ],
  "vulnerabilities": [
    {
      "id": "INT-2025-002",
      "ratings": [
        {
          "source": {
            "name": "Security Research Company"
          },
          "score": 6.3,
          "severity": "medium",
          "method": "CVSSv31",
          "vector": "AV:N/AC:L/PR:L/UI:R/S:U/C:L/I:H/A:N"
        },
        {
          "source": {
            "name": "Security Research Company"
          }
        }
      ]
    }
  ]
}
```

CycloneDX 1.6 SBoM sample

```
1 {
2   "SPDXID" : "SPDXRef-DOCUMENT",
3   "spdxVersion" : "SPDX-2.3",
4   "creationInfo" : {
5     "comment" : "This package has been shipped in source and binary form.\n\nThe binaries were created with gcc 4.5.1 and expect to link to\ncompatible",
6     "created" : "2010-01-29T18:30:22Z",
7     "creators" : [ "Tool: LicenseFind-1.0", "Organization: ExampleCodeInspect ()", "Person: Jane Doe ()" ],
8     "licenseListVersion" : "3.17"
9   },
10  "name" : "SPDX-Tools-v2.0",
11  "dataLicense" : "CC0-1.0",
12  "comment" : "This document was created using SPDX 2.0 using licenses from the web site.",
13  "externalDocumentRefs" : [ {
14    "externalDocumentId" : "DocumentRef-spdx-tool-1.2",
15    "checksum" : {
16      "algorithm" : "SHA1",
17      "checksumValue" : "d6a770ba38583ed4bb4525bd96e50461655d2759"
18    },
19    "spdxDocument" : "http://spdx.org/spdxdocs/spdx-tools-v1.2-3F2504E0-4F89-41D3-9A0C-0305E82C3301"
20  } ],
21  "hasExtractedLicensingInfos" : [ {
22    "licenseId" : "LicenseRef-1",
23    "extractedText" : "/*\n * (c) Copyright 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009 Hewlett-Packard Development Company, LP\n * All",
24  }, {
25    "licenseId" : "LicenseRef-2",
26    "extractedText" : "This package includes the GRDDL parser developed by Hewlett Packard under the following license:\n\nCopyright 2007 Hewlett-Pack",
27  }, {
28    "licenseId" : "LicenseRef-4",
29    "extractedText" : "/*\n * (c) Copyright 2009 University of Bristol\n * All rights reserved.\n * \n * Redistribution and use in source and binary fo",
30  }, {
31    "licenseId" : "LicenseRef-Beerware-4.2",
32    "comment" : "The beerware license has a couple of other standard variants.",
33    "extractedText" : "\"THE BEER-WARE LICENSE\" (Revision 42):\n\nnphk@FreeBSD.ORG wrote this file. As long as you retain this notice you\ncan do whate",
34    "name" : "Beer-Ware License (Version 42)",
35    "seeAlsos" : [ "http://people.freebsd.org/~phk/" ]
36  } ],
37}
```

SPDX 2.3 SBoM sample

Identify Risks

SCA (Software Composition Analysis)

Identifies the list of open/third-party components in your software (source-code or binary)

Identifies licensing (compliance) risks

SCA

Identifies security risks and provide the list of known vulnerabilities including remediation advice

Identifies operational risks

Treat Risks

Treating SBoM risks

- Remove components which are not needed at all (without breaking anything)

Treating security risks

- Assess attack vectors and code reachability
- Inputs: Threat model, source code, knowledge base

Treating license risks

- Evaluate/remove components with non-compliant licenses
- Inputs:
 - Open-source compliance policy
 - Product context

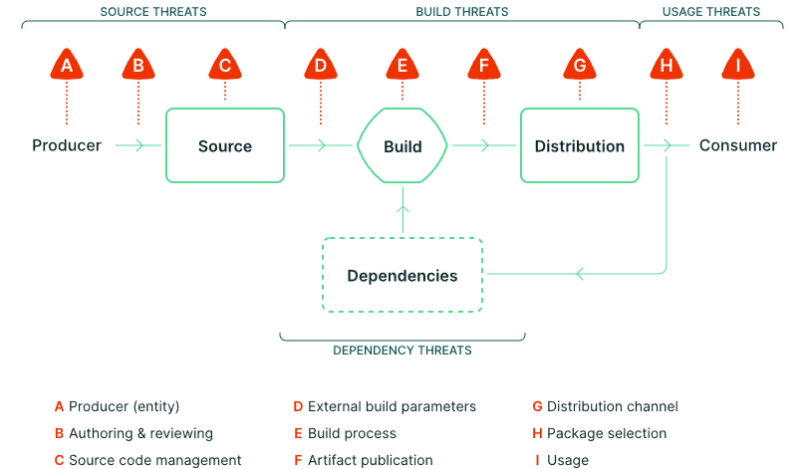
Treating operational risks

- Evaluate/remove non-maintained components
- Inputs: Third-party use guidelines and component monitoring

Control Software Release

Be in control of the following gates in the SDLC:

- Source code versioning
- Software signing and verification
- Binary artifactory pushes
- Third-party (Supplier) software integrity
- Open-source software consumption
- Production software deployment and monitoring



Source: [SLSA](#)

Beyond Open-Source Security: Supply Chain Attacks

- A fascinating and complex security attack vector
- Between 2019 and 2022, software supply chain attacks skyrocketed by an astounding 742%
- Securing your product is not enough, you also need to secure its pre-production environment (Dev, Test, PreProd)

What is SLSA?

Supply-chain Levels for Software Artifacts, or SLSA ("salsa"), is a set of incrementally adoptable guidelines for supply chain security, established by industry consensus. The specification set by SLSA is useful for both software producers and consumers: producers can follow SLSA's guidelines to make their software supply chain more secure, and consumers can use SLSA to make decisions about whether to trust a software package.

SLSA offers:

- A common vocabulary to talk about software supply chain security
- A way to secure your incoming supply chain by evaluating the trustworthiness of the artifacts you consume
- An actionable checklist to improve your own software's security
- A way to measure your efforts toward compliance with the [Secure Software Development Framework \(SSDF\)](#)

Track/Level	Requirements	Focus
Build L0	(none)	(n/a)
Build L1	Provenance showing how the package was built	Mistakes, documentation
Build L2	Signed provenance, generated by a hosted build platform	Tampering after the build
Build L3	Hardened build platform	Tampering during the build

Approaches

Common manual approaches

MANUAL DISCOVERY

- Cumbersome processes
- Occurs at end of SDLC
- High effort and low accuracy
- No ongoing controls

SPREADSHEET INVENTORY

- Requires consistent developer input
- Difficult to maintain and scale
- Not a full/accurate list of actual usage

SPORADIC VULNERABILITY TRACKING

- No single responsible entity
- Labor intensive manual effort
- Unmanageable (~11 new vulns/day)

PERIODIC VULNERABILITY SCANNING

- Monthly/quarterly vulnerability assessments
- Not aimed at open source vulnerabilities
- Integrated later in the SDLC

#FAIL

Common automated approaches

Static Application Security Testing (SAST)

SAST

- Analyzes any source code, not only FOSS specific
- Finds common vulnerability patterns such as:
 - SQL injection
 - Cross-site scripting
 - Buffer overflows, etc.

Advantages

- Finds some publicly known and unknown security vulnerabilities in the source code
- No additional tool/testing stage needed
- SAST can be performed in various pipeline stages
- SAST tools can have a separate module that inspects software composition

Disadvantages

- Limited insight into Software Composition Analysis
- No Software Bill of Material
- No licensing information

Common automated approaches

Dynamic Application Security Testing (DAST) + penetration testing + vulnerability scanning

Dynamic testing

- Tests running apps automatically (DAST/vulnerability scanning) and manually (pentest)
- Finds flaws, misconfigurations and vulnerabilities in apps

Advantages

- Finds both publicly known and unknown security vulnerabilities
- No additional tool/testing stage needed
- Fewer false positives than SAST because it focuses on exposed components

Disadvantages

- Less comprehensive than repository scans as it examines running software from outside
- Runs later in a later pipeline stage
- Very incomplete Bill of Material
- No licensing information
- Results represent a point in time

Common automated approaches

Source Code Repository Checks

- Built-in functionality
- Runs regularly
- Creates pull requests

Advantages

- Examines open-source components automatically
- No triggered scan needed
- Seamless integration
- Often easy remediation in the repository via pull request
- Continuous monitoring

Disadvantages

- Focus on dependencies but no code snippets or modified files/directories
- Basic license compliance
- Could miss relevant or catch non-relevant dependencies which would not be deployed into the product release
- Project-level (developer-friendly) view only

Common automated approaches

Source Code Repository Checks - Example

Command Injection in Xstream #36

Open Opened yesterday on com.thoughtworks.xstream:xstream (Maven) · pom.xml

Upgrade com.thoughtworks.xstream:xstream to fix 37 Dependabot alerts in pom.xml

Upgrade com.thoughtworks.xstream:xstream to version 1.4.21 or later. For example:

```
<dependency>
<groupId>com.thoughtworks.xstream</groupId>
<artifactId>xstream</artifactId>
<version>[1.4.21,</version>
</dependency>
```

Create Dependabot security update

Package	Affected versions	Patched version
com.thoughtworks.xstream:xstream (Maven)	< 1.4.7	1.4.7

Xstream API versions up to 1.4.6 and version 1.4.10, if the security framework has not been initialized, may allow a remote attacker to run arbitrary shell commands by manipulating the processed input stream when unmarshaling XML or any supported format. e.g. JSON.

Severity: **Critical** 9.8 / 10

CVSS v3 base metrics

Attack vector	Network
Attack complexity	Low
Privileges required	None
User interaction	None
Scope	Unchanged
Confidentiality	High
Integrity	High
Availability	High

Learn more about base metrics

CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/CH/IH/A:H

EPSS score: 6.686% (90th percentile)

GitHub Dependabot

Dependency graph

Dependencies Dependents Dependabot **Export SBOM**

Automatically detect additional dependencies for Maven

[Enable automatic dependency submission](#) to automatically identify and submit transitive dependencies to GitHub, so you can receive Dependabot alerts for known vulnerabilities in them.

Search all dependencies

90 Total Ecosystem

com.thoughtworks.xstream:xstream 1.4.5	2 critical
Maven · pom.xml · Detected automatically on Apr 01, 2025 · NOASSERTION	
org.bitbucket.b_c:jose4j 0.9.3	1 moderate
Maven · pom.xml · Detected automatically on Apr 01, 2025 · Apache-2.0	
actions/checkout 4.1.6	
GitHub Actions · .github/workflows/build.yml · Detected automatically on Apr 01, 2025	
actions/setup-java 4.*.*	
GitHub Actions · .github/workflows/build.yml · Detected automatically on Apr 01, 2025	

Common automated approaches

Binary Repository Checks

- Built-in functionality
- Triggered for each new artifact in the repository

Advantages

- Could be one catch-for-all: Examines all binary components known for open-source vulnerabilities before deployment including container images
- Easy access to artifacts
- Can be triggered on-demand or automatically when new artifacts appear
- Easy implementation of approved artifacts only (due to licensing, whitelisting,...)
- Easy integration + Continuous analysis

Disadvantages

- Coverage is not always that strong for the compiled code (C/C++ in particular)
- Basic license compliance information

Common automated approaches

Binary Repository Checks - Examples

The screenshot shows the Xray JFrog interface. On the left, a sidebar lists various categories like Policy Violations, SBOM, Security Issues, Vulnerabilities, Malicious Packages, Secrets, Services, Applications, Descendants, and Ancestors. The main panel displays a list of 179 vulnerabilities. The selected vulnerability, CVE-2020-1747, is shown in detail on the right. It is a Critical issue with a CVSS Score of 9.8 (x3). The description states: "Insufficient input validation in the PyYAML library allows unauthenticated network attackers to perform code execution when parsing a crafted YAML file." The remediation path is to upgrade from version 5.2 to 5.3.1. A "Full screen (f)" button is visible at the bottom right of the details panel.

Severity	ID
Critical	CVE-2020-1747
Critical	CVE-2021-35042
Critical	CVE-2020-14343
Critical	XRAY-94986
Critical	CVE-2018-20060
Critical	CVE-2022-41902
Critical	CVE-2022-41910
Critical	CVE-2022-41880
Critical	CVE-2022-41900

CVE-2020-1747
jFrog research last updated on 31 Dec 9:47 PM
This CVE is enriched by jFrog research and provides more accurate information

Xray ID: XRAY-95701

Contextual Analysis: **APPLICABLE**

jFrog Severity: **Critical**

Component: PyYAML

Version: 5.2

Upgrade to: 5.3.1

CVSS Score: 9.8 (x3)

CWE: CWE-20

Show Less

jFrog Research | Contextual Analysis | Public Sources | Impact Paths | References

Summary

Insufficient input validation in the PyYAML library allows unauthenticated network attackers to perform code execution when parsing a crafted YAML file.

Remediation

Details

jFrog Research Severity Reasons

Xray JFrog

The screenshot shows the Sonatype Lifecycle interface for the component org.xmlunit:xmlunit-core:2.9.1. It is identified as a Maven Transitive Dependency. The vulnerability section shows a Highest CVSS Score of 9.2. Under Disclosed Vulnerabilities, a table lists CVE-2024-31573 with a status of "Sonatype Verified" and an analysis state of "Not Affected".

org.xmlunit:xmlunit-core:2.9.1

SBOM Webinar | Demo Video | SBOM 2024-10-09 10:04:47

Maven | Transitive Dependency

pkg:maven/org.xmlunit/xmlunit-core@2.9.1?type=jar

Vulnerability

Component Summary

Highest CVSS Score: 9.2

Vulnerabilities Verified: 1 Sonatype Verified, 0 Unverified

Disclosed Vulnerabilities

Existing vulnerabilities disclosed by the originator of this SBOM.

CVSS SCORE	ISSUE	VERIFIED STATUS	ANALYSIS STATE	JUSTIFICATION	ACTIONS
9.2	CVE-2024-31573	Sonatype Verified	Not Affected	Code not present	

Additional Sonatype Identified Vulnerabilities

Sonatype Lifecycle

Common automated approaches

SCA standalone


- Designed for open-source scanning
- Provides Bill of Material
- Can have dedicated database vulnerabilities
- Monitors for new vulnerabilities
- Some solutions can find copied code snippets

Advantages

- Detailed information on open-source risks
- Few false positives due to several ways of identifying open-source components
- Both compiled and uncompiled code can be analysed
- Usually faster in scanning FOSS components than SAST/DAST
- Can detect code snippets
- Most solutions offer monitoring purposes

Disadvantages

- Creates overhead by implementing another testing step
- Does not find publicly unknown vulnerabilities, so need to be complemented with SAST/DAST



DEPENDENCY-CHECK

Dependency-Check is an open source tool performing a best effort analysis of 3rd party dependencies; false positives and false negatives may exist in the analysis performed by the tool. Use of the tool and the reporting provided constitutes acceptance for use in an AS IS condition, and there are NO warranties, implied or otherwise, with regard to the analysis or its use. Any use of the tool and the reporting provided is at the user's discretion. Copyright holder or OWASP be held liable for any damages whatsoever arising out of or in connection with the use of this tool, the analysis performed, or the resulting report.

[How to read the report](#) | [Suppressing false positives](#) | [Getting Help: github issues](#)

[Sponsor](#)

Project:

Scan Information ([show all](#)):

- dependency-check version: 12.1.1
- Report Generated On: Mon, 26 May 2025 21:31:24 +0200
- Dependencies Scanned: 318 (280 unique)
- Vulnerable Dependencies: 20
- Vulnerabilities Found: 94
- Vulnerabilities Suppressed: 0
- ...

Summary

Display: [Showing Vulnerable Dependencies \(click to show all\)](#)

Dependency	Vulnerability IDs	Package	Highest Severity	CVE Count	Confidence	Evidence Count
webgoat.jar:bootstrap-5.3.3.jar		pkg:maven/org.webjars/bootstrap@5.3.3	MEDIUM	1		20
webgoat.jar:bootstrap.min.js		pkg:javascript/bootstrap@3.1.1	MEDIUM	9		3
webgoat.jar:bootstrap.min.js		pkg:javascript/bootstrap@3.1.1	MEDIUM	9		3
webgoat.jar:jwt-0.9.1.jar	cpe:2.3:a:json_web_token_project:json_web_token:0.9.1:***** cpe:2.3:a:web_project:web:0.9.1:*****	pkg:maven/io.jsonwebtoken/jwt@0.9.1	MEDIUM	1	High	26
webgoat.jar:jwt-0.9.3.jar	cpe:2.3:a:jwt_project:jwt:0.9.3:*****	pkg:maven/org.bitbucket.b_c/jose4j@0.9.3	MEDIUM	1	Highest	38
webgoat.jar:jquery-1.10.2.min.js		pkg:javascript/jquery@1.10.2.min	MEDIUM*	5		3
webgoat.jar:jquery-2.1.4.min.js		pkg:javascript/jquery@2.1.4.min	MEDIUM*	5		3
webgoat.jar:jquery-ui-1.10.4.custom.min.js		pkg:javascript/jquery-ui-dialong@1.10.4	MEDIUM	5		5

-

Common automated approaches

SCA standalone testing – OWASP tooling

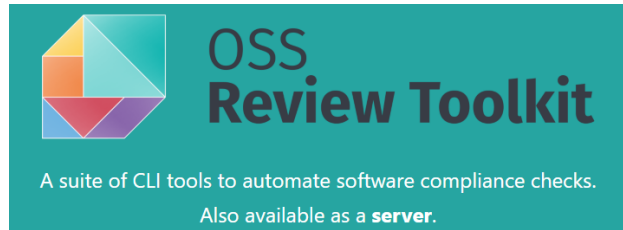


- Acts as a platform
- Allows for monitoring
- Powerful vulnerability intelligence:
 - Integrates with the EPSS (Exploit Prediction Scoring System) vulnerability model
 - In addition to NVD information (vulnerabilities), takes input from OSV (Open-source vulnerability) database (malicious pgs), GitHub Security Advisories and VulnDB

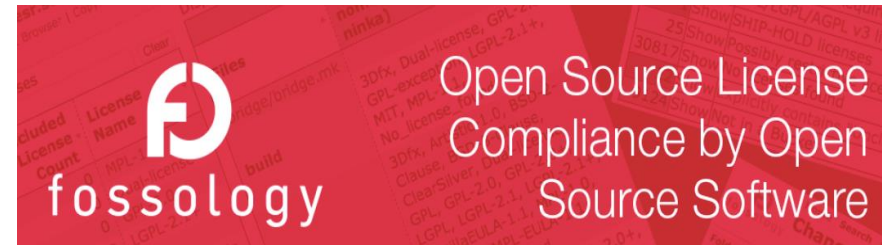
OWASP Dependency Track

Common automated approaches

Open-source license compliance tool examples



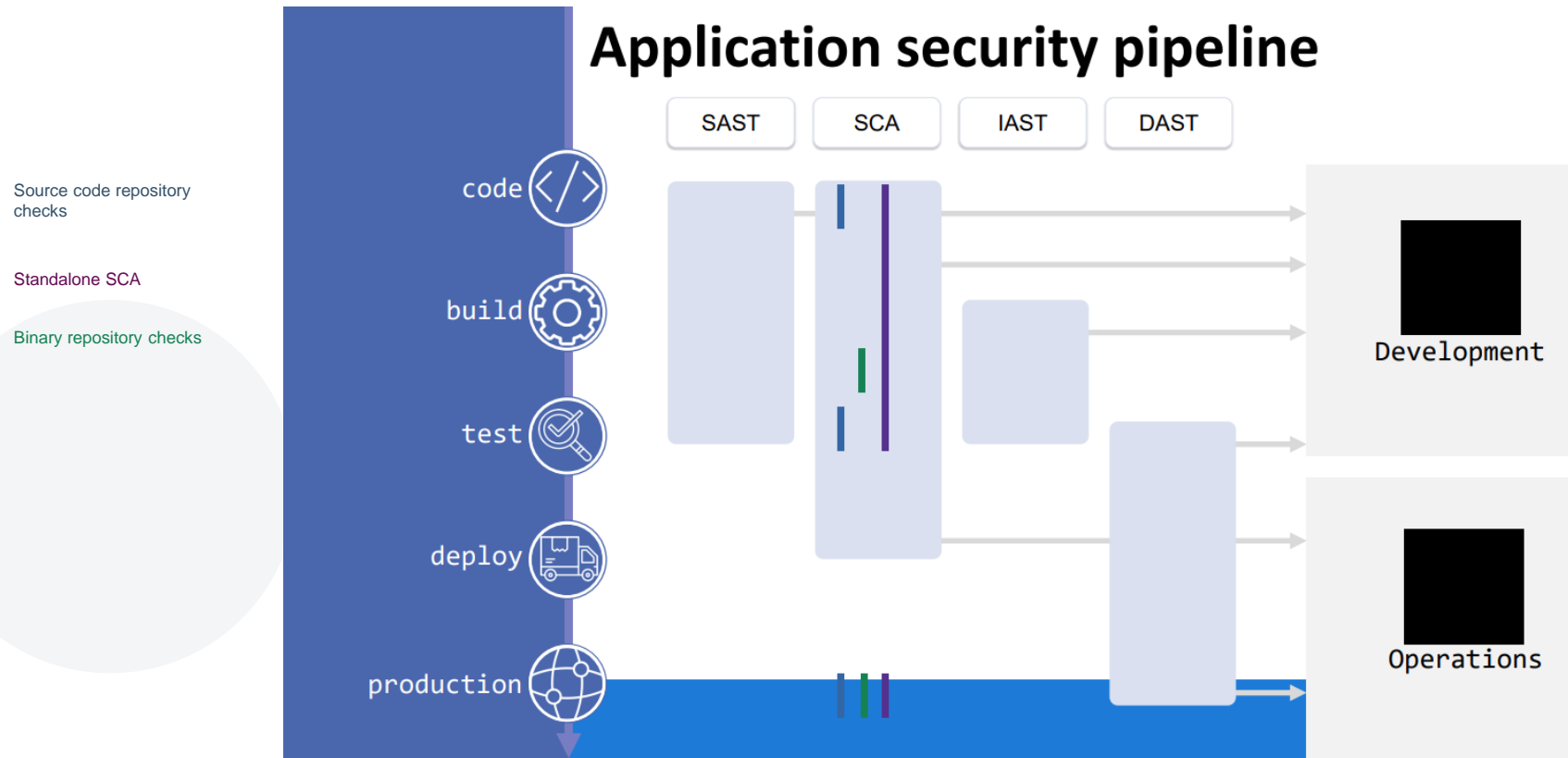
[OSS Review Toolkit](#)




[Fossology](#)

Common automated approaches

Integration





Q&A

Thanks for your attention!