SBOM & VEX
Getting lulled into a false sense of security

Henrik Plate (Endor Labs)
Feb 2024

About me

Main interests:

- **Detection, assessment and mitigation of known open source vulns**

  Co-author of Eclipse Steady and Project KB

- **Classification & detection of supply chain attacks**

  Co-author of Backstabber's Knife Collection and Risk Explorer

- **Change Impact Analysis and Debloating**

Henrik Plate
Security Researcher (Endor Labs)

Previously at SAP
> 10 years on OSS security

Email henrik@endor.ai
LinkedIn henrikplate
Google Scholar

Outline

- Accurate SBOM and VEX documents require linking applications, components versions, vulnerabilities and vulnerable code
- Structured overview about problems that make those links brittle and weak
- Exemplified by vulnerability information from OSV *

My goals for today: Make you

- take SBOMs and VEX documents with grains of salt,
- choose the right apps for tool evaluations, and
- ask the right questions to SBOM/VEX tool providers.

Vulnerability Exploitability eXchange (VEX)

VEX documents are distributed as part of an SBOM document, or separately

They assert the [status] of a [product_id] with respect to [vul_id] [1]

- Status MUST be one of [under investigation, not affected, affected, fixed]

For status “not affected”, a VEX statement must provide an impact statement (free text) or a justification with 5 possible values [1, p.10]:

- Component not present, Vulnerable code not present, Vulnerable code not in execute path, Vulnerable code cannot be controlled by adversary, Inline mitigations already exist

CycloneDX schema offers nine possible values for justification [2], e.g.

- Code not present, Code not reachable or Protected at perimeter

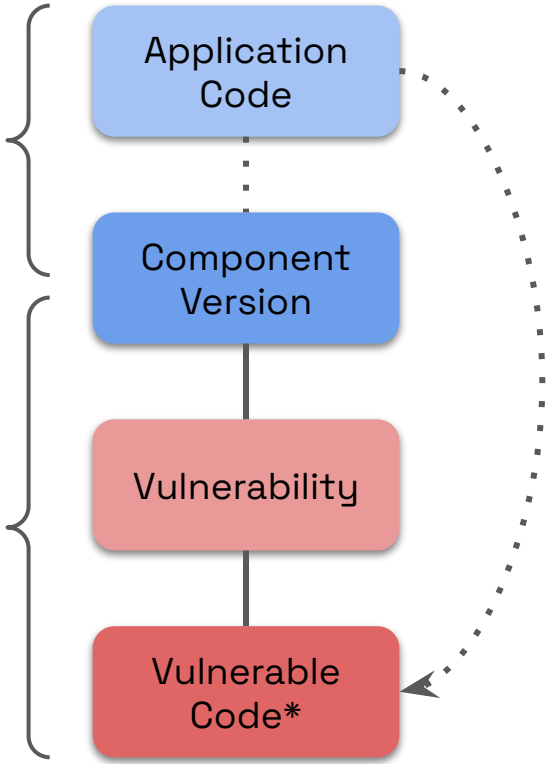
[1] CISA: Minimum Requirements for Vulnerability Exploitability eXchange (VEX) (2023)
[2] CycloneDX 1.5: https://cyclonedx.org/docs/1.5/json/#vulnerabilities_items_analysis_justification

How to answer those questions?

Through manifest files or other means

Public* and Private Vulnerability Databases

(ex. NVD, OSV**)

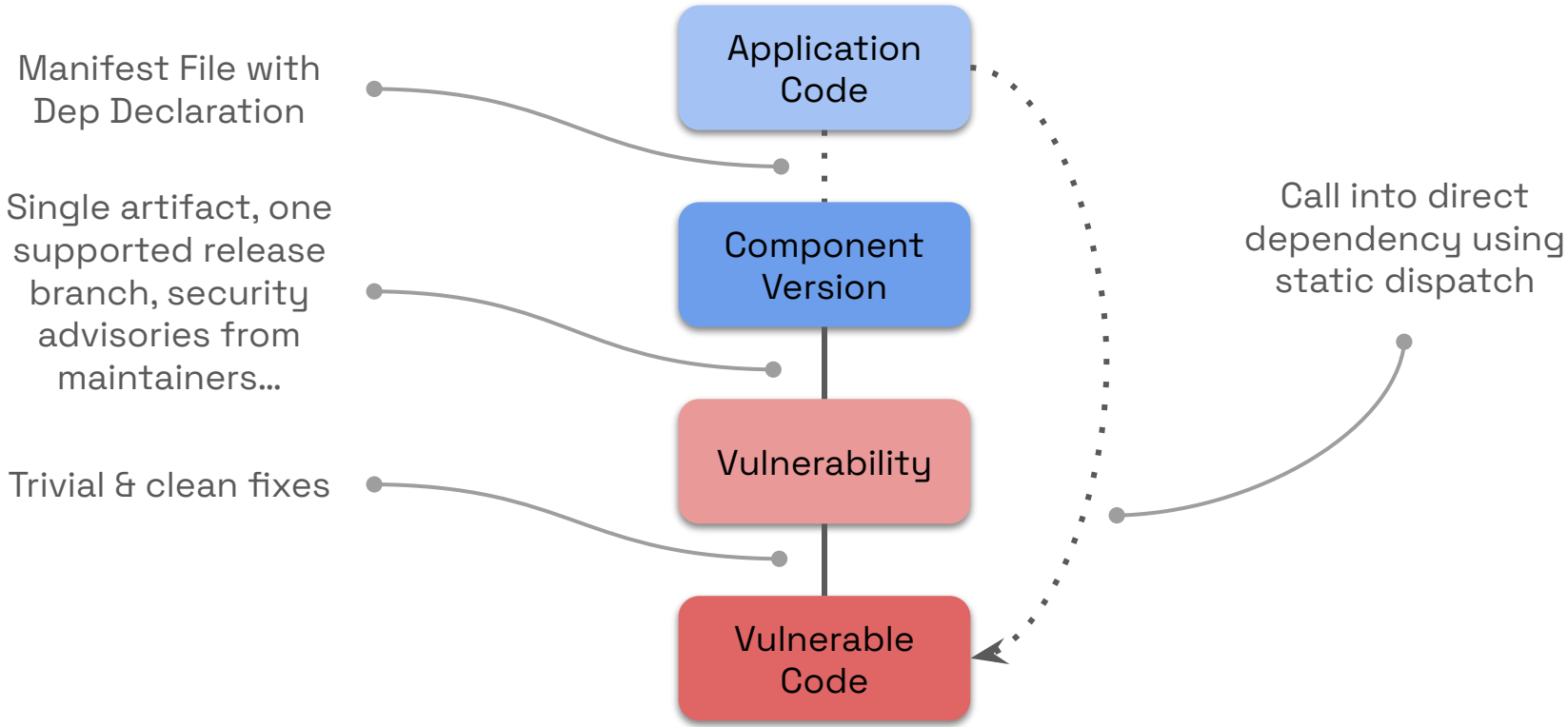


Is there any vulnerable code?

Can it run in my app context?

Can it be exploited in my app context?

The Happy Path



The Happy Path

https://litfl.com/wp-content/uploads/2020/10/streetlight-effect.jpg

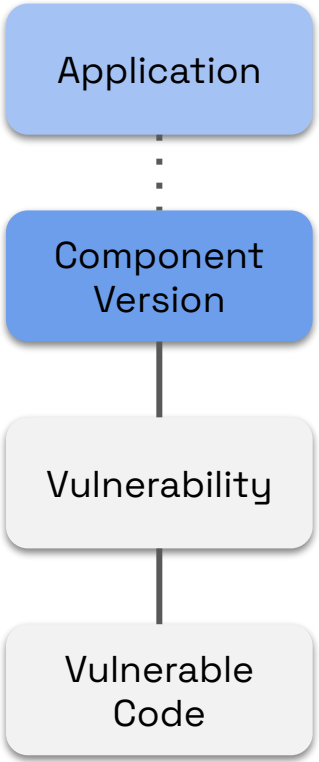
Phantom Dependencies

Problem: Manifest files are just one out of many ways to establish dependencies .

Examples:

- Manual or scripted installation through pip, brew or apt-get (comparable to provided deps in the Maven world)
- Dynamic installation à la try-except-install

```
if strategy_name.lower() == "sigopt":
    try:
        import yaml  # flake8: noqa
    except ImportError:
        if sys.version_info.major == 2:
            subprocess.check_call(['apt-get', 'install', '-y', 'python-yaml'])
        else:
            subprocess.check_call(['apt-get', 'install', '-y', 'python3-yaml'])
        import yaml  # flake8: noqa
```

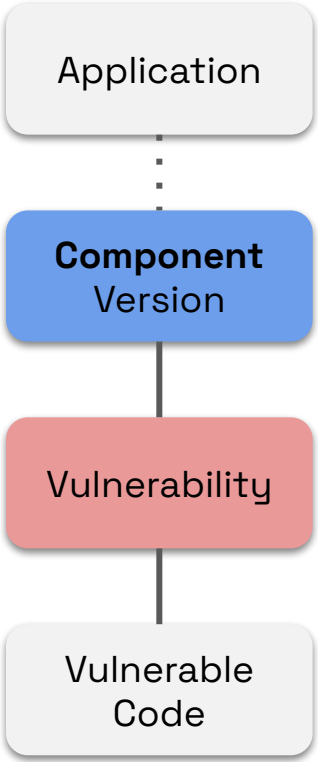


Name-changes

Problems: Project renaming, forking and “exotic” distribution channels hinder the tracking of vulnerable code and the enumeration of all affected artifact identifiers.

Example: CVE-2022-1279 in EBICS Java Client

- Originally on SourceForge, continued, renamed and forked on GH
- Components with vulnerable code have 3 different Maven GAs:
  - `org.kopi:ebics` (when building from the sources in ebics-java/ebics-java-client)
  - `com.github.ebics-java:ebics-java-client` (when consuming the JAR from JitPack)
  - `io.github.element36-io:ebics-cli` (from a fork, deployed on Maven Central, not fixed)
- OSV marks the GitHub repo ebics-java/ebics-java-client as affected, but no Maven GAV



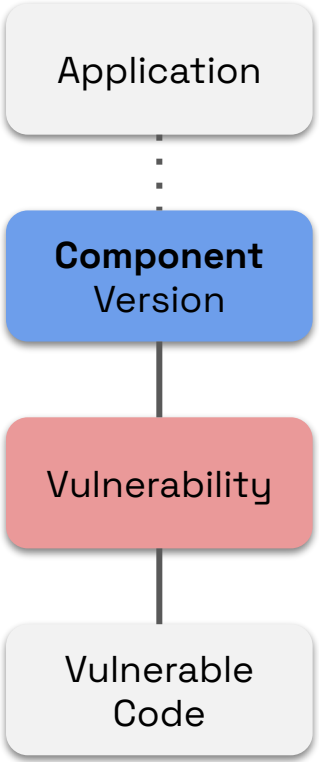
Multi-module Projects

Problem:

- Many projects produce multiple artifacts with different registry identifiers, and vulnerable code may be part of multiple ones.

Examples:

1. CVE-2023-33202 for Bouncycastle crypto library
   - 84 artifacts with groupId org.bouncycastle on Central
   - OSV marks 2 as affected, but the vulnerable class(es) are contained in 28 artifacts
2. CVE-2023-36566 in Microsoft Common Data Model SDK
   - 4 ecosystems supported from 1 GitHub repo, all affected
   - OSV marks Maven, PyPI and NuGet (but not npm)



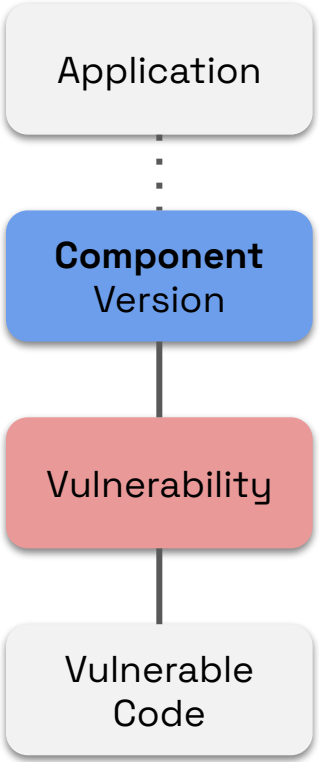
Multi-module Projects & Rebundling

Problems:

- Many artifacts comprise code from other projects.

Examples:

1. CVE-2018-1270 in Spring Framework
   - Fixed with e0de91 in DefaultSubscriptionRegistry
   - Comprised in 1 of 58 Spring artifacts: `org.springframework:spring-messaging`
   - OSV marks `org.springframework:spring-core` as affected
   - Class also rebundled in `org.apache.servicemix.bundles:org.apache.servicemix.bundles.spring-messaging`



Rebundling in Java

Background: groupId, artifactId, and version identify an artifact on Central
Example: `org.apache.logging.log4j` : `log4j-core` : `2.15.0`

- Study [1]: Search for rebundles of **254 known-vulnerable classes** from 38 components.

	Recompiled	**Uber-JAR**	**Uber-JAR (w/o meta)**	**Repackaged**
# rebundled classes	143 / 254	222 / 254	222 / 254	17 / 254
# distinct GAVs on Central	5,919	36,609	24,500	168
# distinct GAs	360	6,728	3,882	89

- Study [2]: 297 GAVs on Maven Central rebundle vulnerable log4j-core classes

[1] A Dann, et al.: Identifying Challenges for OSS Vulnerability Scanners - A Study & Test Suite (2021)
[2] https://github.com/CodeShield-Security/Log4JShell-Bytecode-Detector

Rebundling in Python

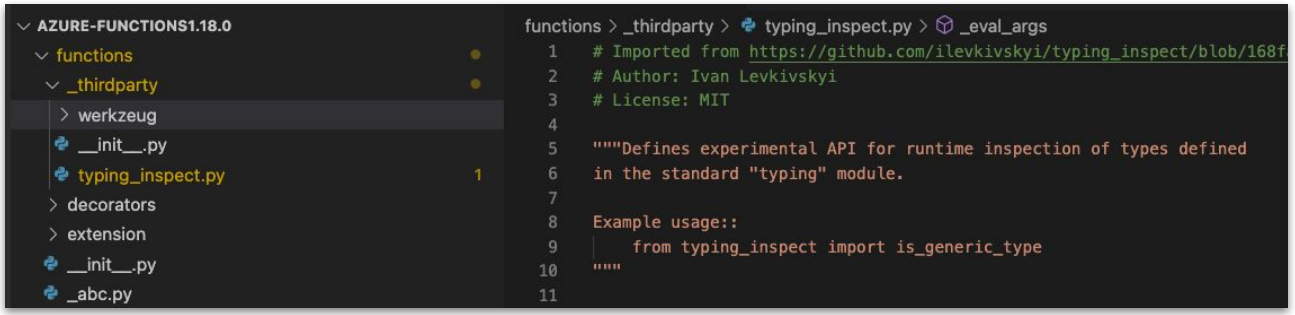
Examples:

1. CVE-2023-4863 in libwebp (WebP image codec)
   - Rebundled in 50 Python packages [1]
   - OSV covers 6
2. azure-functions 1.18.0
   - Rebundles werkzeug and a single Python file from GitHub

Top rebundled binaries in PyPI [1]

Bundled Library	Name	Min Projects
libgcc_s.so.X	GCC Runtime	920
libgomp.so.X	GNU OpenMP	747
libstdc++.so.X	GNU C++	527
libz.so.X	zlib	487
libgfortran.so.X	libgfortran	374
libquadmath.so.X	GCC Quad Precision Math	372
libcrypto.so.X / libssl.so.X	OpenSSL (or others)	341
liblzma.so.X	Xz Utils	235
libbz2.so.X	Bzip2	200
libselinux.so.X	SE Linux	189

Rebundled code in azure-functions 1.18.0



[1] Seth Larson: Patching the libwebp vulnerability across the Python ecosystem (2023)

Rebundling in JavaScript [1]

```
1  (window.webpackJsonp=window.webpackJsonp||[]).push([[80],{
2  maj8:function(e,t,n){"use strict";/* object-assign, (c) Sindre Sorhus, @license MIT */ var r=Object.
       getOwnPropertySymbols,i=Object.prototype.hasOwnProperty,o=Object.prototype.propertyIsEnumerable;function a(e){if(
       null==e)throw new TypeError("Object.assign cannot be called with null or undefined");return Object(e)}...},
3  ZK3j:f
4  xy6B:f
5  });
```

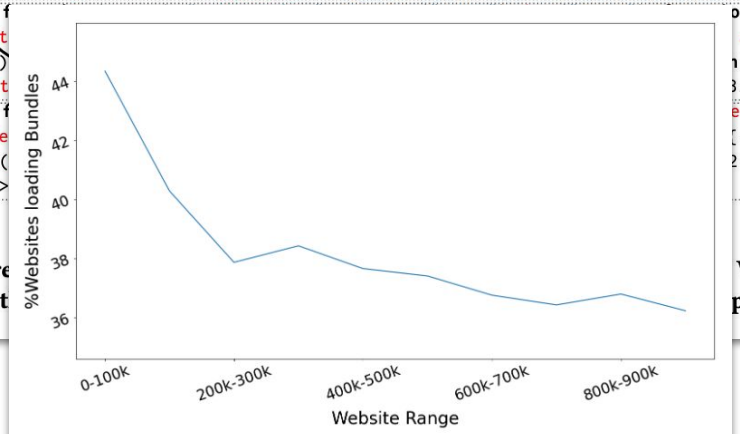


Figure 3: The prevalence of bundles

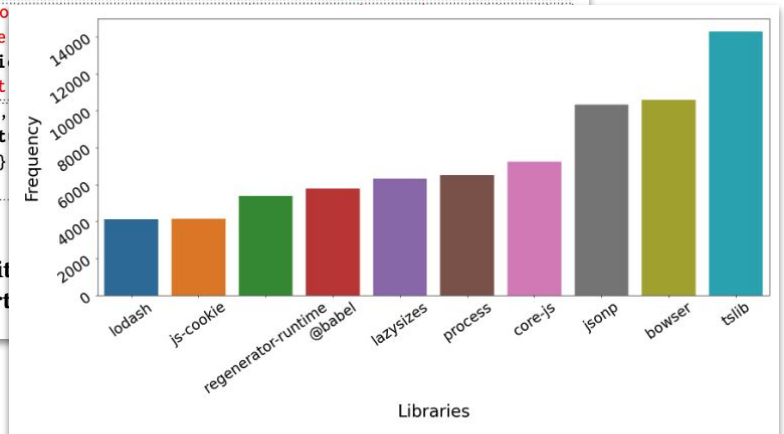


Figure 13: Top10 bundled libraries

[1] J Rack, et al.: Jack-in-the-box: An Empirical Study of JavaScript Bundling on the Web and its Security Implications (CCS, 2023)

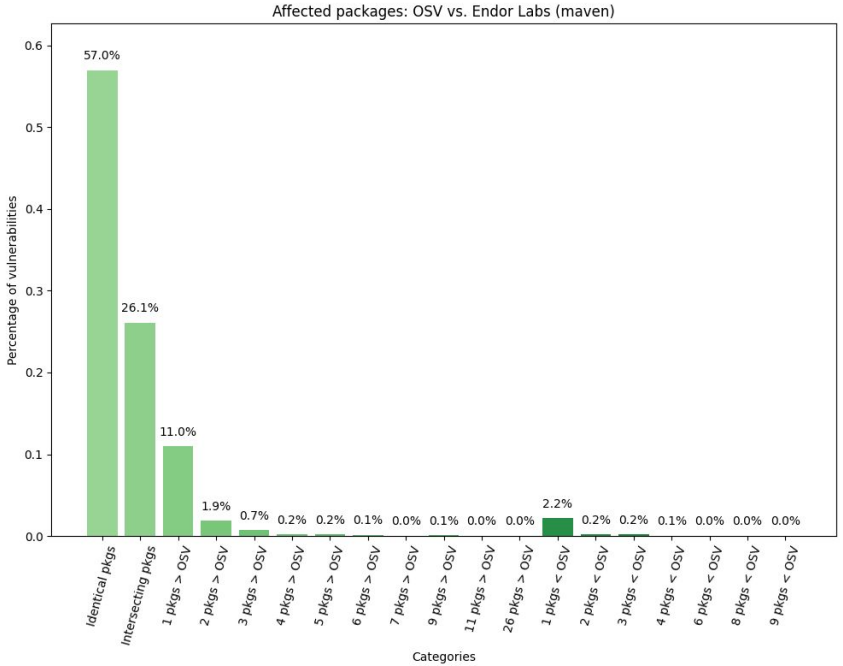
Component Confusion Stats

For Maven, OSV and Endor Labs …

- Agree for 57% of vulns on affected components (groupId:artifactId)
- Differ for 43% of vulns

Differences lead to FPs and FNs:

- For 11%, Endor Labs marks one additional GA as affected
- For 2%, OSV marks one additional GA as affected

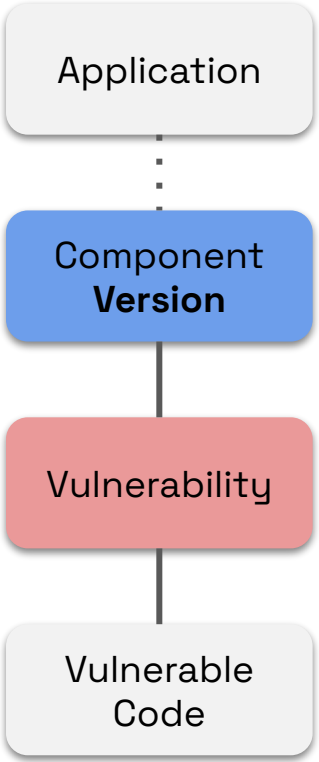


Confusion of Affected Versions

Problems: Identifying affected versions is mostly manual work, not done by project maintainers for EOL versions, and error-prone due to communication mishaps.

Examples:

1. CVE-2023-41080 in Apache Tomcat
   - 8.0.x reached EOL → not checked or fixed by project maintainers
   - The vulnerable function exists as-is since 5.5.23
   - OSV marks releases as of 8.5.x as affected
2. CVE-2023-50164 in Apache Struts
   - Official advisory marks EOL versions 2.0.0 - 2.3.7 as affected
   - Vulnerable function did not exist, but exploit worked as-is
   - OSV marked 2.5.0 and later



High-touch Maintenance

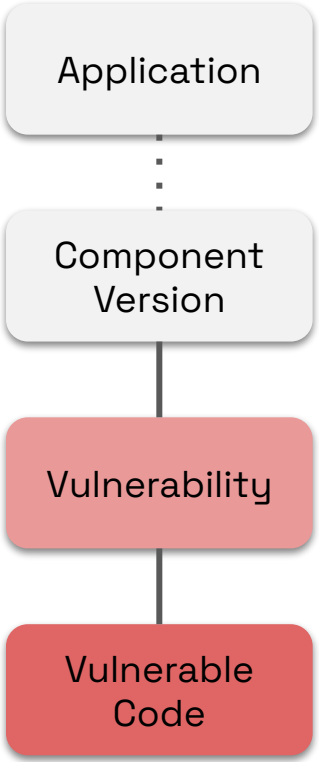
Dec 27, 2023	CVE-2023-51080 in Hutool, published to NVD and GHAD (unreviewed)
Jan 3, 2024	Reviewed & published by GHAD (affected: 0 - 5.8.23, fixed: 5.8.24)
Jan 5 → 9	1st correction submitted & merged (affected: 5.8.22 - 5.8.24, fixed: none)
Jan 11	Hutool v5.8.25 released on Central → **Fixed or not, how do tools interpret GHAD/OSV data?**
Jan 30 → 31	2nd correction submitted & merged (affected: 5.8.22 - 5.8.24, fixed: 5.8.25)

Non-trivial Fix Commits

Problem: The identification of vulnerable code is difficult if fixes comprise many commits, potentially for different release branches, and if they are “polluted” with unrelated changes.

Example: CVE-2020-35662 in SaltStack Salt

- 18 fix commits
- 14 functions modified to validate SSL certs

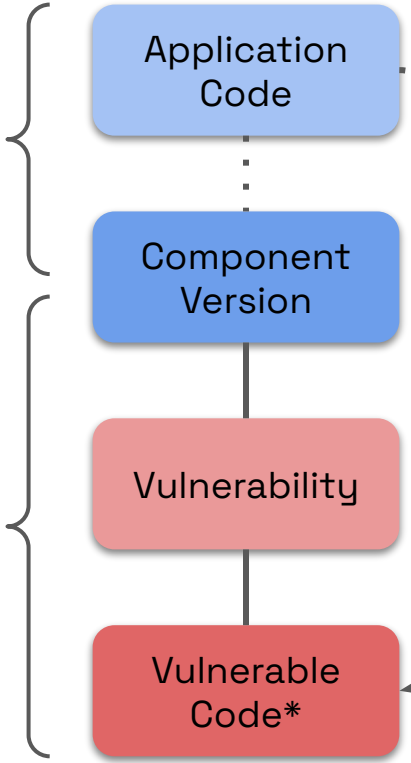


Cabinet Of Challenges

(without any claim to completeness)

- **Phantom dependencies** (not established through manifest files)
- **Vendored Code *** (copied into own repo)

- **Component Confusion** (e.g., forks, multi-module projects, name changes upon distribution, rebundling)
- **Confusion of Affected Versions**
- **Non-trivial Fix Commits**
- **Different naming schemes and granularities** ** (eg. CPE and GAV)



- **Reflection, eval etc.**
- **Inversion of Control**
- **Configuration vulnerabilities**
- **Cross-language calls**
- **Megamorphic call sites (dynamic dispatch w/ base types)**

Take-aways

Status-quo

- Brittle links between apps and vulnerable code
- High quality vulnerability databases require significant manual work

Opportunities:

- Comprehensive, code-level open-source vulnerability database
- Reliable way to identify vulnerable **code**, no matter where it is contained

When talking to your local SCA dealer:

- Do not (only) choose happy-path apps for product evaluations
- Ask for implementation details & statistics regarding the **3 critical areas** (dependency identification, vuln. database, and reachability analysis)

Thank you!

Email henrik@endor.ai
LinkedIn henrikplate

ENDOR LABS