Prioritization of SCA Findings in Software Dependencies

Using Static Reachability Analysis

22-JAN-2025

Joseph Hejderup

ENDOR LABS

Joseph Hejderup

Member of Technical Staff
(Researcher)

Main Interests

- Scalable Program Analysis Techniques
- Empirical Research on Software Supply Chains

Previously

- PhD researcher at Delft University of Technology (TU Delft), Netherlands

/joseph.hejderup

Morning News

QZ Quartz

How one programmer broke the internet by deleting a tiny piece of code

Lots of npm packages, however, relied on left-pad to do it for them, which is how this tiny bit of code became so important. Some of the largest, ...

Mar 27, 2016

WIRED

A Second SolarWinds Hack Deepens Third-Party Software Fears

It's been more than two months since revelations that alleged Russia-backed hackers broke into the IT management firm SolarWinds and used ...

5 days ago

SecurityWeek

Two Years On, Log4Shell Vulnerability Still Being Exploited to Deploy Malware

Two Years On, Log4Shell Vulnerability Still Being Exploited to Deploy Malware. More than two years after the Log4j crisis, organizations are...

Aug 22, 2024

From Morning News to Incident Management

Where do we begin?

- Which projects/systems use this library?
- Do this new exploit pose a risk to our organization?
- What are the short-term/long-term remediation steps?
- What resources are required to remediate this?

Inventory Building

Do we even use this library in our organization?

- Classic `pkgmgr build dependency:tree` per project
- Simple grep or search
- SBOM (Software Bills of Material) generator
- **Challenge:** Building an overview is not straightforward, teams tend to use different versions of the same library

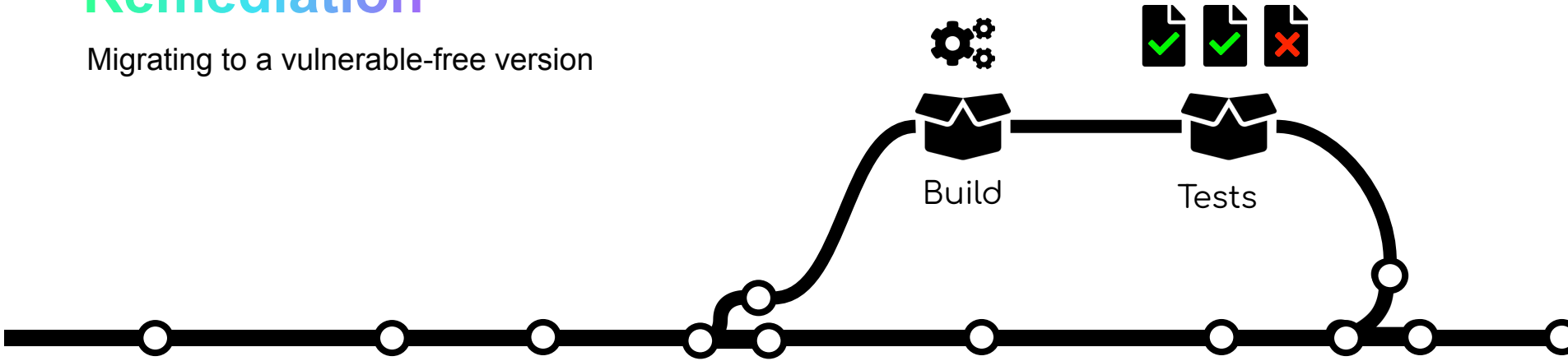
Risk Management

We are using this library but are we affected?

- Read the CVE/Security Advisory
  - Understanding affected versions, impact, and the vulnerability
- Asses the usage of the vulnerable library
  - Grep or scanning in source code
- Compile and assessment of how it impacts the project/organization
- **Challenge:** Manual effort, domain knowledge, and transitive dependencies

Remediation

Migrating to a vulnerable-free version

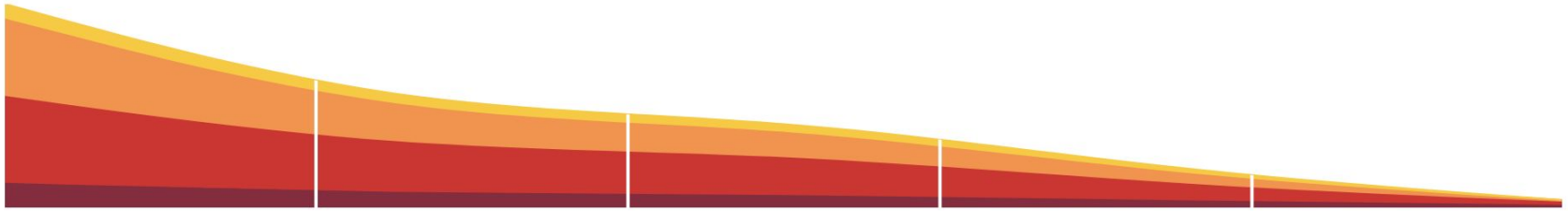


- It compiled™
- Automated dependency updating
- **Challenge:** trade-offs between versions

Supply Chain Vulnerabilties

Software Composition Analysis (SCA)

3.34K	1.6K	1.51K	591	125
Total Open Vulnerabilities	Not In Test	Fix Available		



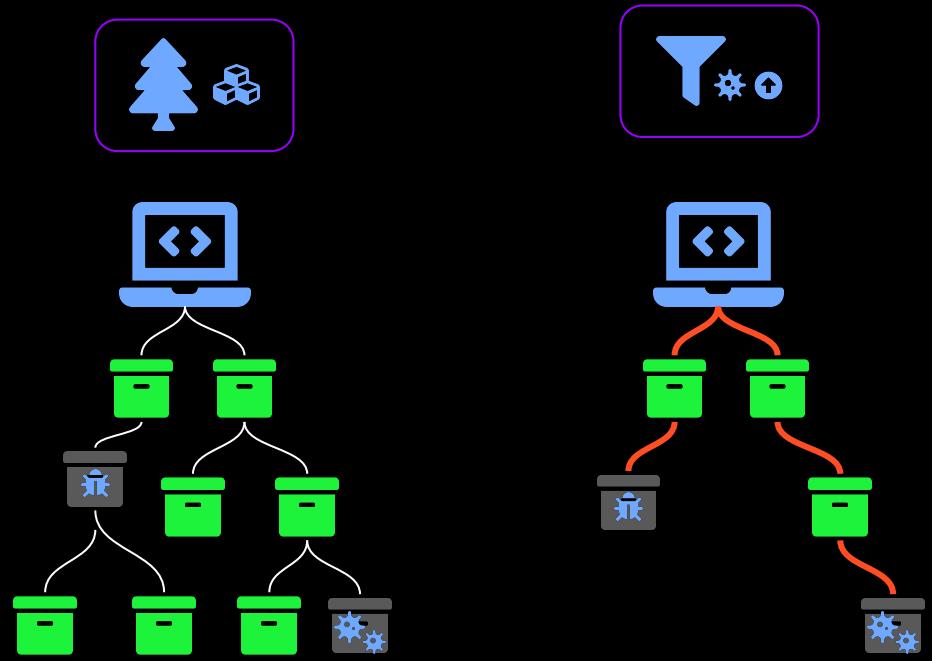
Getting noisy….

Traditional Methods

Analysis on Package & Build Manifests

```
[package]
name = “mypackage”
version = "4.16.3"


[dependencies]
accepts = "~1.3.5"
body-parser = "1.18.2"
depd = "~1.1.2"
encodeurl = "1.0.2"
escape-html = "~1.0.3"
etag = "1.8.1"
proxy-addr = "~2.0.3"
```



Traditional Methods

What is the problem?

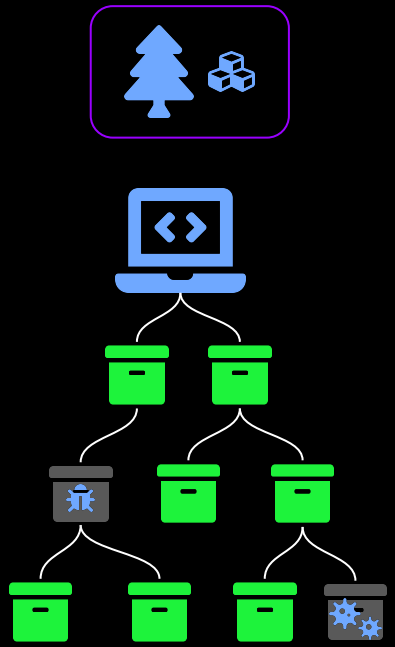
alert fatigue

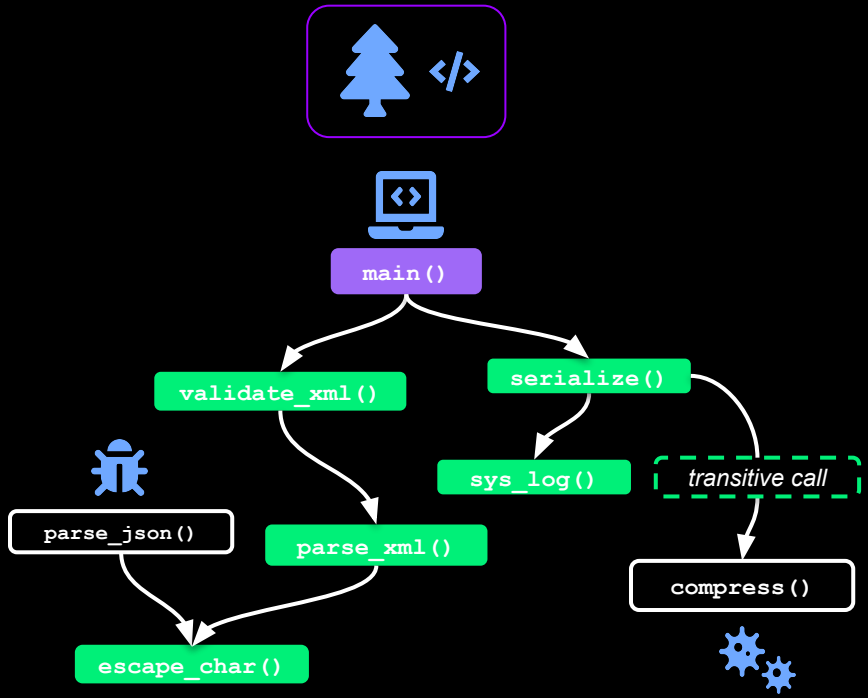
zero code analysis

declaration ≠ usage

Static Analysis - Call Graphs

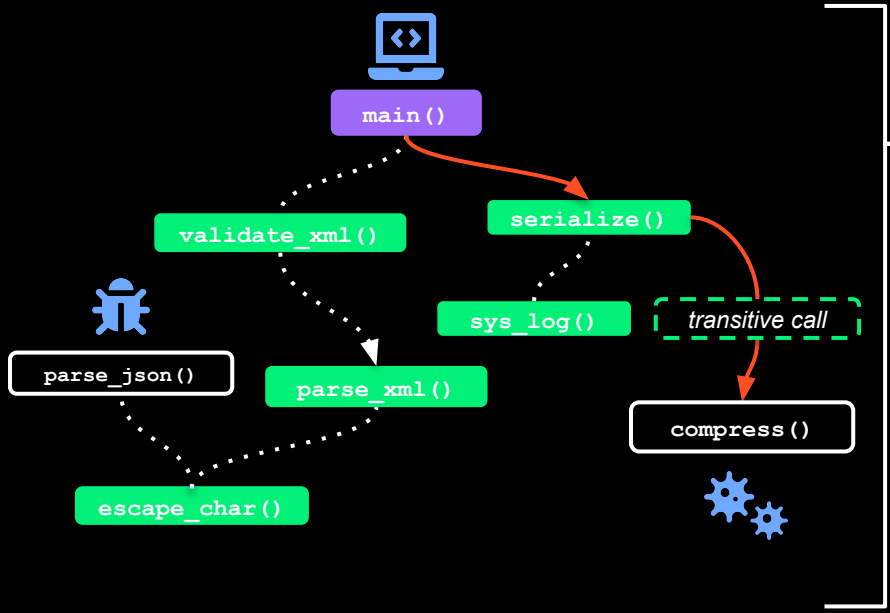
Moving to source code analysis

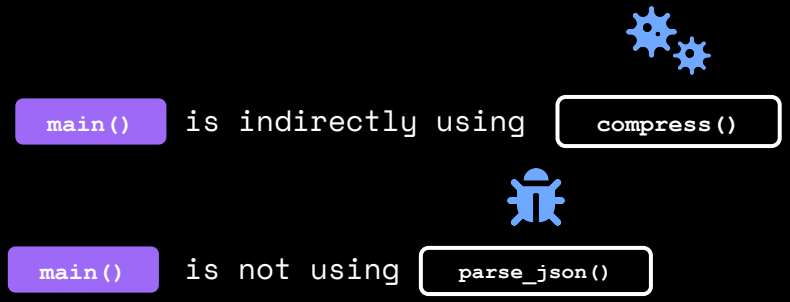




Reachability Analysis

Enables fine-grained prioritization





Result: 1 of 2 affects our app

Empirical Study

Rust crates.io

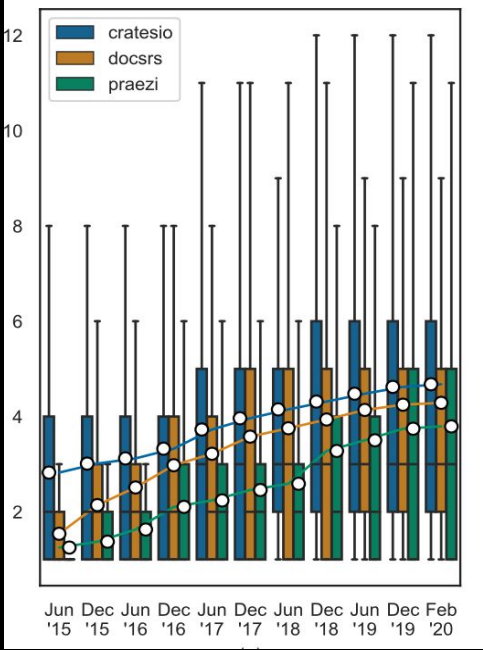
Inventory count

three representations
(traditional, compiled, and source)

Hejderup, J., Beller, M., Triantafyllou, K., & Gousios, G. (2022). Präzi: from package-based to call-based dependency networks. *Empirical Software Engineering*, *27*(5), 1-42.

Static Analysis - Inventory Analysis

Manifest vs Code - Direct Dependencies



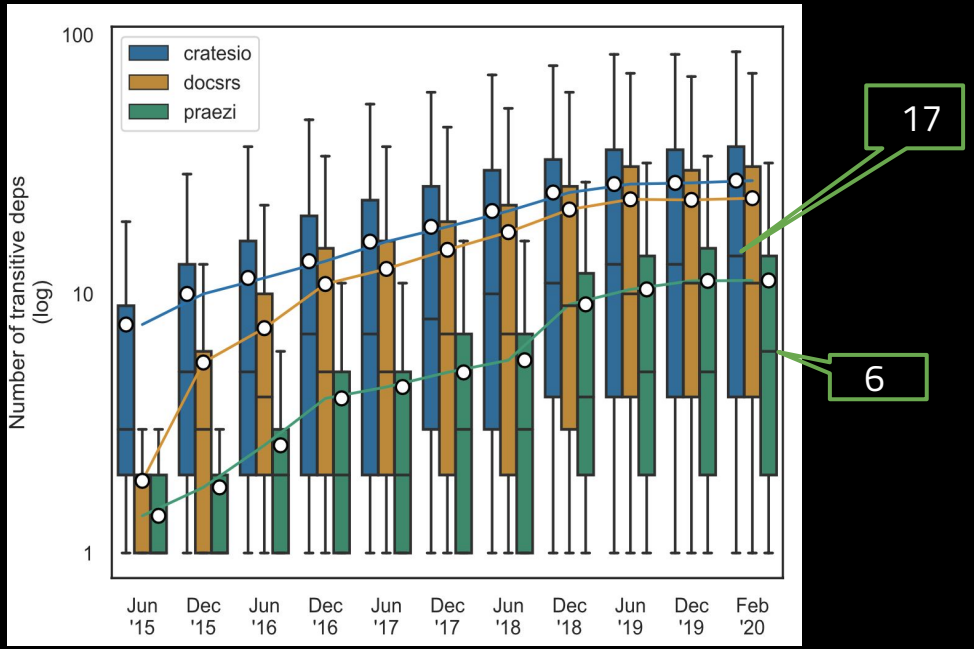
The median is similar; the mean relatively when comparing the three representations.

A manifest-inferred direct deps slightly over-approx. a static analysis inferred direct deps!

Hejderup, J., Beller, M., Triantafyllou, K., & Gousios, G. (2022). Präzi: from package-based to call-based dependency networks. *Empirical Software Engineering*, *27*(5), 1-42.

Static Analysis - Inventory Analysis

Manifest vs Code - Transitive Dependencies



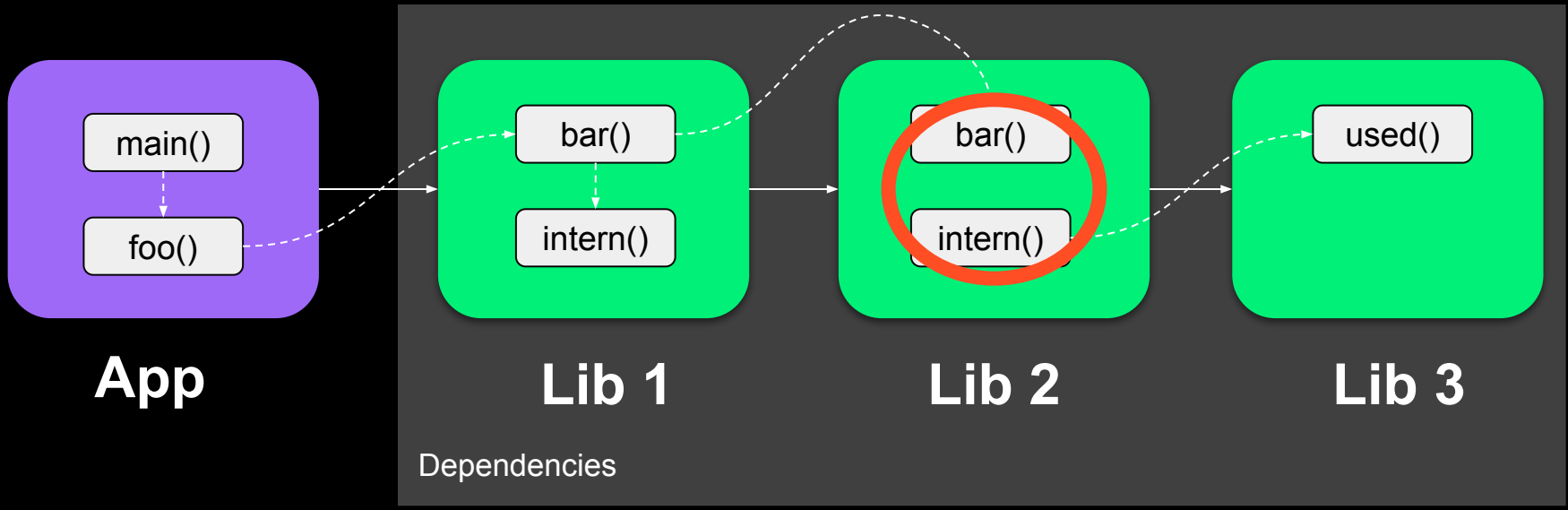
No close approximation of each other!

Mean: packages are indirectly not calling **60% of its resolved transitive dependencies**

Hejderup, J., Beller, M., Triantafyllou, K., & Gousios, G. (2022). Präzi: from package-based to call-based dependency networks. *Empirical Software Engineering*, *27*(5), 1-42.

Reachability Analysis

Why do inter-procedural analysis?



Implications - Direct Dependencies

- A declared dependency closely estimate a utilized dependency
- Manifest Analysis > Static Analysis

Pros:

- No need to implement expensive static analysis
- Techniques almost interchangeable

Cons:

- Insensitive to import statements or specific APIs
- Cannot detect libraries that does only code-gen

Implications - Transitive Dependencies

- Static Analysis > Manifest Analysis

Pros:

- Capturing actual usage, reducing false positives
- Better actionability

Cons:

- False Negatives is a risk and hard to detect
- Highly dynamic libraries (e.g., Java's reflection) are challenging to analyze

Implications - Transitive Dependencies

- Static Analysis > Manifest Analysis

Pros:

- Capturing actual usage, reducing false positives
- Better actionability

Cons:

- False Negatives is a risk and hard to detect
- Highly dynamic libraries (e.g., Java's reflection) are challenging to analyze

What about dynamic analysis?

Also other techniques than call graphs?

- Dynamic analysis (such as test execution) is precise but depends highly on coverage
- Configuration & setup with agents can be a hassle and introduce friction
- Dataflow hard to scale

Not all code analysis are the same

Why simply not do AST analysis? Many reasons! One is type propagation

```
public class Main {

    public static void printArea(Shape shape) {
        System.out.println("Area: " + shape.area());
    }

    public static void main(String[] args) {
        Shape circle = new Circle(5);
        printArea(circle);
    }
}
```

```
interface Shape {
    double area();
}
```

```
class Circle implements Shape {
    private double radius;

    public Circle(double radius) { this.radius = radius; }

    @Override
    public double area() { return Math.PI * radius * radius; }
}
```

Static Analysis - Phantom Dependencies

Manifest analysis = incomplete SBOMs

```
import subprocess
import sys

# Command to install pandas
command = [sys.executable, "-m", "pip", "install", "pandas"]

# Execute the command
subprocess.run(command, check=True)
```

Remediation

Automated Dependency Updates

1. Do we even write tests against dependencies in the first place?

2. Do project test suites even cover usages of dependencies in the source code?

3. Are tests sufficient alone for detecting bad updates?

Empirical Study

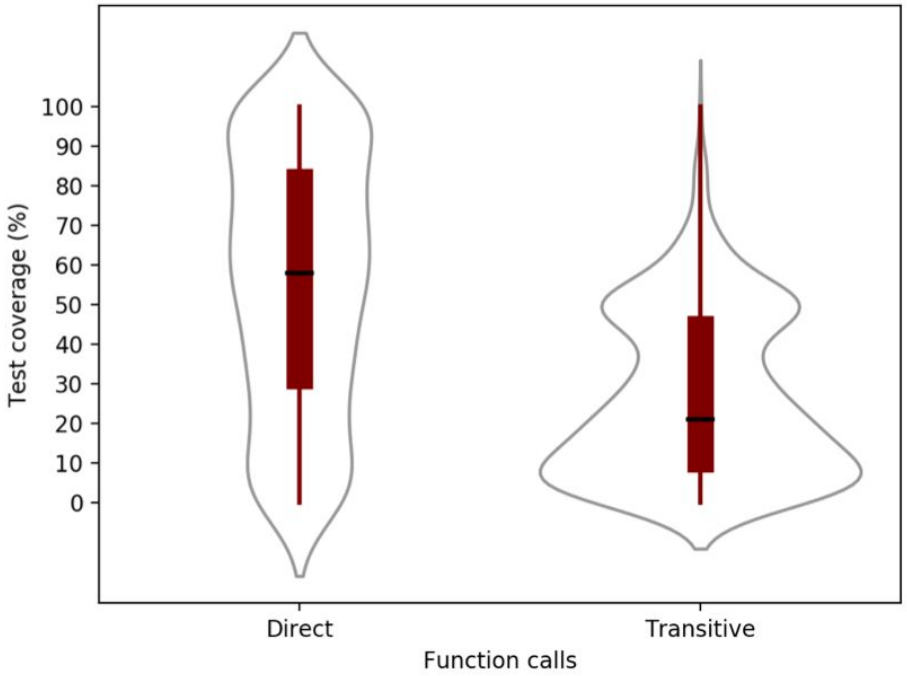
- What is the **statement coverage** of **function calls** to dependencies?
- How **effective** are test suites in detecting updates with **regression errors**?
- How does **static analysis complement/compare** to test suites in updating dependencies?

Static Analysis - Remediation

Are tests reliable?

60%

median coverage of **direct dependencies**



20% median coverage of **transitive dependencies**

Updates on untested code!

Hejderup, Joseph, and Georgios Gousios. "Can we trust tests to automate dependency updates? a case study of java projects." *Journal of Systems and Software* 183 (2022): 111097

Does this matter at all?

National Cyber Security Centre

ABOUT NCSC | CISP | REPORT AN INCIDENT | CONTACT US

Home | Information for... | Advice & guidance | Education & skills | Products & services | News, blogs, events...

NEWS

Alert: Apache Log4j vulnerabilities

The NCSC is advising organisations to take steps to mitigate the Apache Log4j vulnerabilities.

Download / Print Article PDF

Share

PUBLISHED

10 December 2021

WRITTEN FOR

Large organisations

Public sector

Cyber security professionals

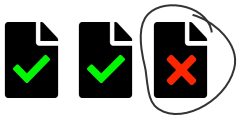
Measuring Test Effectiveness

Mutation testing

```
def add(x,y):
    return x + y
```

Arithmetic Mutation

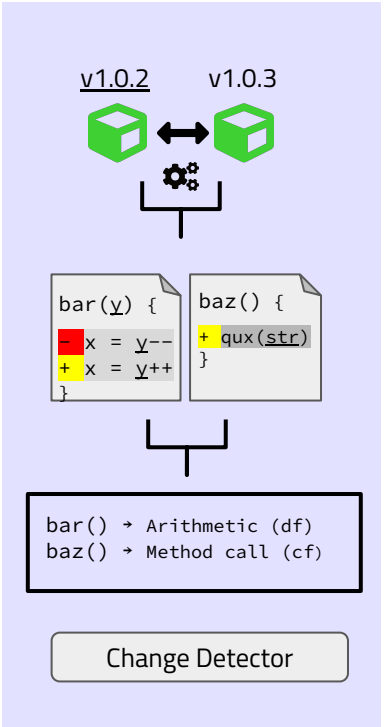
```
def add(x,y):
    return x - y
```

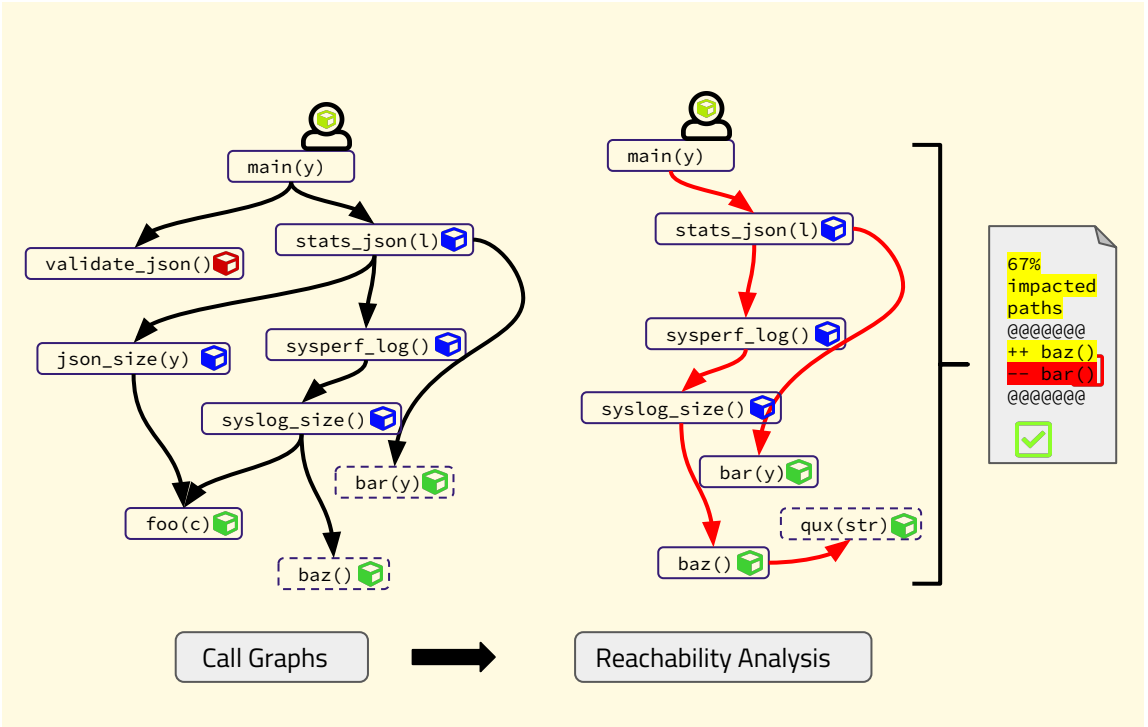


We use PITest with a twist: We don't mutate all dependency functions; only those reachable by tests!

Static Analysis - Remediation

Are tests reliable?

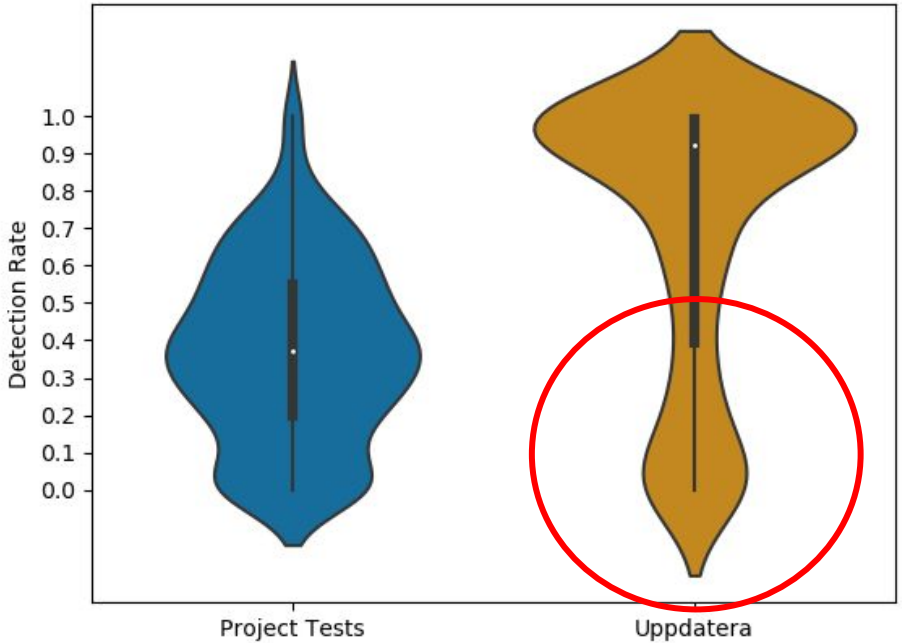




jhejderup/Uppdatera

Measuring Test Effectiveness

A million artificial updates on 262 GH Updates



On average,

37% **detected by tests!**

72% **detected by Uppdatera!**

No guarantees that tests can prevent bad updates!

Recommendations

- ❏ Confidence Score
  - ❏ How reliable is my test suite for a particular library?
  - ❏ Indication on where to direct test efforts
- ❏ Gaps in Test Coverage
  - ❏ Complement with Static Analysis
  - ❏ Catch early errors without running build/tests

Recommendations

- ❏ Confidence Score
  - ❏ How reliable is my test suite for a particular library?
  - ❏ Indication on where to direct test efforts
- ❏ Gaps in Test Coverage
  - ❏ Complement with Static Analysis
  - ❏ Catch early errors without running build/tests

Recommendations

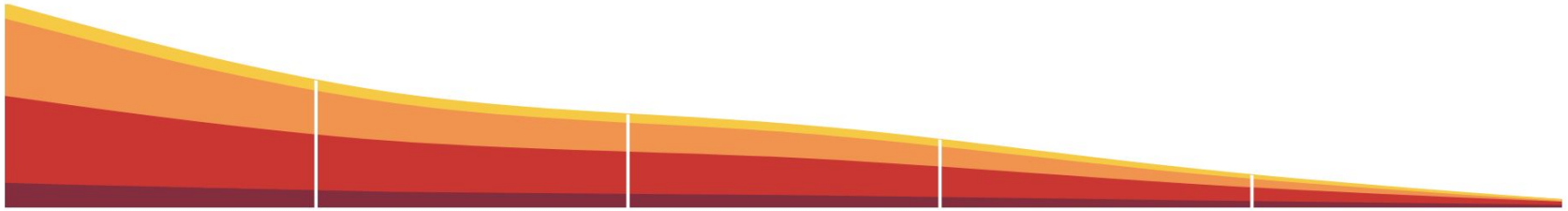
- ❏ Reuse is “free” but the operational/maintenance costs are not “free”
- ❏ Should not blindly trust automated dependency updates—I guess no one does this :D
- ❏ Write tests for critical dependencies

Implications

Supply Chain Vulnerabilties

Software Composition Analysis (SCA)

3.34K	1.6K	1.51K	591	125
Total Open Vulnerabilities	Not In Test	Fix Available	Reachability	Exploitable Likelihood
			Reachable Function	EPSS > 1%



Getting less noisy….

Why do you need Code-First SCA?

Most effective way to cut noise!

Transitive Dependencies

Analysis going beyond usage of your imported libraries

Phantom Dependencies

```
subprocess.check_call([sys.executable, "-m", "pip", "install", "scipy"])
```

Upgrade Impact Analysis

Call graphs can complement coverage gaps of tests for library updates

Questions?


ENDOR LABS
ENTER
THE RAFFLE
TO WIN!
https://www.endorlabs.com/private-event/ndc-security-raffle-2025