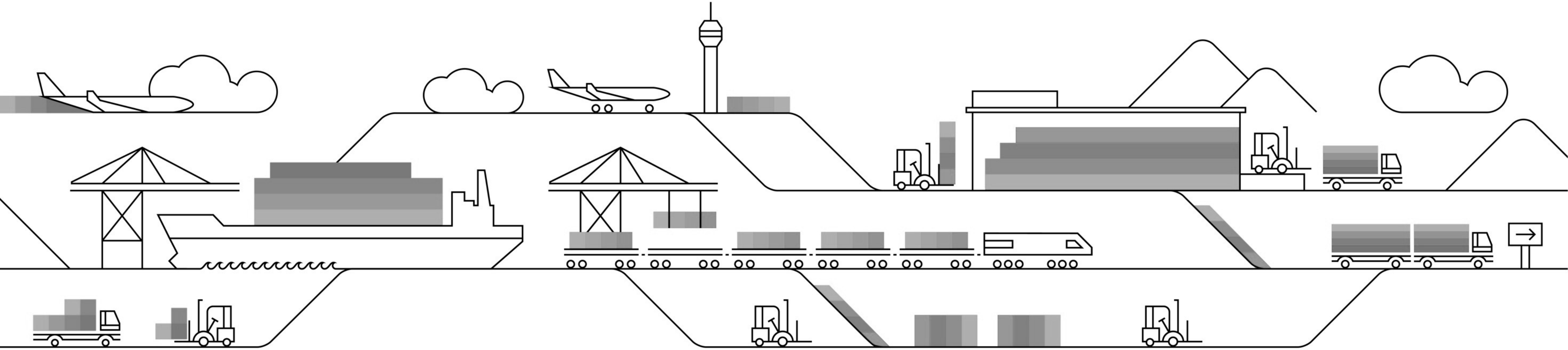


Threats to the Secure Consumption of Open Source

1st Workshop on Trustworthy Software Ecosystems
March 24, 2021
Henrik Plate (SAP Security Research)

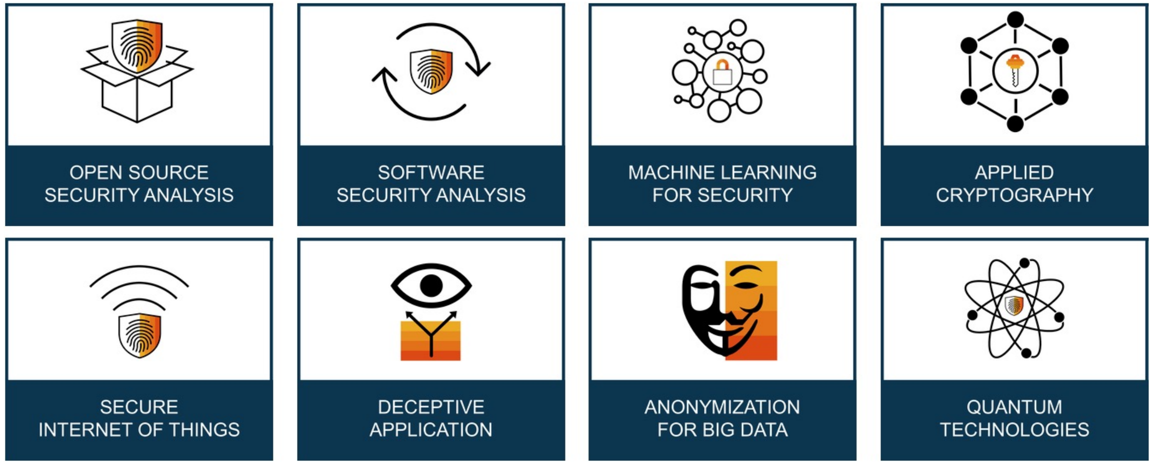
PUBLIC

Threats to the Secure Consumption of Open Source
Agenda

- Two Security Challenges
  - Using Open Source With Known Vulnerabilities
  - Open Source Supply Chain Attacks
- Research Directions of SAP Security Research

About SAP Security Research
Applied Research

- Bridging Academia with SAP Product Development
- Located in Sophia Antipolis, France and Karlsruhe, Germany
- 30 researchers, with 50+ peer reviewed publications since 2017 [1]
- 10+ years of participation in national and EU research projects, e.g., Sparta or AssureMOSS
- 8 strategic research areas [2]



[1] Google Scholar: https://scholar.google.fr/citations?hl=en&user=FOEVZyYAAAAJ&view_op=list_works&sortby=pubdate
[2] Brochure: https://www.sap.com/documents/2017/12/cc047065-e67c-0010-82c7-eda71af511fa.html

Using Open Source
With Known Vulnerabilities

Using Open Source
With Known Vulnerabilities

Wide-spread use of open source [1,5]:

- 80% to 90% of software products on the market include open source components
- Open source in codebase between 10% and 76%
- Development teams use an average of 135 software components of which 90% are open source. It was not uncommon to see applications assembled from 2,000 – 4,000 OSS component releases.

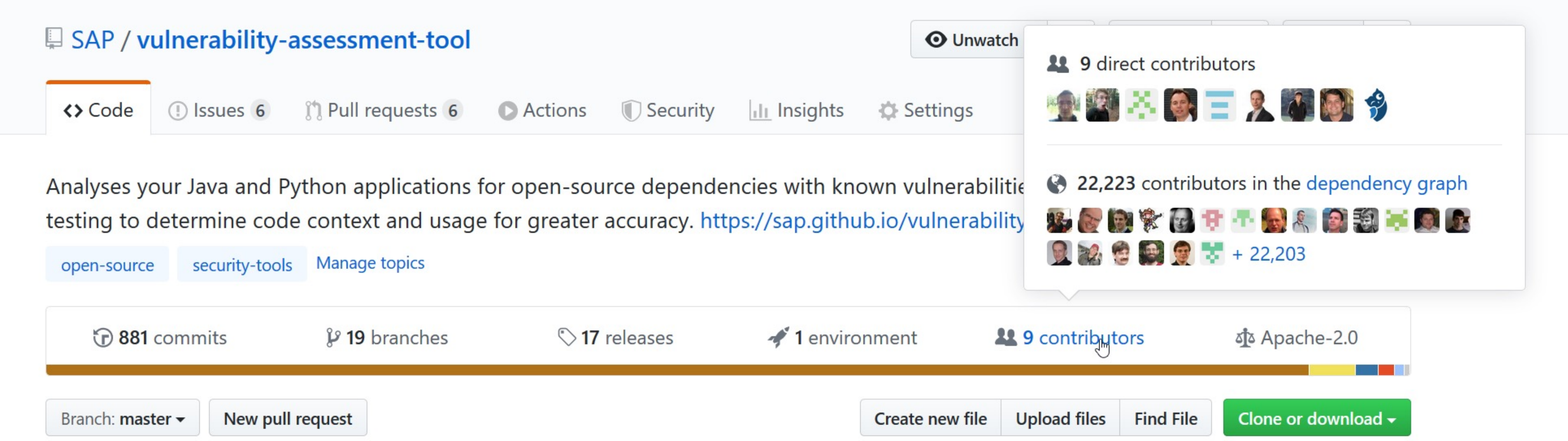
Use of components with known vulnerabilities:

- 11% of the open source components had at least one known security vulnerability. On average, the applications contained 38 known vulnerabilities. [5]
- Included in OWASP Top 10 (2013-2017) [4]
- Root cause of major data breaches, e.g.,
Mossack Fonseca (Panama Papers) and Equifax breach [2]

[1] https://www.synopsys.com/content/dam/synopsys/sig-assets/reports/2018-ossra.pdf
[2] https://snyk.io/blog/owasp-top-10-breaches/
[3] https://snyk.io/opensourcesecurity-2017/
[4] https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf
[5] Sonatype: 2020 State of the Software Supply Chain Report

Using open source components

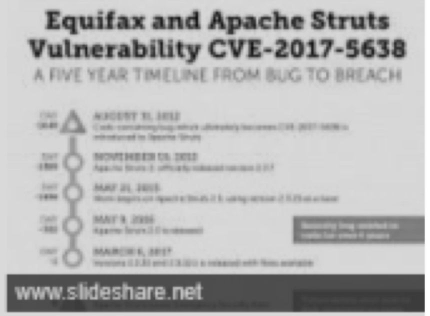
9 + 22223 contributors



The impact of using vulnerable open source can be significant

Example: Equifax

Equifax confirmed that their high profile, high impact data breach was due to an exploit of a vulnerability in an open source component, **Apache Struts CVE-2017-5638. Apache Struts** is a mainstream web framework, widely used by Fortune 100 companies in education, government, financial services, retail and media.



Behind the Equifax Breach: A Deep Dive Into Apache Struts CVE-2017 ...
https://www.brighttalk.com/.../behind-the-equifax-breach-a-deep-dive-into-apache-struts...

Equifax's Data Breach Costs Hit $1.4 Billion

Massive 2017 Breach Continues to Bite the Credit Reporting Giant's Bottom Line

Mathew J. Schwartz (@euroinfosec) • May 13, 2019

Twitter Facebook LinkedIn Credit Eligible Get Permission

https://www.bankinfosecurity.com/equifaxs-data-breach-costs-hit-14-billion-a-12473

Source: Google

After Heartbleed and Equifax
Entering the Hamster Wheel

- **Check** for new vulnerability disclosures (hopefully automated)
- Dismiss false-positives, **assess** true positives (keep fingers crossed for false-negatives)
- **Mitigate** (from *piece-of-cake* to *very expensive*)
- **Release patch** (cloud ☺ on-premise 😐 devices ☹)

Example Vulnerability for Eclipse Mojarra
CVE-2018-14371

The `getLocalePrefix` function in `ResourceManager.java` in Eclipse Mojarra before 2.3.7 is affected by Directory Traversal via the `loc` parameter. A remote attacker can download configuration files or Java bytecodes from applications.

CVSS Base Score: 7.5 (high)

References: fix-commit and issue

Affected products:

- `cpe:2.3:a:eclipse:mojarra:*` up to (excluding) 2.3.7

Still the best public vuln. DB, but many problems, e.g.

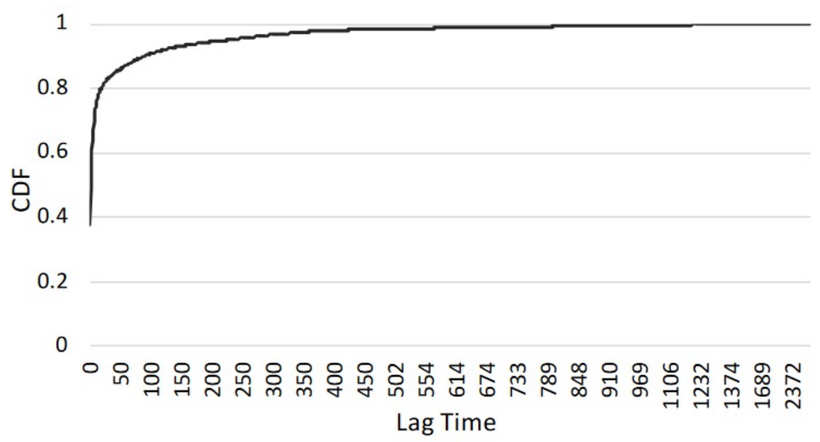
- Short CVE descriptions and varying quality of referenced information
- CPE identifier != package identifier (30 search hits for "mojarra" on Maven Central don't include `org.glassfish:javax.faces`)
- Coarse-granular reference of entire projects, ignoring reusable components and code [3] (700+ artifact versions contain the resp. classes)
- Error-prone (2.3.5 and 2.3.6 were also affected)
- Some ecosystems are not well covered, e.g., npm

References:
[1] Anwar, A., et al.: Cleaning the NVD: Comprehensive Quality Assessment,Improvements, and Analyses (2020)
[2] The State of Exploit Development: 80% of Exploits Publish Faster than CVEs (2020)
[3] Plate, H., et al.: Impact assessmentfor vulnerabilities inopen-sourcesoftware libraries (2015)

NVD Publication Lags
Response Windows are Getting Smaller

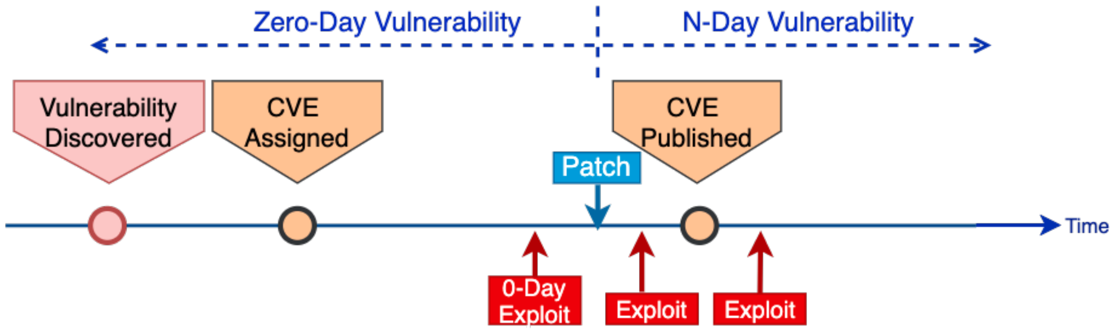
Anwar et al. analyzed lag between publication of 107.2K CVEs and referenced web pages [1]:

- ~38% have a lag of zero day
- ~28% have a lag of more than a week



Palo Alto Networks correlated 11K exploits from EDB with CVE and patch information [2]:

- 14% exploits are published before the patch
- 23% within a week after the patch
- 80% before the CVEs are published



Equifax and CVE-2017-5638

- 3 days between patch (March 7th), data breach and CVE publication (both March 10th) [3]

References:
[1] Anwar, A., et al.: Cleaning the NVD: Comprehensive Quality Assessment,Improvements, and Analyses (2020)
[2] Palo Alto Networks: The State of Exploit Development: 80% of Exploits Publish Faster than CVEs (2020)
[3] Fruhlinger, J.: https://www.csoonline.com/article/3444488/equifax-data-breach-faq-what-happened-who-was-affected-what-was-the-impact.html (2020)

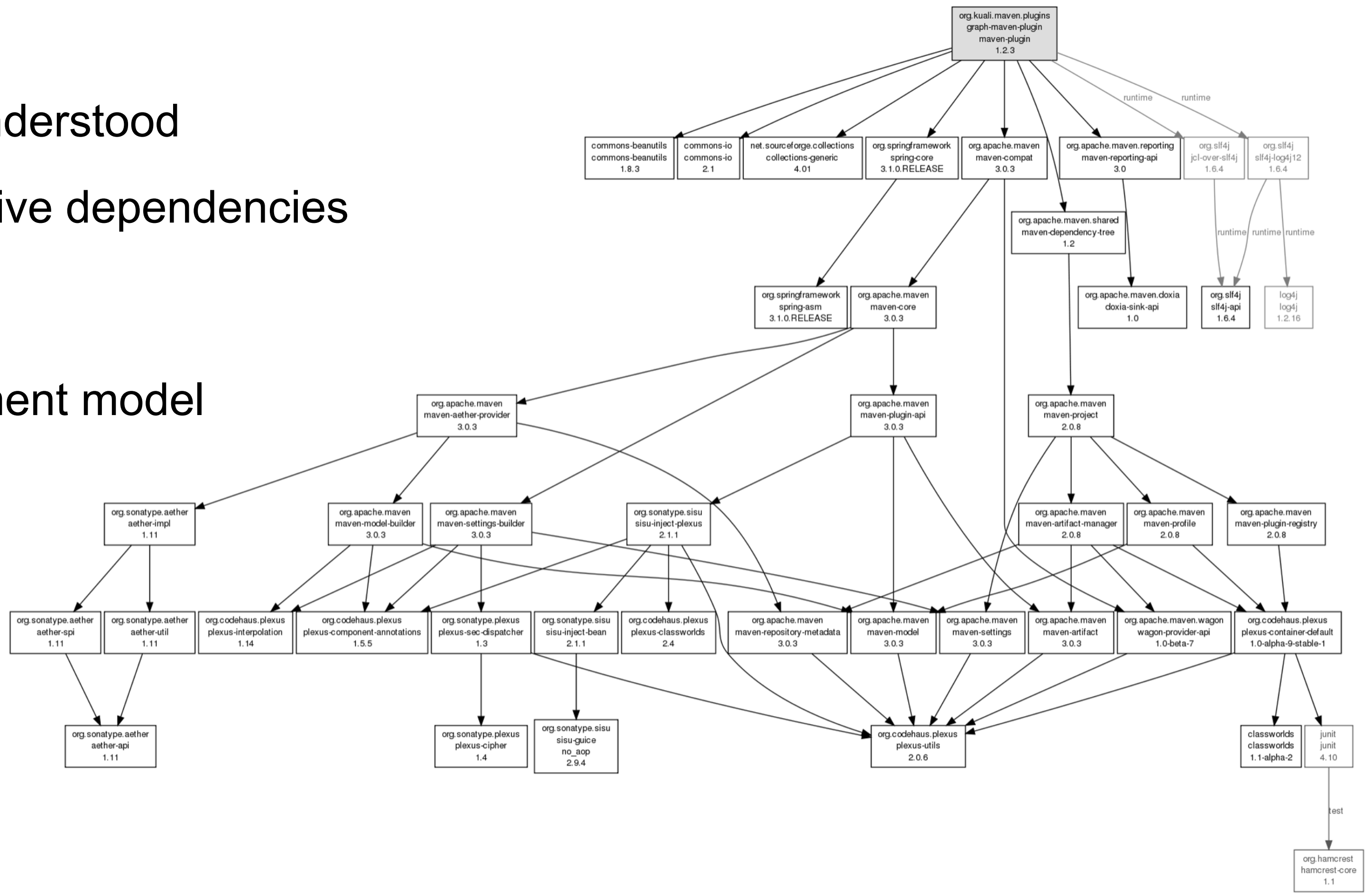
Assessment and mitigation is difficult
Complex dependency graphs

Direct vs. transitive dependencies

- Transitive dependencies not known or understood
- 78% of vulnerabilities are found in transitive dependencies

Simply update?

- Depends on lifecycle phase and deployment model
- Breaking changes

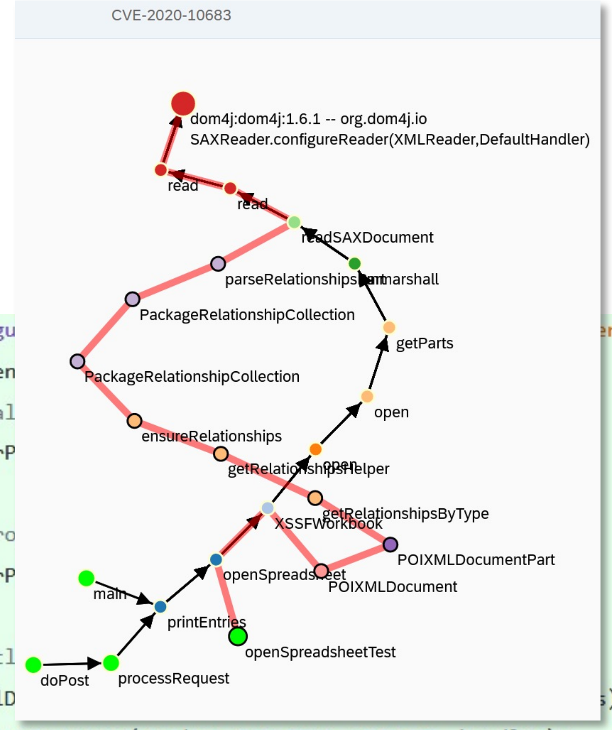


Open Source Vulnerability Detection
Two Approaches

Metadata-based

- Primarily rely on package names and versions, package digests, CPEs, etc.
- Example: OWASP Dependency Check (light-weight, maps against CVE/NVD)

Fig. 2. Static and dynamic paths to vulnerable method



Code-based

- Detect the presence of code (no matter the package)
- Example: Eclipse Steady (heavy-weight, requires fix-commits)
- Supports impact assessments (static and dynamic analyses), esp. important for later lifecycle phases and non-cloud
- Supports update metrics to avoid regressions [1]
- Based on Project KB, which contains fix commits for given vulnerabilities

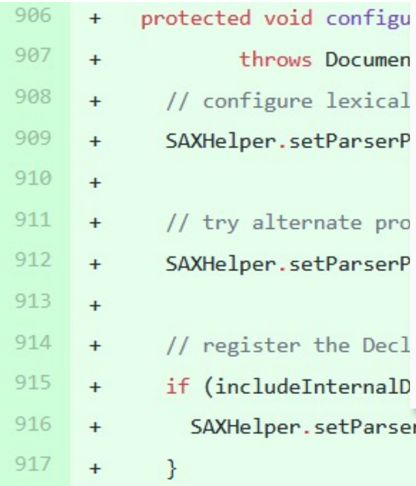


Fig. 1. Fix-commit for CVE-2020-10683

References:
[1] Ponta, S., et al.: Beyond Metadata: Code-Centric and Usage-Based Analysis of Known Vulnerabilities in Open-Source Software (2018)

Open Source Supply Chain Attacks

NPM package `event-stream`
November 2018

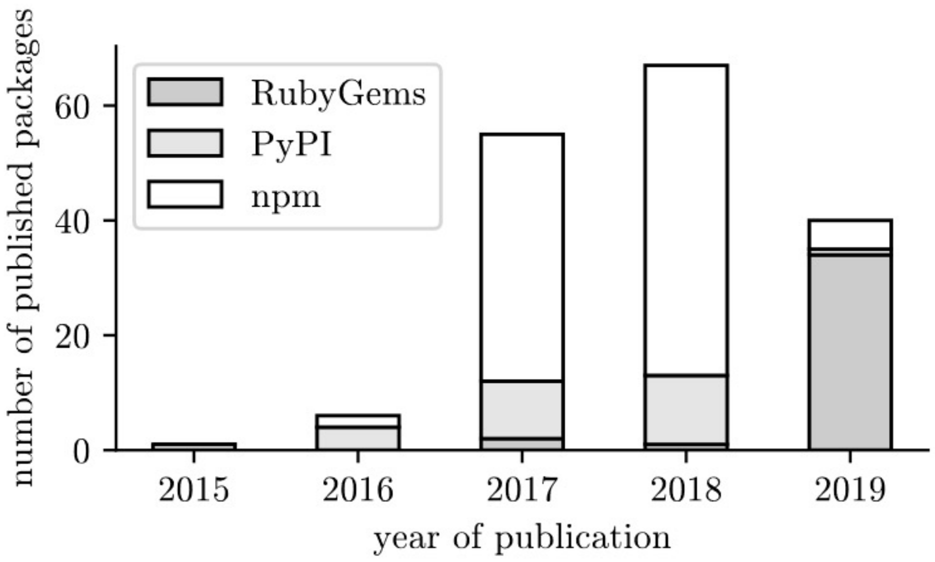
- 1.5+ million downloads/week, 1600 dependent packages
- When contacted by mail, the original developer handed-over the ownership to “right9control”
- Added dependency on the malicious package `flatmap-stream`
- Malicious code (and encrypted payload) only present in published NPM package
- Malware and decryption only ran in the context of a release build of the bitcoin wallet `copay`
- `Credentials.getKeys` was monkey-patched and exfiltrated wallet credentials
- Malware was discovered only by incident: Use of deprecated command resulting in a warning

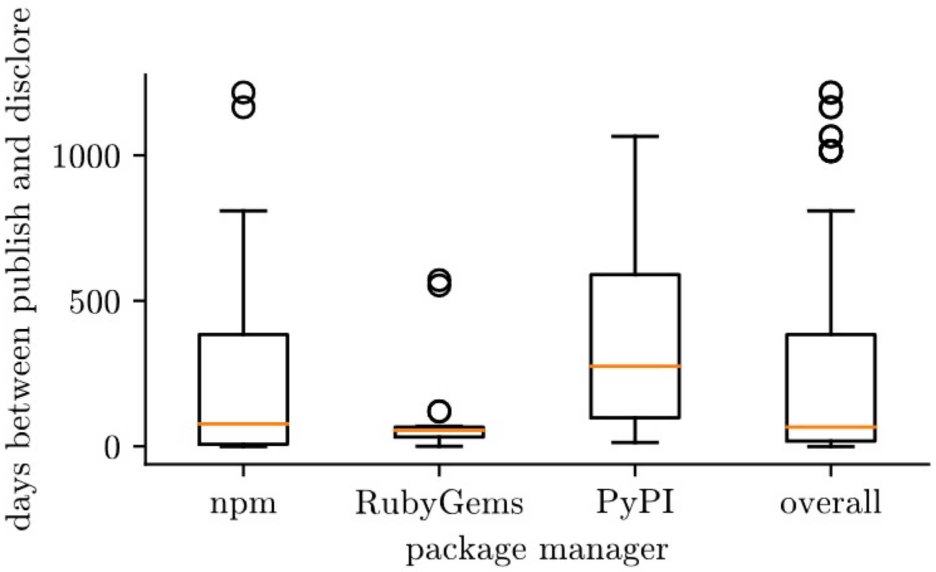
References:
- https://www.theregister.co.uk/2018/11/26/npm_repo_bitcoin_stealer/
- https://medium.com/intrinsic/compromised-npm-package-event-stream-d47d08605502

Increasing Number of Supply Chain Attacks

Open dataset with 174 malicious packages [1], for which the actual code could be obtained

Manual classification by Ohm et al. [2]: Temporal aspects, trigger, injection technique, conditional execution, primary objective, targeted OS, use of obfuscation, and clusters/campaigns





References:
[1] https://github.com/dasfreak/Backstabbers-Knife-Collection
[2] Ohm, M., et al.: Backstabber's Knife Collection (2020)

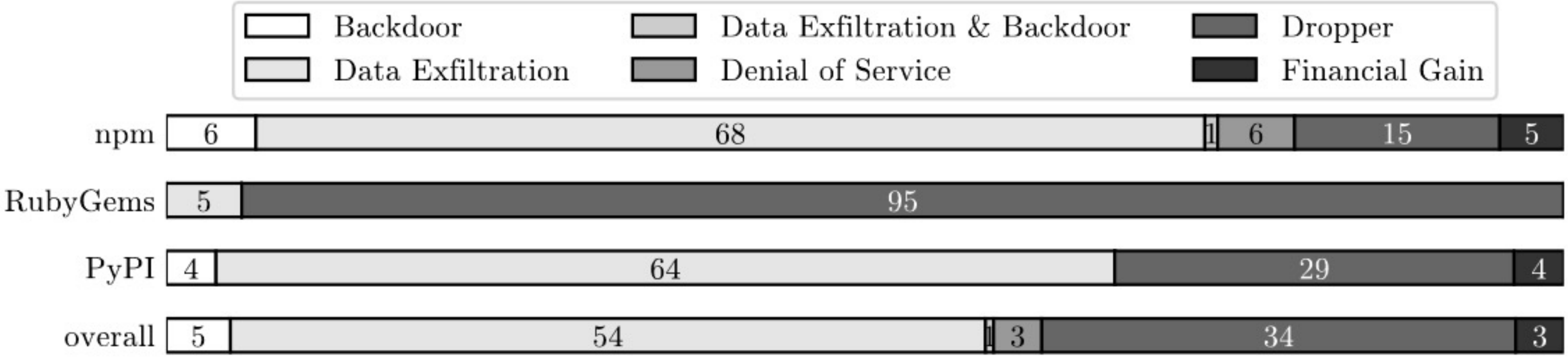
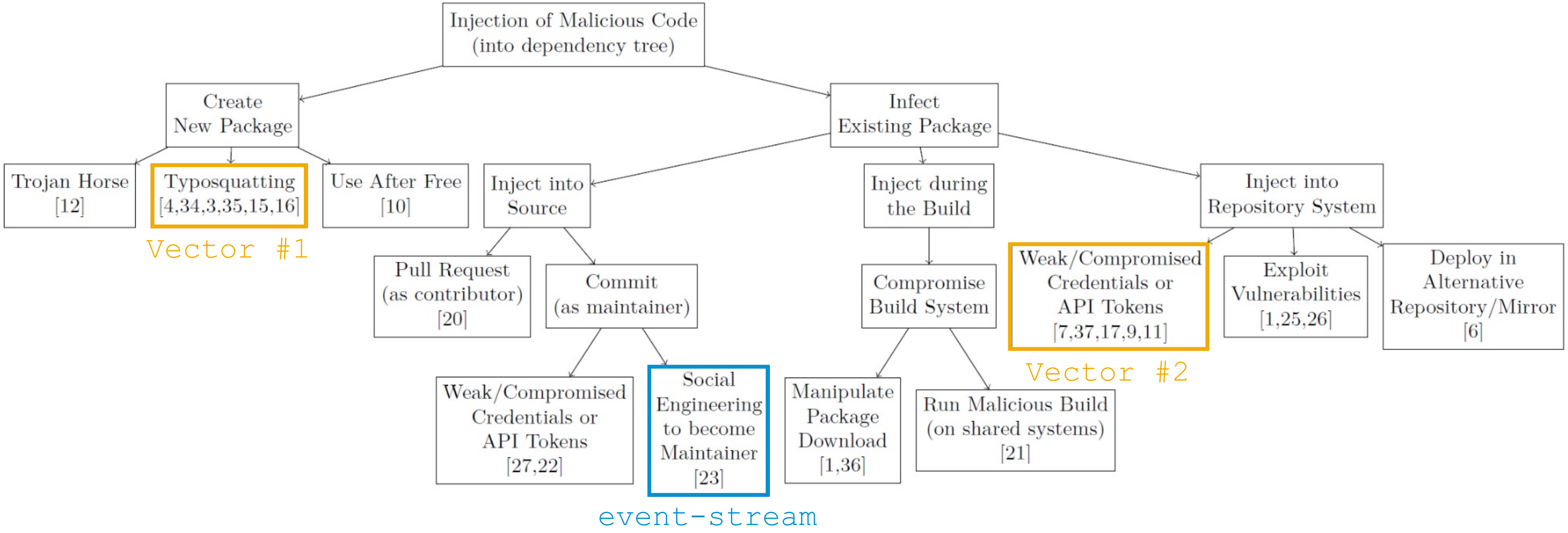


Fig. 9. Primary objective of the malicious package per package repository and overall.

Attack Tree

Make Downstream Users Depend on a Malicious Package



References:

- Ohm, M., et al.: Backstabber's Knife Collection (2020)

"Attacks abuse
users' trust in the authenticity of packages hosted on external servers,
and their adoption of automated build systems that encourage this practice" [1]

References:
[1] Chess, B., et al.: Attacking the Build through Cross-Build Injection: How Your Build Process Can Open the Gates to a Trojan Horse. (2007)

A Closer Look at Trust
The npm Ecosystem

Metrics defined by Zimmermann et al. [1]

- Package Reach (PR) and Maintainer Reach (MR)
- Implicitly Trusted Packages (ITP) and Maintainers (ITM)

Dual-use

- Attackers: “Those maintainers/projects are attractive targets”
- Defenders: “Those require special support and care”

Model to reflect cost/benefit considerations of attackers, in order to protect likely targets

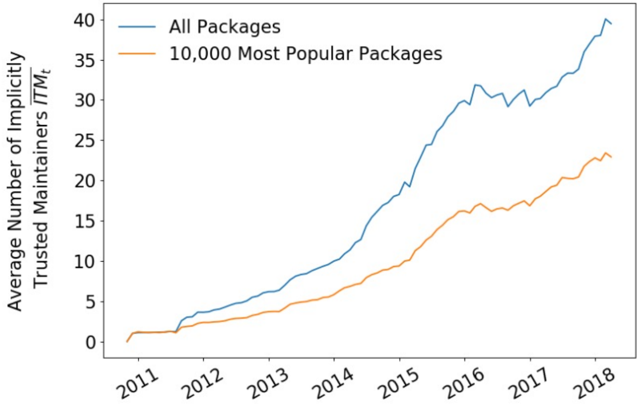


Figure 8: Evolution of average number of implicitly trusted maintainers over years in all packages and in the most popular ones.

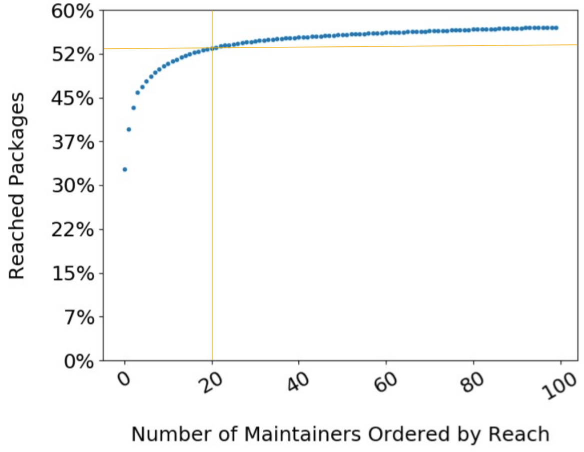


Figure 11: Combined reach of 100 influential maintainers.

References:
[1] Zimmermann, M., et al.: Small World with High Risks (2019)

Trusting Developers to Understand Password Security

Gathering weak npm credentials [1]

Valid credentials of 17088 accounts were bruteforced or leaked.

16901 accounts have published something (~13% of all 125665 accounts).

Directly affected packages: 73983 (14%), indirectly affected packages: ~ 54%

4 users from the top-20 list were affected:

- One who controls > 20 million downloads/month improved the previously revoked password by adding ”!”
- One of those set their password back to the leaked one shortly after it was reset.

References:
[1] Chalker: Gathering weak npm credentials (2017)

Vulnerable or Malicious?
Telling things apart is difficult!

1) Don’t look at the source code repository but at distributed packages

2) 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;
```

= != ==

3) Research and, as far as known, recent attacks, focus on interpreted languages [2,3,4,5]
   Detection gets more difficult with compilation, code generation, re-bundling, re-packaging, …

References:
[1] Wysopal, C., End, C.: Static Detection of Application Backdoors (2010)
[2] Vu, Duc-Ly, et al.: Poster: Towards Using Source Code Repositories to Identify Software Supply Chain Attacks (2020)
[3] Pfretzschner, B., et al.: Identification of Dependency-based Attacks on Node.js (2017)
[4] Garret, K., et al.: Detecting suspicious package updates (2019)
[5] Taylor, M., et al.: SpellBound: Defending Against Package Typosquatting (2020)

Supply Chain Attacks
Take-Aways

- Many people thank you for putting trust in their security capabilities
- Number of dependencies and actors + complexity of build processes and infrastructures result in a considerable the attack surface
- Noticeable increase in supply chain attacks targeting open source ecosystems
- Python, Node.js and Ruby ecosystems are the primary targets
(but some ecosystems like Java have not been analyzed in a systematic fashion)

Protection against malicious open source components

- All dependencies matter (not only compile/runtime ones as for known vulnerabilities)
- The truth is in downloaded packages (source code visible in GitHub etc. does not matter)

Active field of research, e.g., as part of EU Research Project SPARTA [1]

References:
[1] https://www.sparta.eu/

SAP Security Research

Research Directions

Research Directions
Attack Surface Reduction

Goals

- Remove unused (bloated) open source code that is pulled automatically by package managers, but not actually used by the app
- Removal of exploit gadgets and vulnerable code reduces attack surface and maintenance effort

Evaluate state-of-the-art debloat tools for Java in industrial environments, e.g.,

- DepClean [1]
  "Our key result is that 75.1% of the analyzed dependency relationships are bloated."
- Jshrink [2]
  "able to debloat our real-world Java benchmark suite by up to 47% (14% on average)"

References:
[1] Soto-Valero, C., et al.: A Comprehensive Study of Bloated Dependencies inthe Maven Ecosystem (2020)
[2] Bruce, R., et al.: JShrink: In-Depth Investigation into Debloating Modern Java Applications (2020)

Research Directions
Supply Chain Attacks

Goals

- Detect malicious code in upstream open source code projects
- Esp. in those ecosystems/languages not sufficiently covered by research (compiled and hybrid languages, e.g., C code in PyPI packages or C#)

State-of-the-art (moving very quickly)

- Dynamic detection at build time [1]
- Detection of typo-squatting attacks [2]
- Static source code analysis [3]
- Anomaly-based detection based on API usage [4]
- …

References:
[1] Ohm, M., et al.: Towards Detection of Software Supply Chain Attacks by Forensic Artifacts (2020)
[2] Vu, Duc-Ly, et al.: Typosquatting and Combosquatting Attacks on thePython Ecosystem (2020)
[3] Pfretzschner, B., et al.: Identification of Dependency-based Attacks on Node.js (2017)
[4] Garret, K., et al.: Detecting suspicious package updates (2019)

Research Directions
Code Fingerprints

Goals

- Establish equality/distance of Java source code and byte code
- Use-cases: Malware detection and identification of vulnerable code

State-of-the-art

- Java Decompilation [1]
  “Our results show that no single modern decompiler is able to correctly handle the variety of bytecode structures coming from real-world programs.”
- Use of IR for byte code comparison [2]
- Definition of metrics to link a Java binary to its source code [3]

References:
[1] Harrand, N., et al.: Java decompiler diversity and its application to meta-decompilation (2020)
[2] Dann, A., et al.: Bytecode Comparison Across Different Java Compilers (2019)
[3] Davies, J., et al.: Software Bertillonage (2012)

Thank you.

Henrik Plate

Senior Researcher

SAP Security Research

SAP Labs France, 805 Av. Maurice Donat

F-06250 Mougins