Dependable software engineering
Can we increase trust in our components?

Marcelo Pasin

Hes·so
Haute Ecole Spécialisée
de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences and Arts
Western Switzerland

unine
Université de Neuchâtel

SCS

ifip

IFIP Working Group 10.4
Dependable Computing and Fault Tolerance

87th meeting – Praia do Forte, Bahia, Brazil – 6-10 Feb 2025

How much we can trust in current software distributions?

- Let’s say we want to deploy a web server
  - Example: nginx, latest version: 1.27
- Easy today
  - `apt install nginx`
  - `docker run nginx`
  - `kubectl apply` *`(your manifest here)`*
- **Is it safe?**
- Existing tools:
  - GitHub Dependabot, OWASP dependency check
  - npm/pi/cargo audit, Snyk, Trivy
  - SCAs Black Duck, JFrog Xray, Sonatype Nexus

What is the fundamental problem?

- Software can be hacked (old story)

- All software is the result of compositions
  - Including numerous dependencies (and sub-dependencies)
  - Composition is done at different levels (compilation, packaging, etc)

- **It is hard to figure out what we are actually executing**
  - It is hard to ensure trust in software supply chains (traceability)
  - Can any of my components be hacked?
- The composition (supply) chains can be hacked
  1. Malicious additions to genuine source code
  2. Abuses in the distribution (malicious replacements, homonyms, etc)

Remember
- SolarWinds
- Log4j?

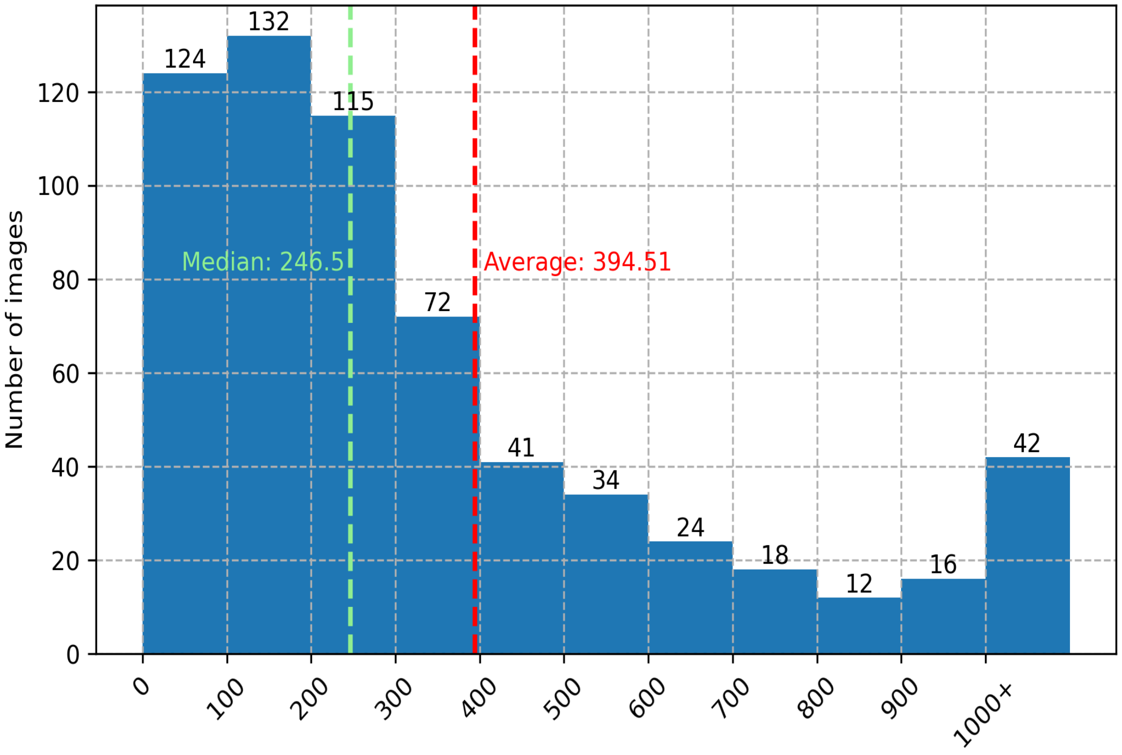
How bad is this problem?

How much we can trust in existing software dependencies?

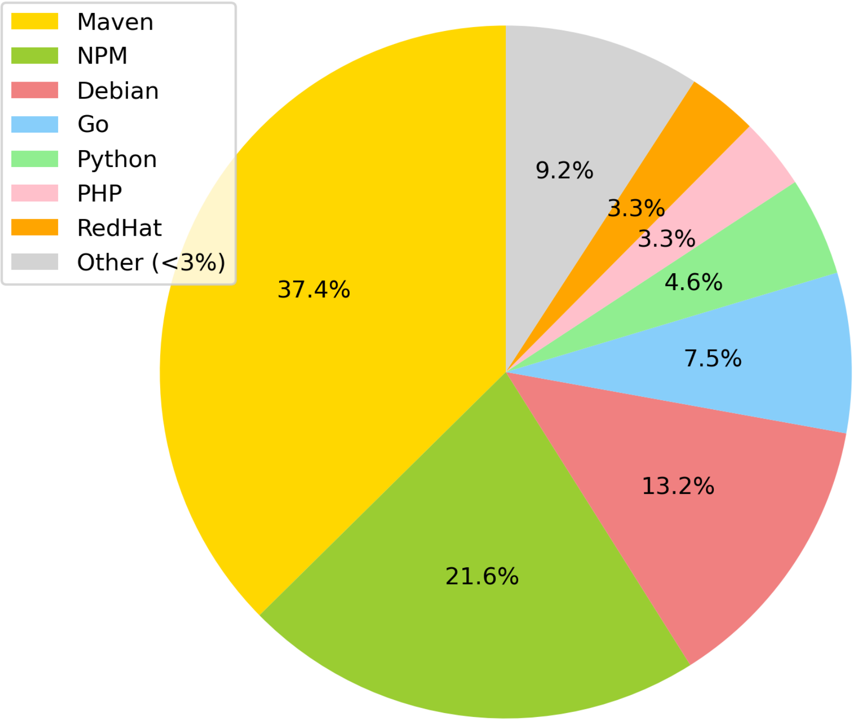
- Analysed Docker Hub's 1000 Top Repositories
- Safe to assume they are widely deployed, maybe even in critical systems

- Used information from images' BOMs
  - 63% of images contained a BOM

Docker Hub dependencies

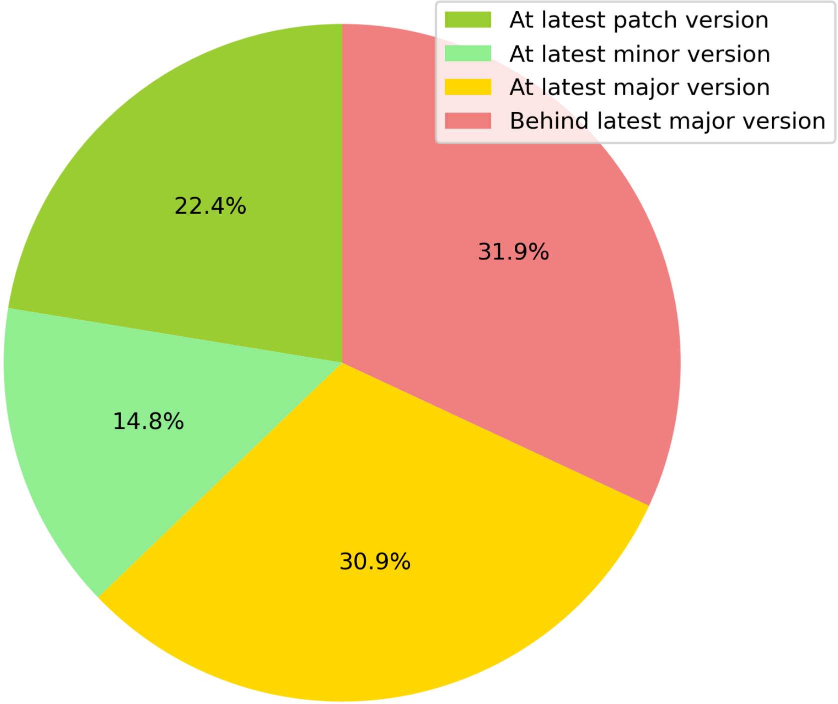


Number of dependencies

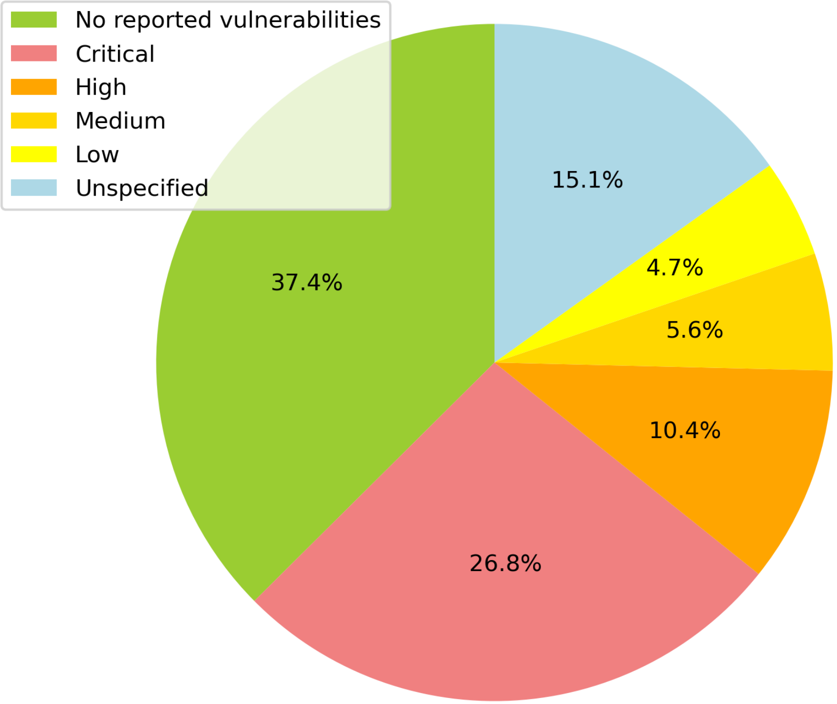


Dependencies sources

Docker Hub dependencies



Dependencies versions’ upkeep



Known CVE vulnerabilities

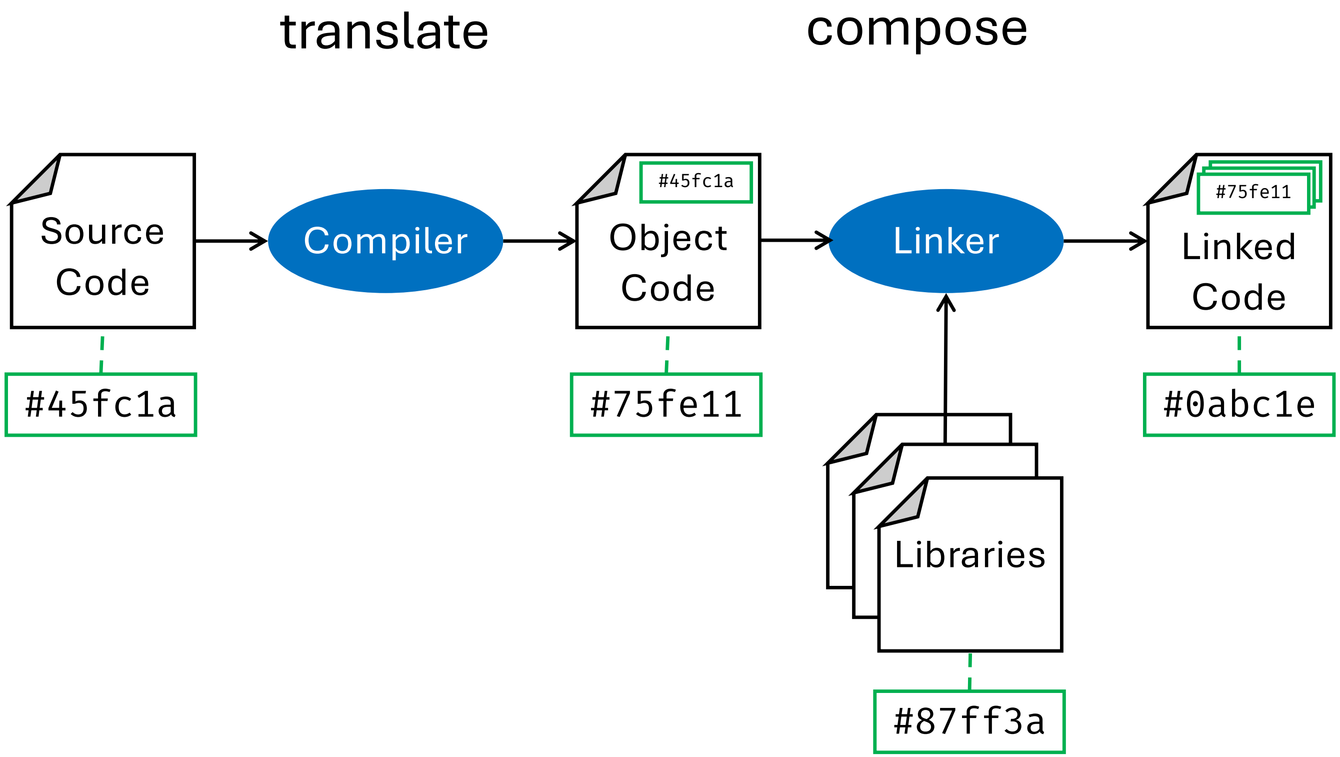
How can we add trust?

Full traceability can help solving the problem

- Consider software composition as a tree
  - Executable is the root, sources are leaves
  - Composition points are edges
  - Intermediary files are nodes in the tree

- Simple steps
  - Annotate all nodes in the tree
  - Certify source code at all leaves
  - Add traceability throughout the tree
    - Annotate node with children's hashes (its sources)
  - Attest the executable code (the root) before executing

Basic traceable software supply chain



Certification

- Expert code review
  - Actual visual code review
  - Formal analysis
  - Pen tests
  - IA can help in all steps, some can be automated
- Not flawless, but many decent techniques exist

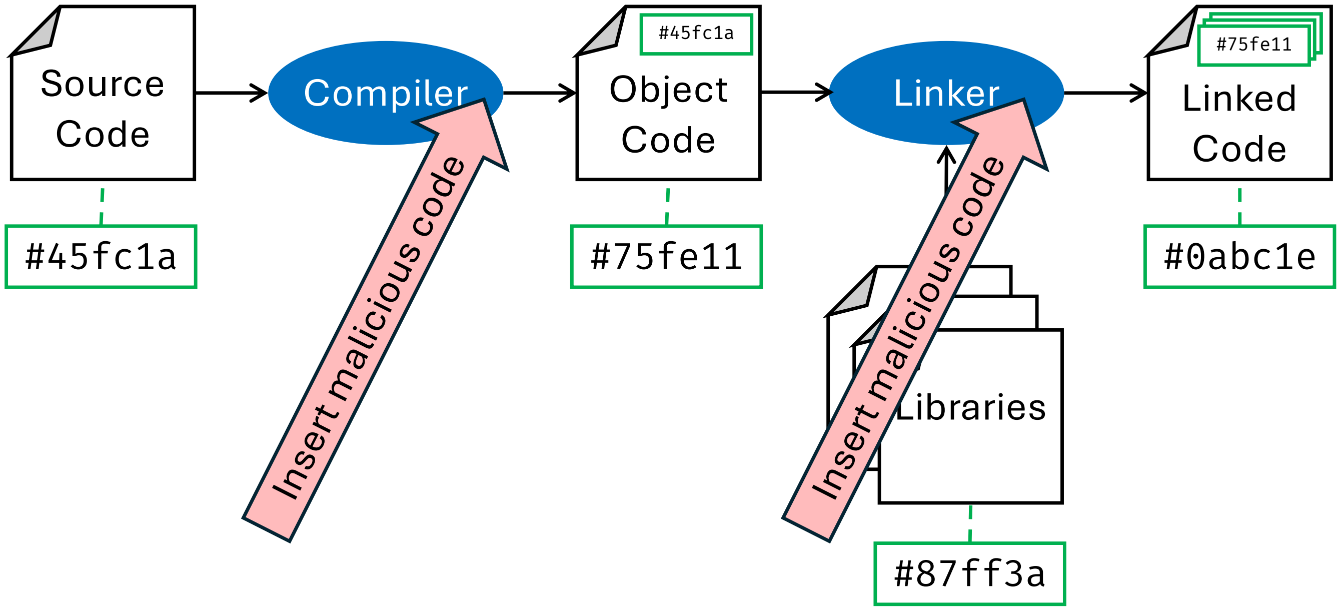
- If pass, a certificate is issued and signed
  - All source files must carry a certificate

Is the "flawless fault detector" a thing?

Traceability

- Start with certified/signed sources
- Hashes are combined at every composition (ex. linking)
  - E.g. Merkle trees
- Annotations must be implemented at all stages
  - Compilation, linking, composition, layering, etc.
- There are not many popular standards
  - How to annotate
  - How to represent the trace tree
  - Current BOMs / manifests could be extended with annotations

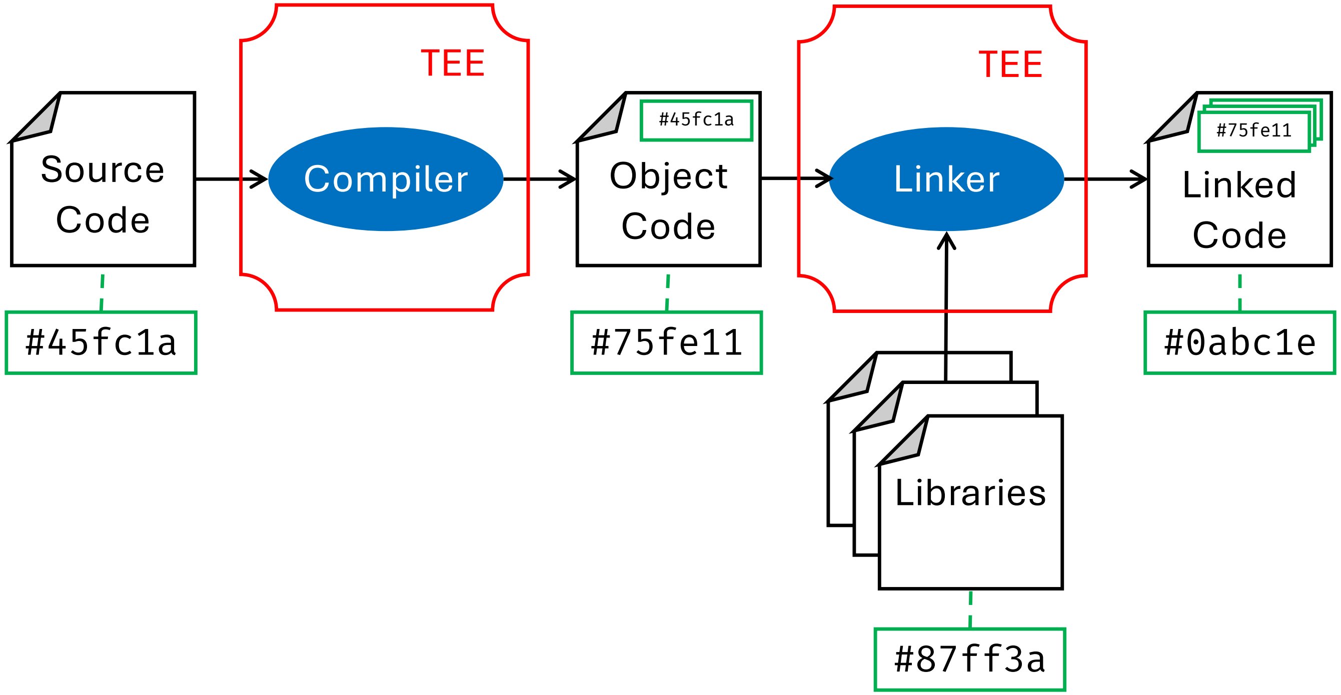
Attacks to the supply chain



Trusted computing

- Need to trust all tools used in the path from source to binary
  - Make sure only genuine tools are running
- Implemented with trusted execution environment (TEE)
  - Intel SGX & TDX, AMD SEV, Arm TrustZone & CCA, RISC-V CoVE
- Changes to running software can be detected
  - Trusted hardware provide remote attestation
- Annotations can be signed inside TEE, increasing trust in the chain

Trusted traceable software supply chain



Prototype implementation

- Development chain for Rust into WebAssembly
  - Implemented in cooperation with SCS https://www.scs.ch/
  - Relies on previous work with UNINE

- Wasmsign
  - Embed metadata in WebAssembly binaries
- Wasmshield
  - Integrated with Rust DK
    - Custom workflows to produce, bundle, and verify proofs
      - Component integrity
      - SBOM vulnerability assessment

Wrap-up

- Fully trusted traceability is rather easy to implement
- Need modifications in the tools for code transformation
- Great help from trusted execution environments

Thank you for your time !

Hes·so
Haute Ecole Spécialisée
de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences and Arts
Western Switzerland

unine
Université de Neuchâtel

SCS