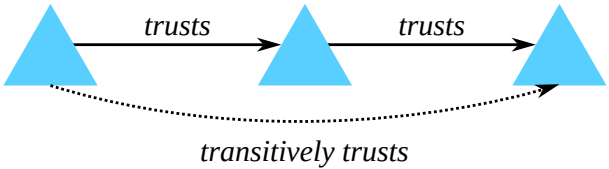
NixCon '24

rebuilding builders instead of trusting trust

Martin Schwaighofer, Michael Roland, René Mayrhofer
Johannes Kepler University Linz
2024-10-25, Berlin (Germany)

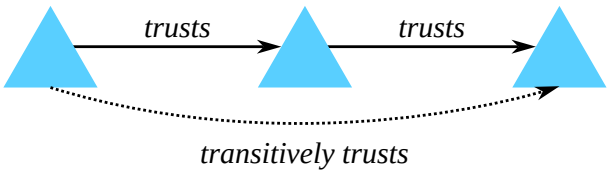
JOHANNES KEPLER UNIVERSITY LINZ
Altenberger Straße 69
4040 Linz, Austria
jku.at

Problem Space



Problem Statement: trusting compiled software necessitates trusting its build environment (recursively) [3]

Problem Space



Problem Statement: trusting compiled software necessitates trusting its build environment (recursively) [3]

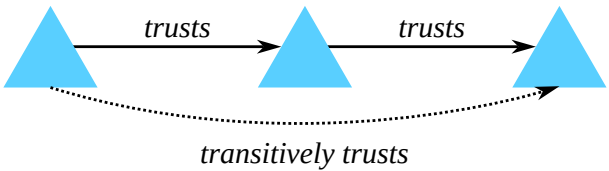
Proposed Solution:

What: eliminate transitive trust for/using cloud build systems [2]

How: make trust relationships of builders as independent as possible

Why: enable distributed ecosystem of builders building open source components

Problem Space



Problem Statement: trusting compiled software necessitates trusting its build environment (recursively) [3]

Proposed Solution:

What: eliminate transitive trust for/using cloud build systems [2]

How: make trust relationships of builders as independent as possible

Why: enable distributed ecosystem of builders building open source components

Related Work: in-toto [4], gitian [1], Trustix [5]

Presentation Overview

Defining Properties of Cloud Build System

- Output Lookup by Hash
- Hermetic Isolation

Presentation Overview

Defining Properties of Cloud Build System

- Output Lookup by Hash
- Hermetic Isolation

Data Structures

- Trace Map Entry
- Provenance Log Entry $\leftarrow$ introduced by us

Presentation Overview

Defining Properties of Cloud Build System

- Output Lookup by Hash
- Hermetic Isolation

Data Structures

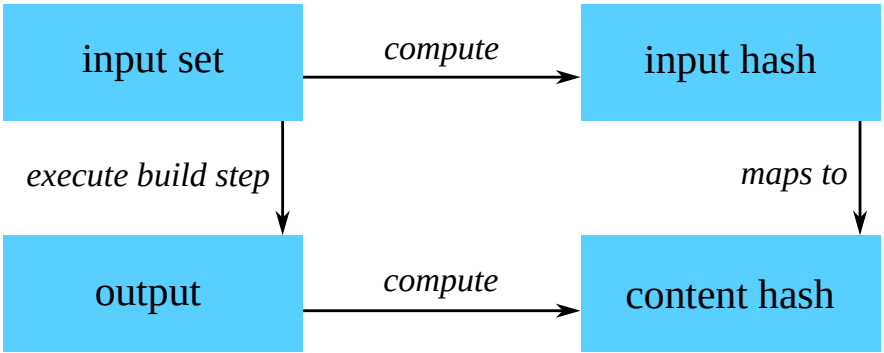
- Trace Map Entry
- Provenance Log Entry ← introduced by us

Usage and Trust Model

Threat Model

1. introduce threat
2. add data to Provenance Log Entry to mitigate threat
3. repeat

Cloud Build Systems: Output Lookup by Hash



Cloud Build Systems: Hermetic Isolation

Build steps can only access dependencies, which are part of the input set.

- Ensures input set is complete
- Prevents tampering during build
- Essential for trustworthy remote building and caching

Cloud Build Systems: Hermetic Isolation

Build steps can only access dependencies, which are part of the input set.

- Ensures input set is complete
- Prevents tampering during build
- Essential for trustworthy remote building and caching
- Separates *intended dependencies* (input set) from both
  - *inadvertent dependencies* like kernel, drivers, cloud build system, and
  - *coincidental dependencies* like random stuff on the system

Data Structures

Trace Map Entry
input hash, content hash of mapped output

data structure (often) at the heart of cloud build systems

Data Structures

Trace Map Entry

input hash, content hash of mapped output

data structure (often) at the heart of cloud build systems

Provenance Log Entry 

input hash, content hash of mapped output **+ arbitrary provenance data**

cryptographically secured, communicates trust, extension, extendable

Trust Model(s)

- set of trusted public keys
- additional constraints on provenance data

Trust Model(s)

- set of trusted public keys
- additional constraints on provenance data

- drives a binary trusted/untrusted decision
- independent from other builders (as much as possible)

Trust Model(s)

- set of trusted public keys
- additional constraints on provenance data

- drives a binary trusted/untrusted decision
- independent from other builders (as much as possible)

- Example trust models:
  - one trusted builder or entity / centralized infra
  - 2 out of 3 of (independent) trusted builders agreeing
  - quorum of all builders that meet criteria agreeing

Usage and Threat Model

What we are trying to do:

checked in build files ➔ dependency tree ➔ input hashes ➔
obtain **trustworthy** artifacts from caches

Usage and Threat Model

What we are trying to do:

checked in build files ➔ dependency tree ➔ input hashes ➔
obtain **trustworthy** artifacts from caches

user's trust model ensures:

- user only accepts trustworthy outputs from caches (when building)
- can verify every link in the dependency tree (when verifying)

Usage and Threat Model

What we are trying to do:

checked in build files ➔ dependency tree ➔ input hashes ➔
obtain **trustworthy** artifacts from caches

user's trust model ensures:

- user only accepts trustworthy outputs from caches (when building)
- can verify every link in the dependency tree (when verifying)

attacker wants to bypass user's declared/intended trust model:

- first assuming user only trusts honest builders (Threat 1 - Threat 3)
- then assuming user may trust dishonest builders as well (Threat 4 - Threat 5)

Threat 1 and 2: Unclear Origin of Outputs

- signatures (in Nix) originally designed for transport security
- signatures do not state who the builder was
- **Threat 1:** Who are we trusting to be an honest builder?
- **Threat 2:** How can we determine reproducibility from provenance data?

Provenance Log Entry	
input hash, content hash of mapped output	

Threat 1 and 2: Unclear Origin of Outputs

- signatures (in Nix) originally designed for transport security
- signatures do not state who the builder was
- **Threat 1:** Who are we trusting to be an honest builder?
- **Threat 2:** How can we determine reproducibility from provenance data?

Provenance Log Entry 
input hash, content hash of mapped output, **claim to have built this [true/false]**

Threat 3: Dependency Resolution Gap

- cannot trust other builders with dependency resolution
  - input set must be constructed using resolved dependencies or
  - resolved dependencies must be part of provenance data

Provenance Log Entry	
input hash, content hash of mapped output, claim to have built this [true/false]	

Threat 3: Dependency Resolution Gap

- cannot trust other builders with dependency resolution
  - input set must be constructed using resolved dependencies or
  - resolved dependencies must be part of provenance data

Provenance Log Entry	
input hash, content hash of mapped output, claim to have built this [true/false], **resolved dependencies** if necessary	

Cloud Build Systems: Output Lookup by Hash

Definition

Cloud build systems construct a **dependency tree** in which each node is identified by a content or input hash.

Terminal inputs, which are leaves in the dependency tree, for example source files or binary blobs, are referred to by **content hash** (a hash of their contents).

The inner nodes of the dependency tree are build steps, which are always identified by **input hash** (a hash of their input set).

Cloud Build Systems: Output Lookup by Hash

Definition

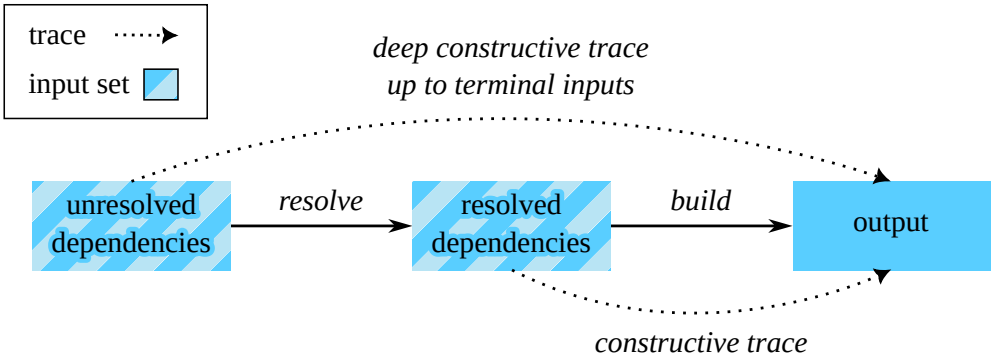
The **input set** of a build step consists of the build instructions that get executed during the specific step, including any terminal inputs, and either

(a) the content hashes of the outputs obtained by building all direct dependencies, or

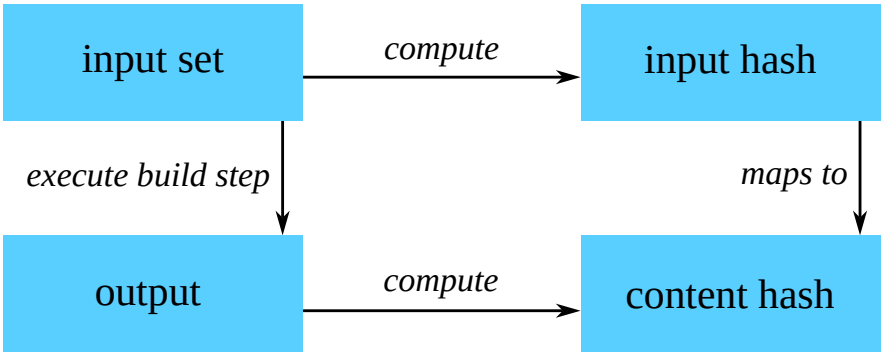
(b) the input hashes of all dependencies, including transitive dependencies via recursion.

The two tracing approaches are known as using **constructive traces** (a), and **deep constructive traces up to terminal inputs** (b).

Tracing Method Comparison



Cloud Build Systems: Output Lookup by Hash



Threat 4 and 5: Dishonest Builders or Outdated Trust Model

- **Threat 4:** What if builders lie on purpose?
- verify instead of trust
- **Threat 5:** What if builders are vulnerable or become compromised?
- increase security over time, recover from compromises

Provenance Log Entry 

input hash, content hash of mapped output, claim to have built [true/false], resolved dependencies if necessary

Threat 4 and 5: Dishonest Builders or Outdated Trust Model

- **Threat 4:** What if builders lie on purpose?
- verify instead of trust
- **Threat 5:** What if builders are vulnerable or become compromised?
- increase security over time, recover from compromises

Provenance Log Entry 

input hash, content hash of mapped output, claim to have built [true/false], resolved dependencies if necessary, **claimed source reference, remote attestation**

Remote Attestation

- Based on TPM 2.0 or Android Key Attestation for example
- Proves key generation in secure hardware
- Incorporates measured boot values
  - Corresponds to built and booted OS image
- Uses as nonce:
  - Trace map entry
  - Monotonically increasing counter

Verify Remote Attestation

The verifier should then be able to

- verify the claimed software configuration of the builder against the verifiers trust model, which may or may not mandate reproducibility,
- derive the expected measurement values for measured boot from the built software configuration,
- verify the signature on the provenance log entry,
- verify from the contents of the remote attestation, that
  - the included trace map entries match up,
  - the upper level measurement values from measured boot match up, the lower level values are trusted,
  - the attestation itself is valid, and
  - the attestation is about the intended signing key.

Conclusion

- Eliminated implicit transitive trust
- Linked provenance data to build steps
- Make trust in build hosts explicit and verify
- Decoupled creation and verification of provenance data
- Potential path forward for supply chain security

Future Work

- Implement proposed extensions in Nix
- Investigate performance implications
- Develop data format for provenance data
- Address open questions on bootstrapping and necessary sandbox improvements
- **let's work on this together**

JOHANNES KEPLER
UNIVERSITY LINZ

NixCon '24

rebuilding builders instead of trusting trust

Email: martin.schwaighofer@ins.jku.at
DOI: https://doi.org/10.1145/3689944.3696169

Martin Schwaighofer, Michael Roland, René Mayrhofer
Johannes Kepler University Linz
2024-10-25, Berlin (Germany)

JOHANNES KEPLER UNIVERSITY LINZ
Altenberger Straße 69
4040 Linz, Austria
jku.at

Bonus Slides

Gitian

Commonalities:

- Signed record of build inputs and outputs
- Aims to increase trust in software supply chain
- Focus on reproducible builds

Differences:

- Single build step
- Relies on trusted Debian packages
- Does not address transitive trust issues
- Better sandboxing using VMs

in-toto

Commonalities:

- Focuses on software supply chain integrity
- Validates signed records of build inputs and outputs against layout
- Uses cryptographic signatures for verification
- Allows for distributed build and deployment pipelines
- Attestation Framework fills similar role to Provenance Data

Differences:

- assigns steps to specific authorized parties
- can be deployed to describe existing build/deployment pipelines
- not a build system

Trustix

- Replaces signatures in Nix with transparency logs
- Additional implications because of switch to transparency logs
- Anchored to builders, not caches (no need for boolean flag)
- Trust Model to distribute trust among various parties
- Practical implementation, abstractly described and extended by this work
- Might be analyzed in future work

Cloud Build Systems: Hermetic Isolation

Definition

In a cloud build system an attacker must not be able to bypass the isolation of executed build steps with adversarial inputs.

We can achieve this by suitably isolating the execution of build steps from the system, each other, and the network.

- Ensures input set is complete
- Prevents tampering during build
- Essential for trustworthy remote building and caching

Backup Questions by Claude.ai

- How do you envision the adoption process for these extensions in existing cloud build systems? What challenges might arise?
- Could you elaborate on how your proposed system would handle scenarios where a previously trusted builder becomes compromised?
- How does your approach compare to or complement other supply chain security initiatives and frameworks like SLSA or in-toto?
- How might this system be extended to handle proprietary or closed-source components in a software supply chain?
- How does your approach address the balance between security and usability for developers and end-users?
- What are the next steps in your research or potential implementation of these ideas?

Acknowledgements

We want to thank Linus Heckemann for first pointing out to us the gaps around the outcome of dependency resolution, which exist in Nix, and which we address in this paper.

This work has been carried out within the scope of Digidow, the Christian Doppler Laboratory for Private Digital Authentication in the Physical World and has partially been supported by the LIT Secure and Correct Systems Lab. We gratefully acknowledge financial support by the Austrian Federal Ministry of Labour and Economy, the National Foundation for Research, Technology and Development, the Christian Doppler Research Association, 3 Banken IT GmbH, ekey biometric systems GmbH, Kepler Universitätsklinikum GmbH, NXP Semiconductors Austria GmbH & Co KG, Österreichische Staatsdruckerei GmbH, and the State of Upper Austria.

References I

[1] Gitian-Builder Contributors. 2024. gitian-builder: Build packages in a secure deterministic fashion inside a VM. Retrieved 07/05/2024 from https://github.com/devrandom/gitian-builder.

[2] Andrey Mokhov, Neil Mitchell, and Simon Peyton Jones. 2018. Build Systems à La Carte. Proc. ACM Program. Lang. 2, ICFP, Article 79, (July 2018), 29 pages. DOI: 10.1145/3236774.

[3] Ken Thompson. 1984. Reflections on trusting trust. Communications of the ACM, 27, 8, 761–763. DOI: 10.1145/358198.358210.

References II

[4] Santiago Torres-Arias. 2020. In-toto: Practical Software Supply Chain Security. PhD thesis. New York University Tandon School of Engineering.

[5] Trustix Project Contributors. 2024. Trustix: Distributed trust and reproducibility tracking for binary caches. Retrieved 05/08/2024 from https://github.com/nix-community/trustix.