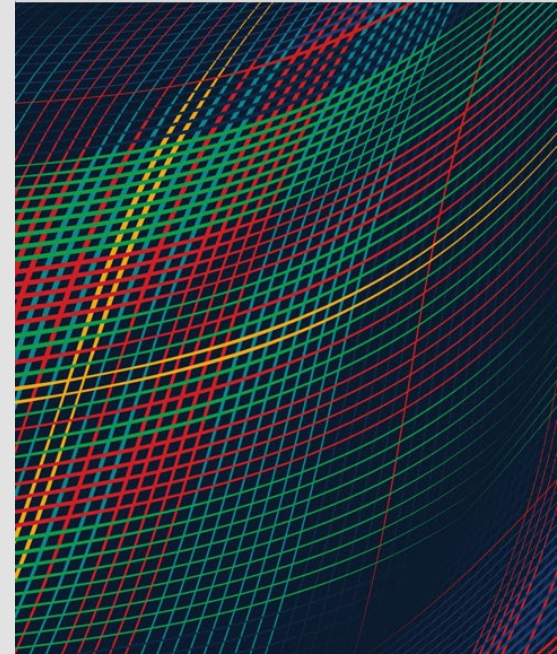


Attestations 101

How to prove your software and supply chain are secure

FEB 28, 2024

Joseph Yankel
Brent Clausner




- Carnegie Mellon University 2024
- This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.
- NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.
- [DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.
- GOVERNMENT PURPOSE RIGHTS – Technical Data
- Contract No.: FA8702-15-D-0002
- Contractor Name: Carnegie Mellon University
- Contractor Address: 4500 Fifth Avenue, Pittsburgh, PA 15213
- The Government's rights to use, modify, reproduce, release, perform, display, or disclose these technical data are restricted by paragraph (b)(2) of the Rights in Technical Data—Noncommercial Items clause contained in the above identified contract. Any reproduction of technical data or portions thereof marked with this legend must also reproduce the markings.
- This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.
- Carnegie Mellon® is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.
- DM23-2173

Agenda

- **Attestations Basics**
- Available Tooling
- How to use attestations
- Example implementation
- Examples in-action

- Presentation Name

Attestation Basics



ATTESTATION

Attestation

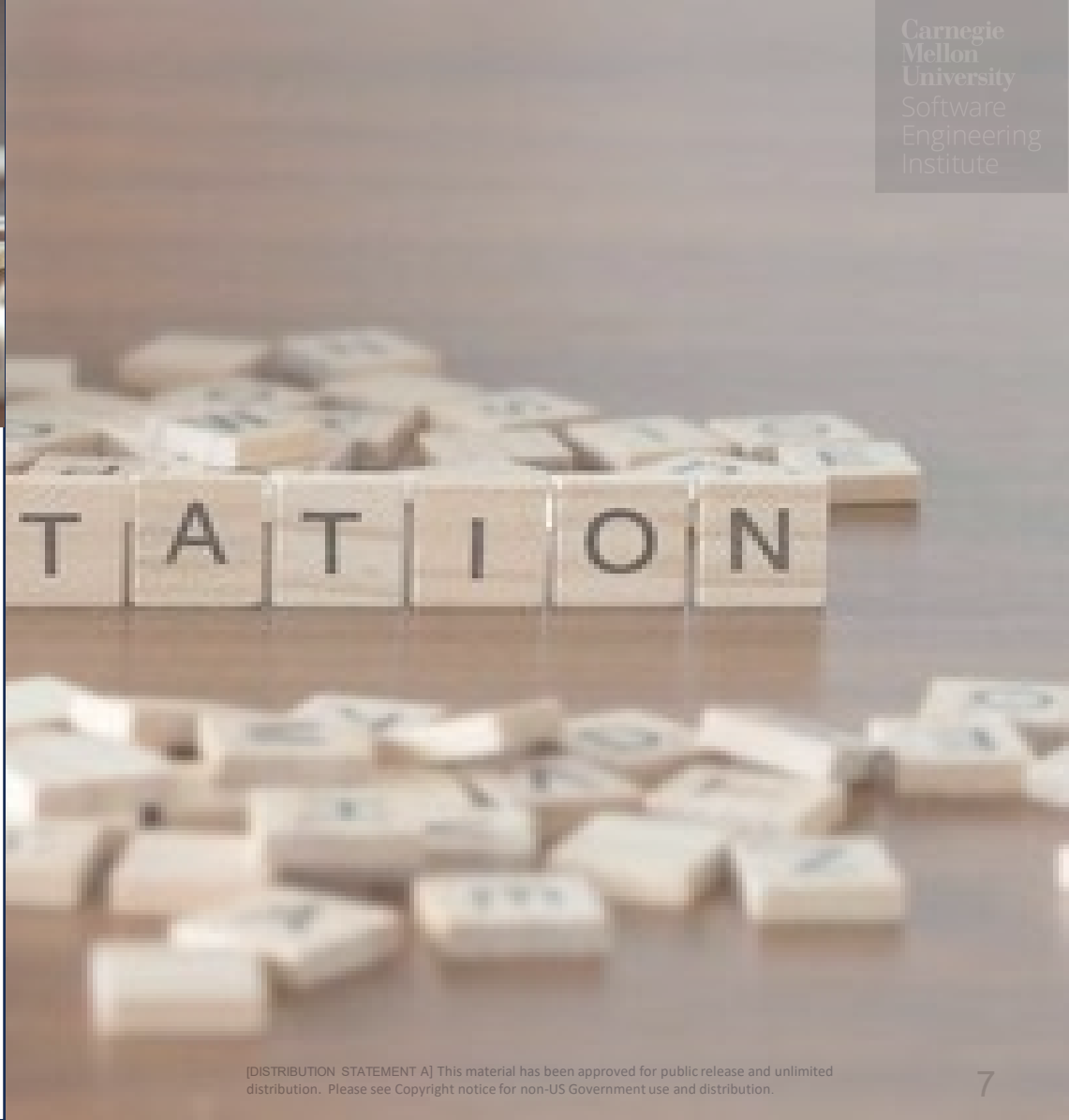
- Evidence or proof of something
- A declaration that something exists or is the case
- The action of being a witness to or formally certifying something [1]

[1] Oxford Dictionary

Software Attestation

- A mechanism that allows a verifier (i.e., Government) to independently validate the integrity of something asserted by the provider (i.e., the vendor).
- For example, an attestation can verify that a software system has been through an agreed-upon process bakes in quality and security. (i.e., a CI/CD pipeline)





In Sept 2022, the white house issued a memorandum impacting all software vendors who sell applications to US government and agencies.

Key Points:

- Agencies are required to obtain a self-attestation from the software vendor before using the software.
- Executive Order 14028, Improving the Nation's Cybersecurity, and NIST Guidance will be followed (SP 800-218, NIST Software Supply Chain Security Guidance)
- Agencies will be required to obtain SBOMs from their software producers
- The Cybersecurity & Infrastructure Security Agency (CISA) will provide a standard common form for self-attestation along with a plan for a government-wide repository for software attestations.

What does a software attestation contain?

- The software producer's name and address
- The product's name, version number, and release/publish date
- A *digital signature* verifying that the software was built using secure development processes
- Other Metadata:
 - User specific signature
 - stdout, stderr
 - Existing files
 - Created files w/sha256
 - Command used
 - Unit-test commands
 - Static analysis commands
 - Build commands

- Presentation Name

Available Tools

Available Tools

- **SLSA**
 - Open-Source Supply Chain Framework, pronounced “Salsa”
 - Supply-chain Levels for Software Artifacts (SLSA)
- **in-toto**
 - Open-Source framework to protect the integrity of the supply chain, used by SLSA
 - Attestations through a metadata standard/specification.
 - Enhancements (ITE), community-driven with support via NSF, DARPA, AFRL
 - Tooling in python, java, go (Focused tool for this presentation)
- **Binary Authorization**
 - Google based Kubernetes solution to verify attestations of containers
- **Witness**
 - Open-Source
 - Attestation generator/Verify – implements in-toto ITE-5, ITE-6, ITE-7

- Presentation Name

How to use attestations



How to use attestations

- Define a series of steps (e.g. CI/CD pipeline) and determine:
 - Who will execute commands?
 - What commands to execute?
 - What materials exist?
 - What products come from the command?
- Determine where to include inspections
- Build a layout, a policy-as-code metadata file
 - in-toto layout
 - Witness policy

In-toto Layout

A layout is where you define the details of steps will occur when building software and by who. It also includes:

- A **readme** (optional description of the layout details)
- **Expiration** – date when the layout expires
- **Functionaries** – functionaries are those who are authorized to perform the steps defined in the layout
- **Signatures** – cryptographic keys of the functionaries (public keys) are added to each step that they will execute, and the layout is signed with the private key of the layout owner.
- **Pipeline steps and Inspections** – inspections are the commands that are run during the verification process which can also list products and materials

Definitions

Artifacts

In-toto uses the term artifact to describe files, source code, binaries, packages, etc, that make up parts of the software supply chain.

Materials

These are the artifacts that are used when a step or inspection is carried out.
(e.g. Source code files)

Products

These are the artifacts that are created after executing a step.
(e.g. An executable created from source-code compilation)

The layout provides a rule language to authorize or enforce the artifacts of a step and to chain them together. This adds these guarantees for any given step or inspection:

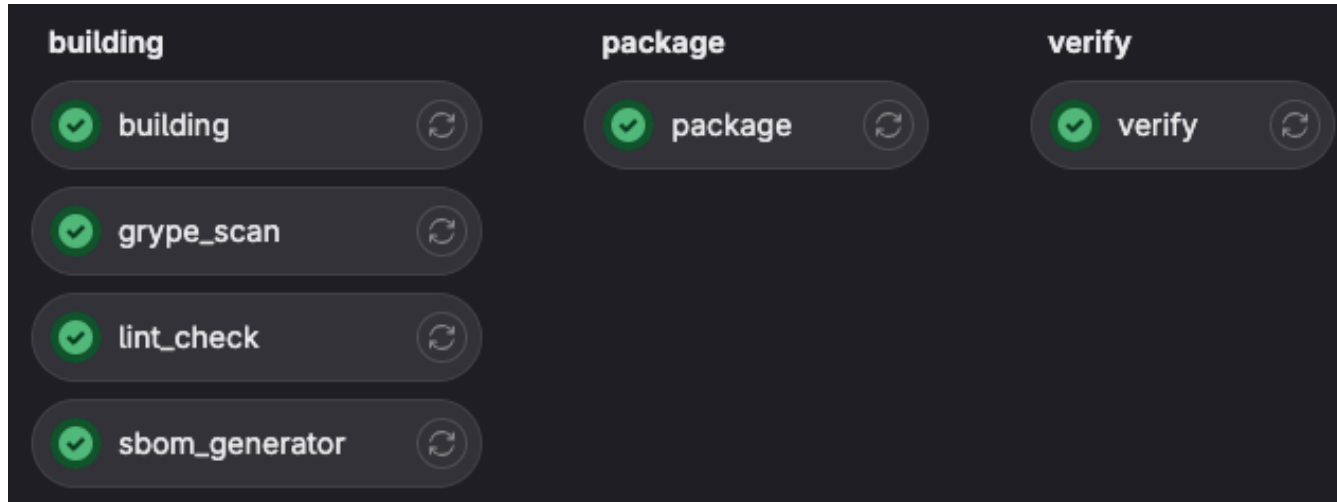
- Only artifacts authorized by the project owner are created, modified, or deleted
- Each defined creation, modification or deletion is enforced and also
- Restricted to the scope of its definition, which chains subsequent steps and inspections together
- CREATE, DELETE, MODIFY, ALLOW, DISALLOW, REQUIRE, MATCH



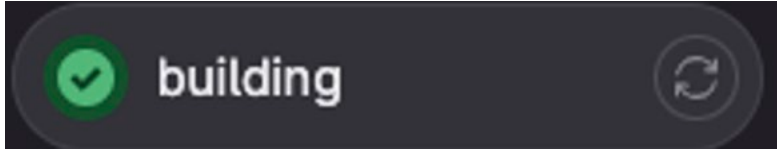
- Presentation Name

Example Implementation

Example Implementation



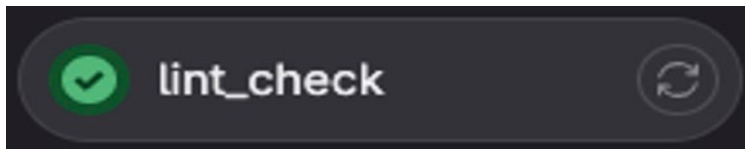
Building



- In-toto layout (JSON)
 - Compile source code
 - Allow the command 'cargo build'
 - Allow materials such as 'src'
 - Enforce functionary 'gitlab'

```
...  
"steps": [  
  {  
    "name": "building",  
    "expected_command": ["cargo", "build"],  
    "expected_materials": ["ALLOW", "src/*"],  
    "expected_products" : [["CREATE", "target/*"],  
    "pubkeys": [ key_gitlab["keyid" ] ,  
    "threshold": 1  
  },  
...  
]
```

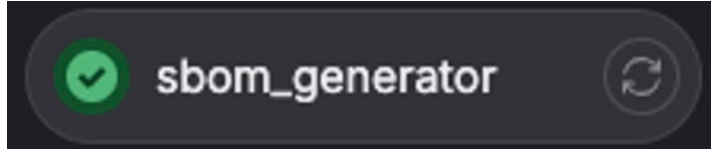
Lint Check



- In-toto layout (JSON)
- Lint Check – enforce static code quality

```
...  
"steps": [{  
  "name": "lint_check",  
  "expected_command": ["cargo", "clippy"],  
  "expected_materials": [  
    ["MATCH", "target/*", "WITH", "PRODUCTS", "FROM",  
      "building"]],  
  "expected_products": [  
    ["MATCH", "*", "WITH", "MATERIALS", "FROM", "building"],  
    ["MATCH", "*", "WITH", "PRODUCTS", "FROM", "building"],  
    ["CREATE", "target/*"]],  
  "pubkeys": [key_gitlab["keyid"] ],  
  "threshold": 1  
},  
...  
...
```

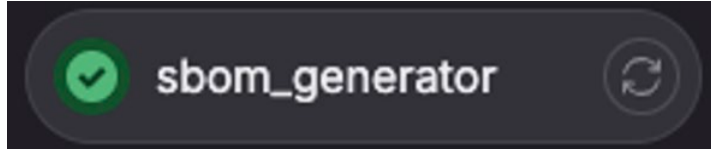
SBOM Generate



- In-toto layout (JSON)
- Generate an SBOM

```
...  
"steps": [{  
  "name": "spdx_sbom_generator",  
  "_type": "step",  
  "expected_command": ['syft', './Cargo.lock', '-o', 'spdx-  
json', '--scope', 'all-layers', '--file', 'spdx.bom.json'],  
  "expected_materials": [ ... ],  
  "expected_products": [ ... ],  
  "pubkeys": [ ... ],  
  "threshold": 1  
},  
...  
...
```

SBOM Scan



- In-toto layout (JSON)
- Analyze SBOM

```
...  
"steps": [{  
  "name": "sbom_check",  
  "_type": "step",  
  "expected_command": ["grype", "spdx.bom.json", "-f",  
"low"],  
  "expected_materials": [ ... ],  
  "expected_products": [ ... ],  
  "pubkeys": [ ... ],  
  "threshold": 1  
},  
...  
...
```

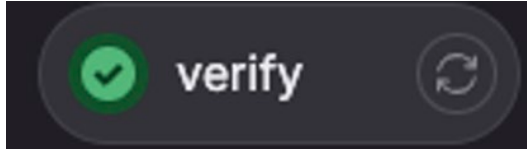
Package



- In-toto layout (JSON)
- Package the product for delivery in future steps

```
...  
"steps": [{  
  "name": "package",  
  "_type": "step",  
  "expected_command": ["cargo", "publish"],  
  "expected_materials": [ ... ],  
  "expected_products": [ ... ],  
  "pubkeys": [ ... ],  
  "threshold": 1  
},  
...  
...
```

Inspection



- In-toto layout (JSON)
- Verify that artifacts from previous step are present and have not been modified

```
...  
"inspect": [{  
  "name": "untar",  
  "expected_materials": [  
    ["MATCH", "target/debug/supplychain-sandbox", "WITH",  
     "PRODUCTS", "FROM", "building"]  
  ]  
}]  
...
```



Contact



Joseph Yankel

DevSecOps Initiative Lead

Software Engineering Institute at Carnegie Mellon University

Email: jdyanke@sei.cmu.edu



Brent Clausner

DevSecOps Engineer

Software Engineering Institute at Carnegie Mellon University

Email: beclusner@sei.cmu.edu