FOSDEM

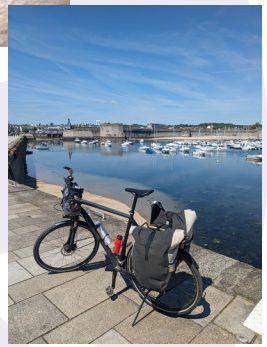
IOT BZH

Lessons learned from integrating SBOM in a supply chain

Sébastien Douheret – February 2nd 2025

About me

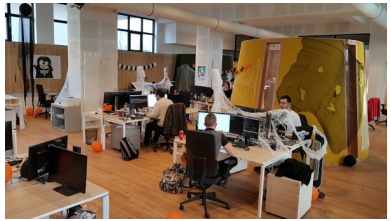
- Technical Director @ IoT.bzh
- Previous/current lives :
  - Uboot + RTOSes (incl. VxWorks)
  - Cloud and embedded software developer (WindRiver, Linux)
- Linux since 2002 (openSUSE, Kubuntu, Fedora, ...)
- sebastien@iot.bzh
- https://www.linkedin.com/in/sebastien-douheret/



IoT.bzh at a glance

Our location

Brittany



European **C**yber**S**ecurity **O**rganisation:
Cyber Valleys mapping

30 years of embedded OS

Wind River (1990) - Intel (2009) - IoT.bzh (2015)

VxWorks®
WIND RIVER

Open Source contributions

TIZEN
SAMSUNG

OS open source, Samsung TVs
Intel Vannes (2011-2015)

AUTOMOTIVE GRADE LINUX
THE LINUX FOUNDATION

Open Source OS for Toyota, Suzuki, Subaru
IoT.bzh: +50% technical contributions 2016-2020

Our product

redpesk®: SaaS platform (or On Prem) Linux for industrial IoT (auto, mil-aero, energy...)

redpesk®

Some partners

Valeo
IDEMIA
RENESAS
SAFRAN
STELLANTIS
TotalEnergies
THALES
PIRIOU
THE LINUX FOUNDATION
BENETEAU

What is SBOM and Why ?

Definition :

"is a formal, machine-readable inventory of software components, and their hierarchical relationships"

Why SBOM is crucial :

- **Enhanced Cyber security** – help to identify vulnerabilities
- **Transparency and Risk Management** – clear view of all components in a software product.
- **Efficient Vulnerability Response** - quickly identify affected components.
- **Supply Chain Security** - helping stakeholders identify potential risks.
- **Compliance and regulatory adherence** - legal obligations arising from European directives (NIS 2 | ANSSI and Cyber Resilience Act).

C. R. A.

CYBER RESILIENCE ACT

- **Legacy date** – 2022/01/01 :
  Products released before this date are excluded from EU CRA, if they didn't undergo any substantial modifications after this date
- **Effective date** – **2024/12/10**: The EU CRA enters into force
- **Notification date** – **2026/09/11** (21 months after effective date):
  Manufactures must notify the relevant authorities about exploitable and severe vulnerabilities
- **Penalty date** – **2027/12/11** (36 months after effective date):
  The EU may charge manufacturers with penalties for violations of the EU CRA

https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=OJ:L_202402847

Types of BOMs

Acronym	Full name	Description
CBOM	Cryptography Bill of Materials	Describe cryptographic assets and their dependencies in software and systems
SAASBOM	Software-as-a-Service Bill of Materials	Offers a list of endpoints, data flows, classifications, and services involved in cloud-native applications
ML-BOM	Machine Learning Bill of Materials	Documents AI technologies within a product, including datasets, training methodologies, and AI framework configurations
HBOM	Hardware Bill of Materials	Captures detailed inventories of physical hardware components and associated firmware in a product
mBOM	Manufacturing Bill of Materials	Lists all assemblies, parts, and materials required to manufacture a finished product
VEX	Vulnerabily Exploitability eXchange	A standard format for communicating the status of vulnerabilities in software component
VDR	Vulnerabily Disclosure Reports	Detailed reports used to communicate information about discovered vulnerabilities to relevant parties
non exhaustive list ...	...	...

Most of them handled by CycloneDx https://cyclonedx.org/capabilities/sbom/

Popular SBOM Formats

Linux Foundation

- Initially designed to track software licenses
- Evolved to include file integrity and vulnerability tracking
- Versions <= 2.x : monolithic approach
  Versions >= 3.x : more flexible
- Became an official ISO/IEC standard in August 2021

OWASP Foundation

- Initially focused on tracking software vulnerabilities (security)
- Adopts a lightweight, extension-based approach
- Widely used across all sectors
- Support complex multi-modal systems description

Competition between 2 BOMs formats but no “better” format between these two

SBOM and VEX

- VEX - Vulnerability Exploitability eXchange

  is the exploitability status of a component in relation to one or more vulnerabilities

- Importance of BOMs combination :
  - SBOM : packages identification, version, licensing
  - VEX / CVE : huge importance to be compliant with directives like NIS2, CRA, …

redpesk® embedded software for IoT

redpesk OS

- *LTS* version based on RHEL *devel* version based on CentOS Stream
- Based on RPM packages
- BSP (Board Support Package) allowing to support various embedded boards
- Enriched by µservices & security frameworks

Sources available at
https://github.com/redpesk

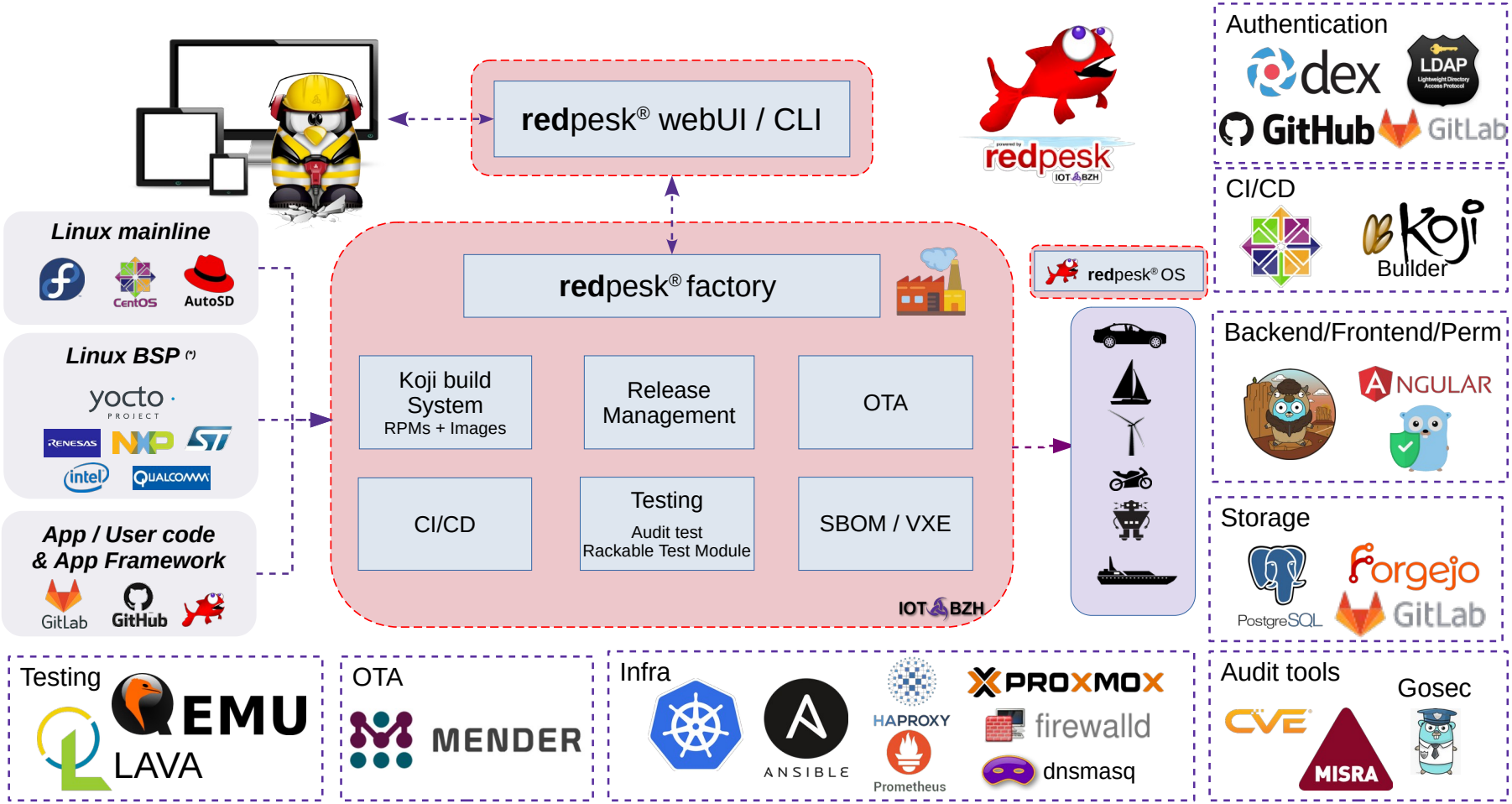
redpesk Factory

- Ease development and integration workflows in cross environment
- CI / CD : automatic rebuild, testing
- Based on Koji (Fedora build system) with extensions to support cross-building and emulate build

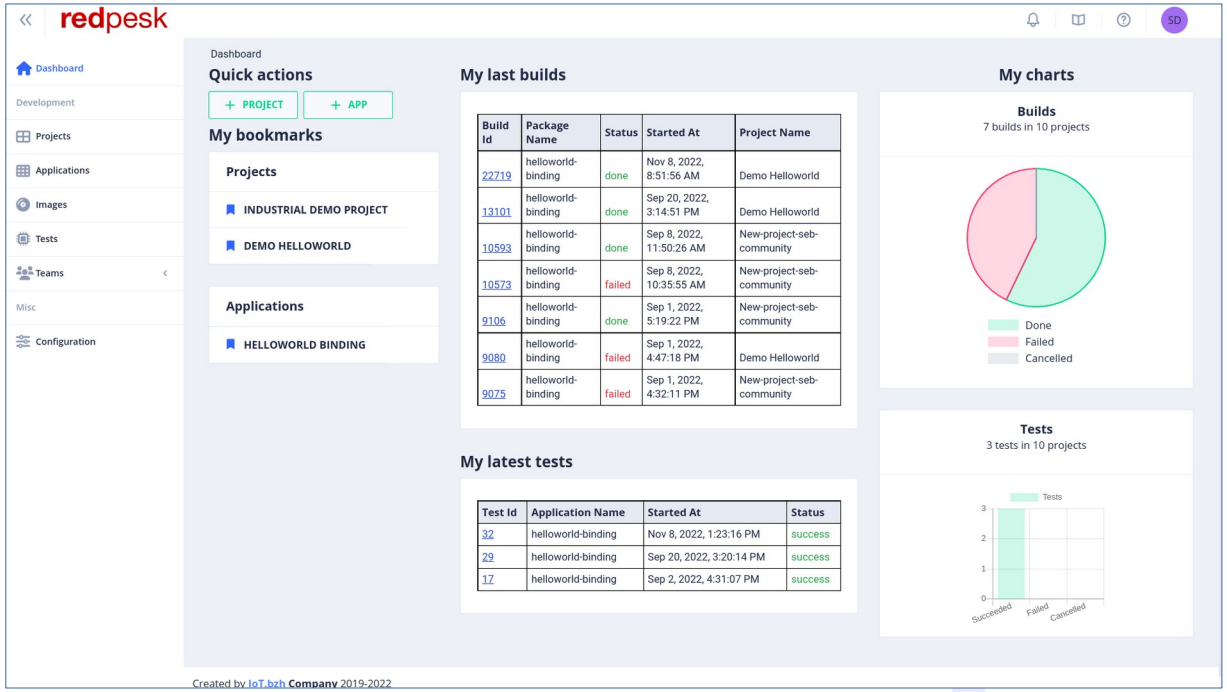
Community edition
https://community-app.redpesk.bzh

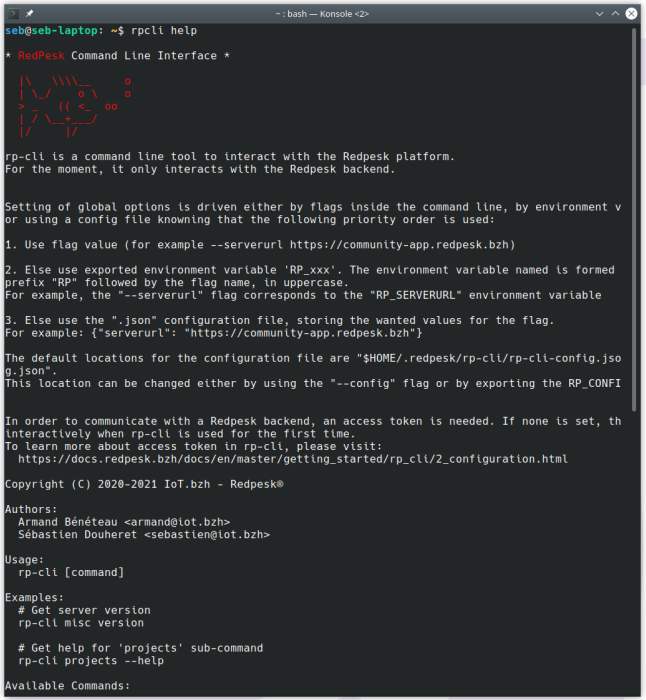
Such a complex supply chain needs to provide SBOM+VEX reports

redpesk® factory based on proven tools



redpesk® factory WebUI & CLI





3 ways to interact with the factory :

- Web UI
- Command Line (rp-cli)
- REST API and WebSocket (optional)

Integration challenges - SBOM

- identify where and how to collect relevent data
  ⇒ don’t re-invent the wheel but extract/capitalize on existing information (RPM, image manifest file,...)

- Merge or integrate SBOMs and artifacts generated externaly
  ⇒ [rust] `cargo sbom`, [go] `syft / cyclonedx-go`, [nodejs] `npm sbom`, …

- adjustement needed to support all corner cases
  ⇒ concrete example: on-going Fedora license SPDX ID migration

Integration challenges - VEX

- redpesk baseOS : relies on RedHat security database (CVE) but with additional patches

  ⇒ support of cross-compilation or fixes due to embedded constraints

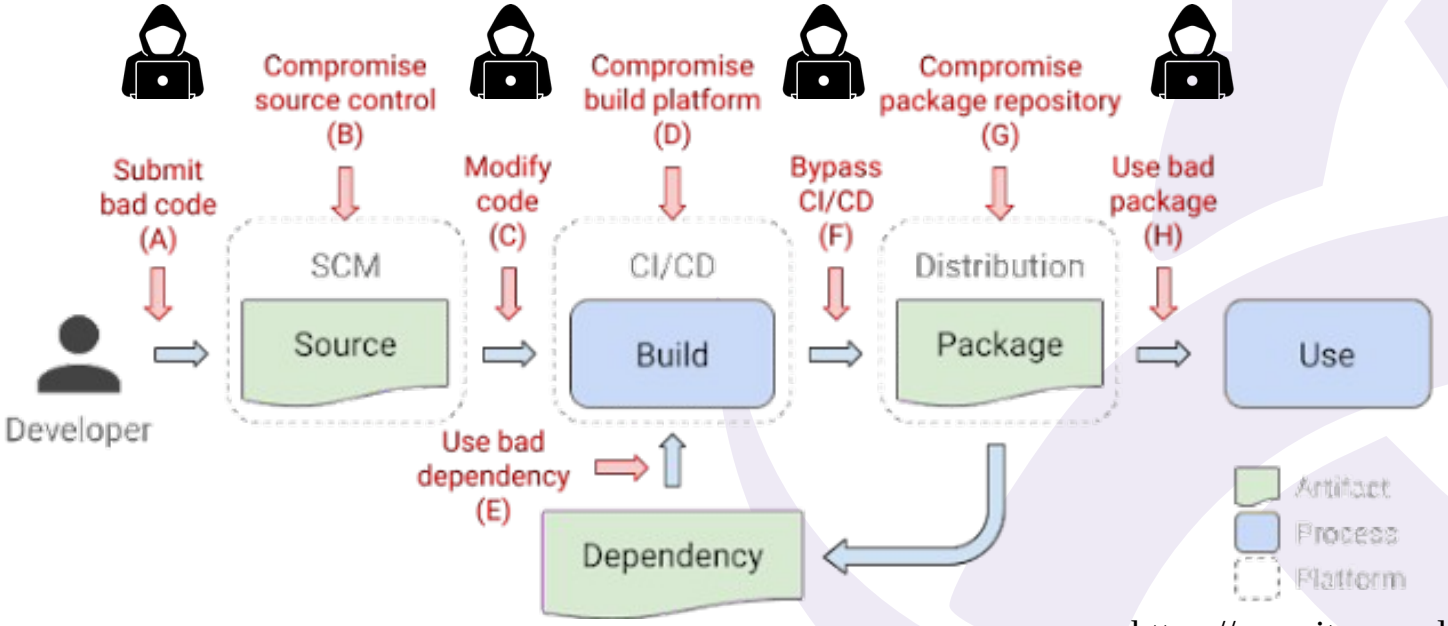
  ⇒ Data Accuracy and Reliability : Importance of regular updates

⇒ setup specific database and micro-service to handle this situation

BOM files generation that’s good but not enough !

No guarantee whether a package has been tampered or not by a malicious user

SLSA provenance attestations + in-toto



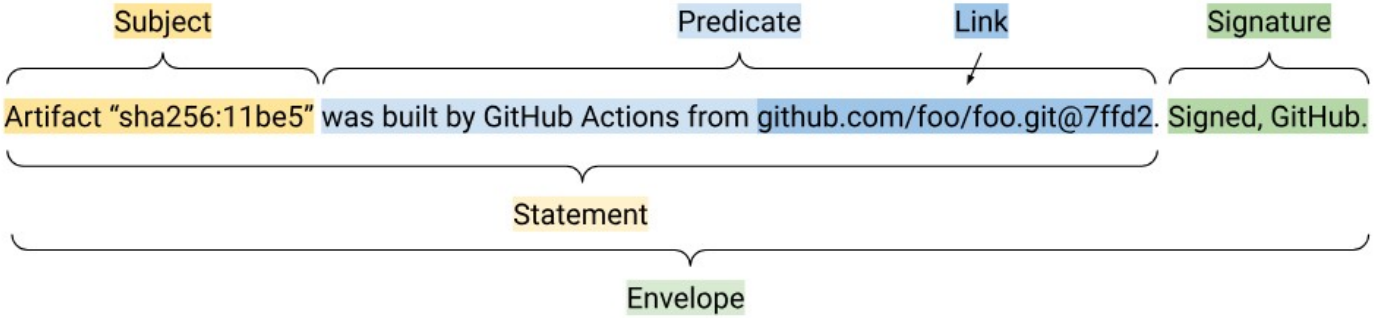
https://security.googleblog.com/

Supply-chain Levels for Software Artifacts

SLSA (pronounced “salsa”)

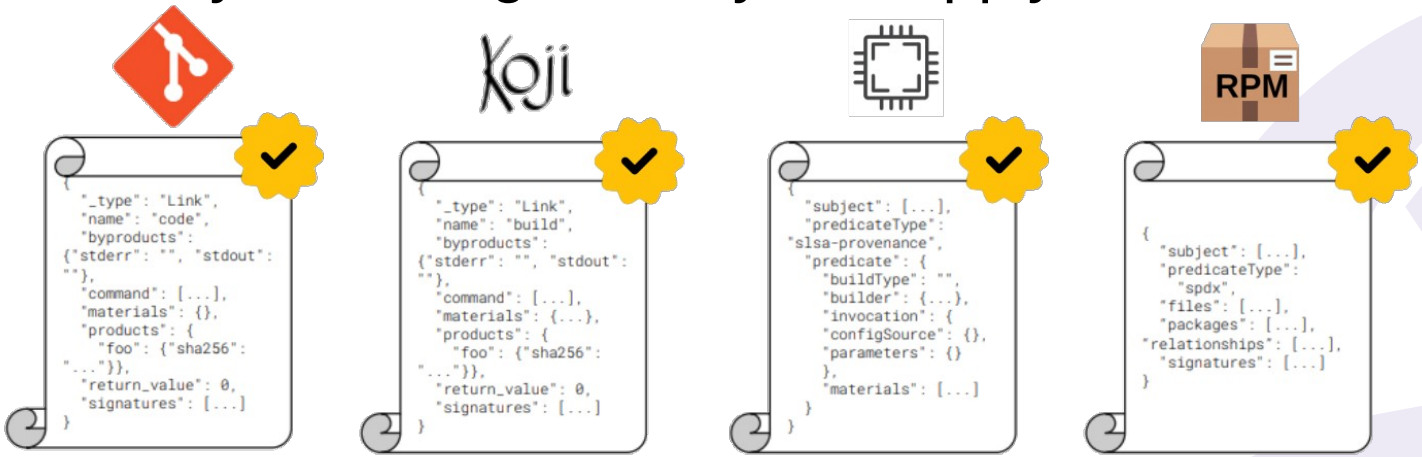
Google initiative (2021) and now under OpenSSF umbrella
It’s a security **framework specifically designed to ensure the integrity** of software artifacts.

- **Artifact**: Immutable blob of data described by an attestation, usually identified by cryptographic content hash.
- **Attestation**: Authenticated, machine-readable metadata about one or more software artifacts. Contain at least an envelope (attestation + signature) and a statement (Subject + predicate)
- **Predicate**: Arbitrary metadata in a predicate-specific schema (ex: link)
- **Bundle**: A collection of Attestations, which are usually but not necessarily related.
- **Storage**/Lookup: where/how verifiers find attestations for a given artifact.



In-toto

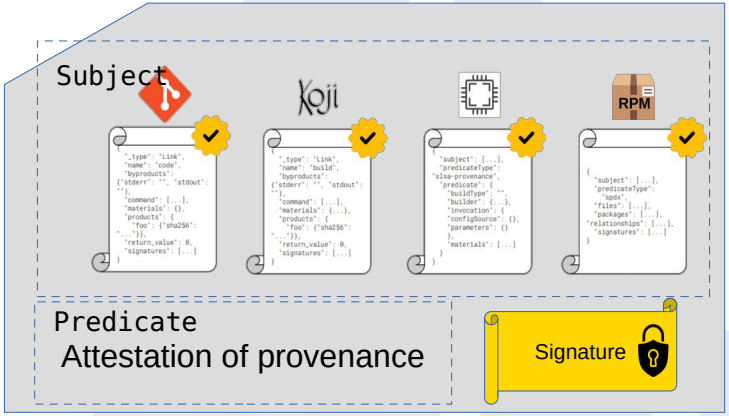
The way to manage all of your supply chain metadata

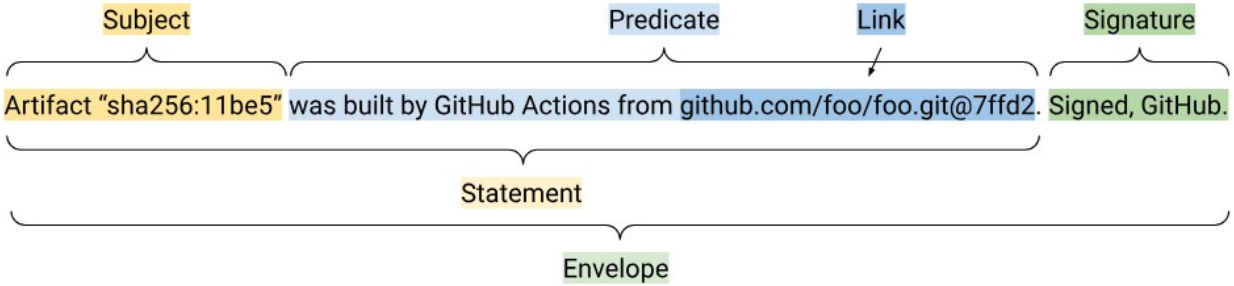


- Think of in-toto as the common “language” for all things software supply chain security.
- SLSA recommends using in-toto attestations as the vehicle to express Provenance and other attributes of software supply chains.

SLSA provenance attestations + in-toto adding trust to BOMs & artifacts

- A SLSA provenance attestation is an in-toto attestation of a certain type.
- An in-toto attestation is made of different nested parts:
  - an **Envelope** that contains a payload and its associated signature
  - a **Statement** that associates a Subject (e.g. an artifact) to a Predicate
- Build platforms (e.g. redpesk factory) must specify and define the relevant External Parameters and their meanings.
- **External Parameters** allow verifiers to make sure an artifact and its associated provenance attestation are the legitimate ones.
- Two community-maintained build types are currently available:
  - GitHub Actions Workflow [#1]
  - Triggered Google Cloud Build [#2]





SLSA + In-toto real example

Taking the python-urllib3 package, here is a
in-toto attestation including slsa provenance predicate :


```
1  {
2      "_type": "https://in-toto.io/Statement/v1",
3      "predicateType": "https://slsa.dev/provenance/v1",
4      "predicate": {
5          "buildDefinition": {
6              "buildType": "https://redpesk.bzh/build-workflow/v1",
7              "externalParameters": {
8                  "stack": "distro",
9                  "project": "apps_f5039dc1",
10                 "application": "python-urllib3_722229d5"
11             },
12             "internalParameters": {},
13             "resolvedDependencies": []
14         },
15         "runDetails": {
16             "builder": {
17                 "id": "https://distro-app-next.lorient.iot"
18             },
19             "metadata": {
20                 "invocationId": "https://distro-app-next.lorient.iot/#/p
                   apps_f5039dc1/applications/python-urllib3_722229d5/app-b
21                 "startedOn": "2024-09-05T10:11:40Z",
22                 "downloadUrl": "https://download.redpesk.bzh/redpesk-lts/batz-2.1-update/package
                   aarch64/os/Packages/p/python3-urllib3-1.26.5-6.apps.rpbatz_1.1.noarch.rpm"
23             }
24         }
25     },
26     "subject": [
27         {
28             "name": "python3-urllib3-1.26.5-6.apps.rpbatz_1.1.noarch.rpm",
29             "digest": {
30                 "sha256": "73ce20546204e2b895895403b16d014be9929bc7a584ef591664bacab531e497"
31             }
32         }
33     ]
34 }
```


- **Tool - cosign** : a tool that allows to sign and verify signatures
- **Infra - sigstore** : keyless signing and verification (based on transparency model)

```
cosign attest-blob
  --key redpesk_factory.key
  --predicate redpesk_factory_predicate.json
  --type slsaprovenance1 -y
  --output-signature attestation_with_key.intoto.json
  python3-urllib3-1.26.5-6.apps.rpbatz_1.1.noarch.rpm
```

cosign

Resulting **signature**: attestation_with_key.intoto.json

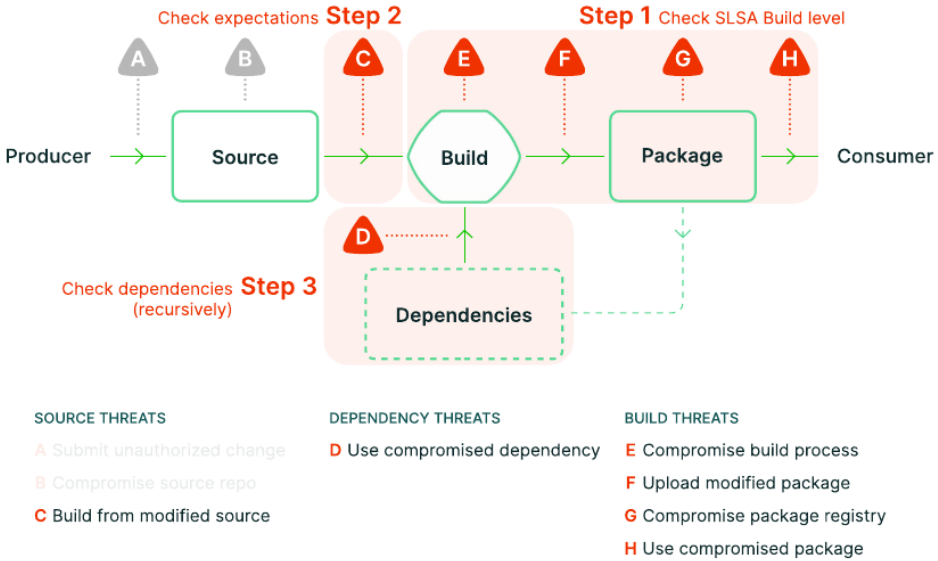

```
1  {
   "payloadType": "application/vnd.in-toto+json",
   "payload":
   "eyJfdHlwZSI6Imh0dHBzOi8vaW4tdG90by5pby9TdGF0ZW1lbnQvdjAuMSIsInByZWRp
   Y2F0ZVR5cGUiOiJodHRwczovL3Nsc2EuZGV2L3Byb3ZlbmFuY2UvdjEiLCJzdWJqZWN0I
   jpbeyJuYW1lIjoicHl0a...
   InNoYTI1NiI6IjczY2UyMDU0NjIwNGUyYjg5NTg5NTQwM2Ix
   Tg0ZWY1OTE0MFoifX19fQ==",
4  "signatures": [
5      {
6          "keyid": "",
7          "sig": "MEYCIQCIN7pqElS
           +q2J7tVLkl9ZU6omwCj5Isp3De6tExDAMbgIhAMe8TWw48w8HuwQF4K0Q0Ijs
           bjaU183d79ZZAOqvpr8a"
8      }
9  ]
10 }
```


Send to ledger

sigstore

Verify an attestation

1. check the cryptographic signatures and the chain of trust,
2. check that the provenance meets expectations about the source,
3. (optional): check recursively dependencies



- Existing tools :
  - slsa-verifier [#3]
  - cosign [#4]
- ease the 1st step of verification by checking that a signed attestation matches either with a given public key or an OpenID Connect identity.
- But in our specific context, step 2 is not covered (verification process) !

Verify example

- verify the attestation signature with the corresponding public key

```
cosign verify-blob-attestation \
    --signature attestation_with_key.intoto.json \
    --key redpesk.pub \
    --type=slsaprovenance1 \
    --verbose\
    python3-urllib3-1.26.5-6.apps.rpbatz_1.1.noarch.rpm
Verified OK
```

- **manually** verify build platform parameters :

```
$ jq -r '.payload' attestation_keyless.intoto.json | \
    base64 -d | \
    jq '.predicate.buildDefinition.externalParameters'
{
    "application": "python-urllib3_722229d5",
    "project": "apps_f5039dc1",
    "stack": "distro"
}
```

SLSA / in-toto / cosign / sigstore challenges

Credit: Pix4Free.org

- Most of examples and existing attestations of provenance use https://github.com/slsa-framework/slsa-github-generator

  and consequently are specific to Github.

- Therefore provenance checking tools are also Github oriented !
  - slsa-verifier : hard-coded support for Github Actions and Google Cloud Build https://github.com/slsa-framework/slsa-verifier/issues/734
  - Cosign : container images oriented, low support for generic blobs (eg. build platform parameters)

Summarize

- Long and complex work to support all use cases
- Multiple formats (eg. SPDX / CycloneDX) don’t simplify implementation
- Not an easy task if you don’t want to rely on Github
- Publish our specific use cases (without github) in order to open discussions

Simple use case of SBOM report for images build available (mid-February)
in next redpesk Factory armel 1.8 !
Feel free to test with Community Edition https://community-app.redpesk.bzh/
Integration of VEX will be available in next version armel 1.9 (July)

Q&A

Lorient Harbour, South Brittany, France

Links

- Cyber Resilient Act https://en.wikipedia.org/wiki/Cyber_Resilience_Act
  https://digital-strategy.ec.europa.eu/en/library/cyber-resilience-act
- Directive NIS-2 https://cyber.gouv.fr/la-directive-nis-2
- SPDX https://spdx.github.io/spdx-spec/v3.0.1/
  https://github.com/spdx
- CycloneDX https://cyclonedx.org/
  https://github.com/cyclonedx
- SLSA https://slsa.dev/
- slsa-verifier https://github.com/slsa-framework/slsa-verifier
- In-toto https://in-toto.io/
- Cosign https://github.com/sigstore/cosign
- CUE https://cuelang.org/
- Rego https://www.openpolicyagent.org/docs/latest/policy-language/

Links

- redpesk:
  - Website: https://www.redpesk.bzh
  - Documentation: https://docs.redpesk.bzh
  - Sources: https://github.com/redpesk

- IoT.bzh:
  - Website: https://iot.bzh/
  - Articles: https://iot.bzh/articles

Annexes

sigstore - rektor

cosign also uploads some metadata to a public immutable ledger that can be audited by anyone.
Here an example of a the rekor transparent log : https://search.sigstore.dev/?logIndex=153334293

sigstore rekor

Rekor Search

Attribute: Log Index

Log Index: 153334293

SEARCH

Showing 1 of 1

Entry UUID: 108e9186e8c5677a2232d4bb45a7250f797fd8e2f72e987b33564650a0ad8eab3110b4cd204827e3

TYPE	LOG INDEX	INTEGRATED TIME
dsse	153334293	2 months ago (2024-12-04T15:59:38+01:00)

Hash

```
sha256:f692ddd998165d93f43a7e0d2efb23cbefade3bcca416371de5a8e9c4bb8ae4c
```

Signature

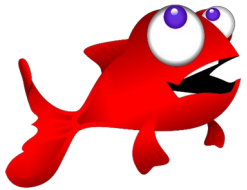
```
MEYCIQDtCwEQoD5/NNwal3LldIuvayPfMoH7NRG5et5Y3kQ6mgIhAM0v3gcj71151z7fS1Wu7fjKwcy+TJuopugs+PZq+fif
```

Public Key

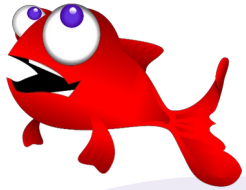
```
-----BEGIN PUBLIC KEY-----
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEnhtMembC4QmTsFanX5qdYmVGBFFt
aFIuKYMIc2WbW2OXOxgXlftJe3ARDiuqxGy42dOMPG09t5LZxBYMCKJOOQ==
-----END PUBLIC KEY-----
```

Raw Body

Verification



a redpesk fish has two sides



redpesk OS

1. *LTS* version based on RHEL *devel* version based on CentOS Stream
2. Enriched by microservices and security frameworks
3. Multiple SoC vendors BSPs are supported
4. Light containers support: redpak
5. Zephyr/RTOS support

redpesk Factory

1. Ease development and integration workflows in cross environment
2. Project/apps management and integration through webUI and CLI
3. Supports developers, integrators, QA engineers, delivery managers
4. Manage multiple projects with a clear hierarchy

Short development cycle - localbuilder

1. Goal:
   - ease developper day to day work (local edition)
   - but still maintain projects & applications in CI/CD factory
2. Solution: rp-cli and localbuilder container (including SDK) running on developer machine
3. Restriction: only for development, unsigned packages