
KubeCon
CloudNativeCon
Europe 2025
LONDON BUS

Enhancing Software Composition Analysis Resilience Against Container Image Obfuscation

Agathe Blaise, Jacopo Bufalino

#whoarewe

Agathe Blaise

Research engineer

Thales SIX GTS France

Jacopo Bufalino

Ph.D. Candidate

Cnam, Paris & Aalto University, Helsinki

European project funding

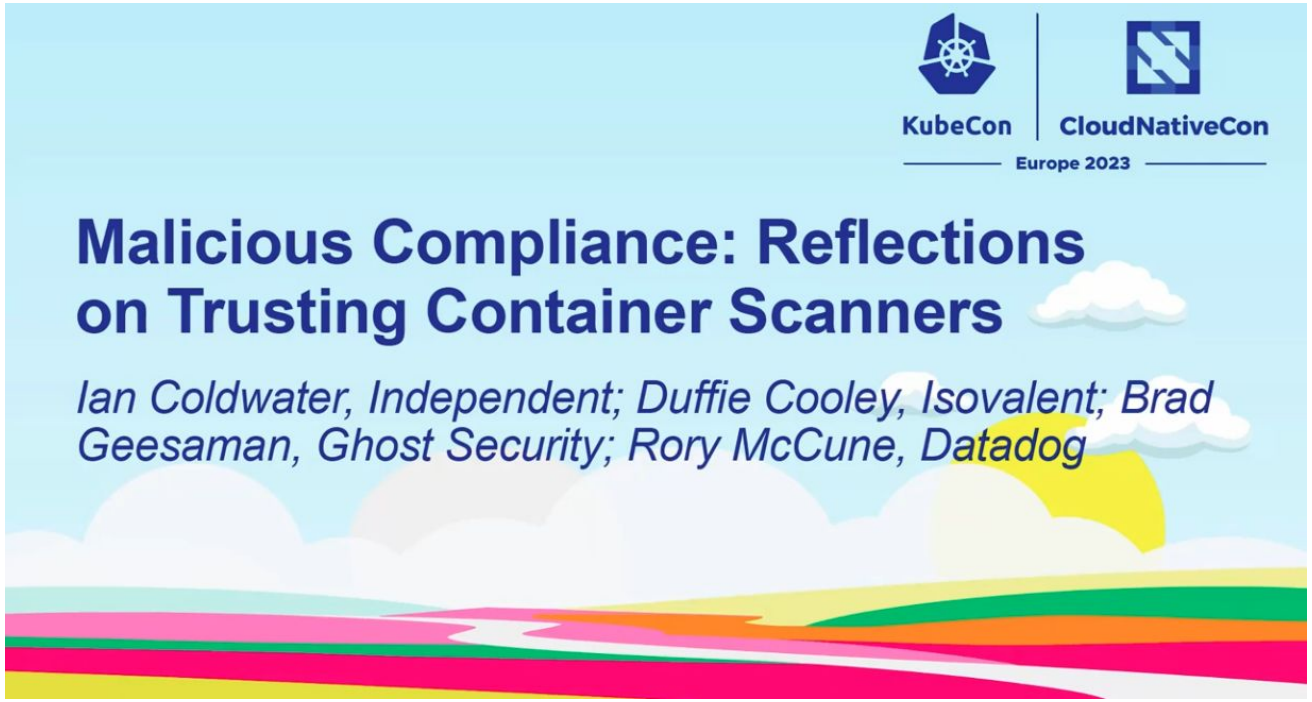
SEC 4AI 4 SEC

Secure AI-enhanced systems and
AI-enhanced Systems for Security
(**Sec4AI4Sec**)

https://www.sec4ai4sec-project.eu/

Motivation

KubeCon 2023: introduced the concept of malicious compliance and container obfuscation



→ **How the landscape evolved in the past two years?**
Academic approach to the problem: classification and analysis of obfuscation for different containers/registries and for different tools

KubeCon | CloudNativeCon
Europe 2025

State-of-the-art container vulnerability analysis

Container vulnerability analysis

Step 1: Container file-system indexing

- Analyze the container filesystem
- Find installed software using Software Composition Analysis (SCA)
- Generate a Software Bill of Materials (SBOM)

Step 2: Search vulnerabilities in the indexed software

- Match indexed software against known vulnerabilities
- Leverage custom vulnerability databases and aggregate multiple information sources

Step 1 - Finding installed software


```
syft python:3.10 --output json > sbom.json
```


```
{
  "artifacts": [
    {
      "id": "adbd50b08f319423",
      "name": "Simple Launcher",
      "version": "1.1.0.14",
      "type": "dotnet",
      "foundBy": "dotnet-portable-executable-cataloger",
      "locations": [
        {
          "path": "/usr/local/lib/python3.10/site-packages/pip/_vendor/distlib/t32.exe",
          "layerID": "sha256:d776b3e6e5bfd1a4f40e00a887ef5ef1ac59006a4cf293354ca49036c959267d"
          "accessPath": "/usr/local/lib/python3.10/site-packages/pip/_vendor/distlib/t32.exe",
          "annotations": {
            "evidence": "primary"
          }
        }
      ],
      "licenses": [],
      "language": "dotnet",
      "cpes": [
        "cpe:2.3:a:Simple_Launcher:Simple_Launcher:1.1.0.14:*:*:*:*:*:*:*"
      ],
      "purl": "pkg:nuget/Simple%20Launcher@1.1.0.14",
      "metadataType": "dotnet-portable-executable-entry",
      "metadata": {
        "assemblyVersion": "",
        "legalCopyright": "Copyright (C) Simple Launcher User",
        "internalName": "t32.exe",
        "companyName": "Simple Launcher User",
        "productName": "Simple Launcher",
        "productVersion": "1.1.0.14"
      }
    },
```


Path of the file used for finding the software

Layer ID where the file is

Package name under the **CPE** notation

Package name under the package-URL (**PURL**) notation

Metadata information on the package

Step 2 - Search for vulnerabilities

KubeCon CloudNativeCon Europe 2025

```
grype sbom:sbom.json
```

python	3.10.16		binary	CVE-2023-36632	High
python	3.10.16	3.11.9, 3.12.3, 3.13.0a5	binary	CVE-2025-1795	Unknown
python	3.10.16	3.12.9, 3.13.2, 3.14.0a5	binary	CVE-2025-0938	Unknown
python	3.10.16	3.14.0	binary	CVE-2024-3220	Unknown
python3.11	3.11.2-6+deb12u5	(won't fix)	deb	CVE-2025-1795	Unknown
python3.11	3.11.2-6+deb12u5	(won't fix)	deb	CVE-2025-0938	Unknown
python3.11-minimal	3.11.2-6+deb12u5	(won't fix)	deb	CVE-2025-1795	Unknown
python3.11-minimal	3.11.2-6+deb12u5	(won't fix)	deb	CVE-2025-0938	Unknown
setuptools	65.5.1	70.0.0	python	GHSA-cx63-2mw6-8hw5	High
tar	1.34+dfsg-1.2+deb12u1		deb	CVE-2005-2541	Negligible
tcl8.6	8.6.13+dfsg-2		deb	CVE-2021-35331	Negligible
tcl8.6-dev	8.6.13+dfsg-2		deb	CVE-2021-35331	Negligible
unzip	6.0-28		deb	CVE-2021-4217	Negligible
util-linux	2.38.1-5+deb12u3		deb	CVE-2022-0563	Negligible
util-linux-extra	2.38.1-5+deb12u3		deb	CVE-2022-0563	Negligible
uuid-dev	2.38.1-5+deb12u3		deb	CVE-2022-0563	Negligible
wget	1.21.3-1+deb12u1	(won't fix)	deb	CVE-2024-10524	Medium
wget	1.21.3-1+deb12u1	(won't fix)	deb	CVE-2021-31879	Medium
zlib1g	1:1.2.13.dfsg-1	(won't fix)	deb	CVE-2023-45853	Critical
zlib1g-dev	1:1.2.13.dfsg-1	(won't fix)	deb	CVE-2023-45853	Critical

Packages identified at the previous step

Version

Fixes (if available)

Package type

CVE of given package

Severity

Container file-system indexing

What do container scanners index to find packages?

- Operating System (OS)
  `/etc/releases`
- OS package manager metadata
  `var/lib/dpkg/status`
- Programming language dependencies
  `requirements.txt`
  `package-lock.json`
- Programming language package-manager metadata files
  `*.dist-info/metadata`
  `node_modules/*package.json`
- Binary metadata
  `go version -m`

Container file-system indexing

How is the container image content indexed?

1. Analyze the **squashed representation** of the container image → combining all layers (default method)

1. Search for **known filenames** containing package-related information
   - Regex-based search of licences, package version and author.

/!\ What happens if some of these files are removed?

KubeCon | CloudNativeCon
Europe 2025

Definitions

Definitions

Container Obfuscation

Obfuscation is the act of **intentionally or unintentionally** (without malicious intent) **modifying**/generating the content of a container image in such a way that the **installed software is partially or totally undetected** by container vulnerability scanners.

Definitions

Container Obfuscation

Obfuscation is the act of **intentionally or unintentionally** (without malicious intent) **modifying**/generating the content of a container image in such a way that the **installed software is partially or totally undetected** by container vulnerability scanners.

For **various reasons**:

- Minimizing the container image to reduce size & complexity
- Removing installation folders
- Using multistage builds

KubeCon | CloudNativeCon
Europe 2025

Our objectives

Our objectives

We will cover **4 main objectives** during this presentation:

- What are the **different obfuscation techniques**?
- Are **state-of-the-art** tools **vulnerable** to obfuscation?
- Are **popular** containers images affected by obfuscation?
- How to **mitigate** obfuscation?

KubeCon | CloudNativeCon
Europe 2025

Objective #1
Taxonomy of obfuscation techniques

Finding obfuscation techniques

1) **Find obfuscation techniques**

   We explored various information sources such as:

   - Research papers
   - Whitepapers
   - Industry standards
   - Open Source Software (OSS) tools

2) **Build a taxonomy**

   We categorized and structured obfuscation techniques into a clear taxonomy.

Obfuscation techniques

Tactic ID	*Target*	*Description*
OS	Operating System	Altering files with information about the operating system name and version
OSPKG	Operating System	Altering files or databases with information on the installed OS packages
DEP	Programming language dependency file	Altering files with dependency information of a programming language
PKG	Programming language package	Altering content of installed dependency
URL*	OSPKG and PKG files	Downloading applications without package manager
ALIAS*	Any file in the system	Creating alias to avoid scanning based by filename
LINK	Any file in the system	Creating links to avoid scanning based by filename
PACK*	Every file in the system	Compress the layers of the container

*: novel obfuscation techniques we identified

Objective #2
Is obfuscation detected by state-of-the-art tools?

Container vulnerability scanning tools

We selected 3 open-source software and 3 cloud-based tools.

Tool	*Company*	*Customization**	*Open-source*	*CNCF Project*
Syft + Grype	Anchore	✓	✓	
DockerScout	Docker			
Trivy	Trivy		✓	✓
Artifact registry	Google			
Defender for Cloud	Microsoft		✓	
Inspector	Amazon			

*: can scan every layer independently rather than the the squashed container
→ more resistant to obfuscation

Building our own dataset

Manual curation of a dataset to test the resistance of tools against container obfuscation

→ Start with a **baseline**: official non-obfuscated `python:3.10.0` image
Incrementally apply obfuscation to the image:

- Removing OS, OS package, dependency, or package metadata
- Installing software without using a package manager
- Creating aliases and links to avoid scanning based on filenames
- Compressing all layers of the container
- + Combination of those techniques

Dataset is available here →

Results

Technique(s)	Trivy		Syft		Syft (All)		Scout		Microsoft		Gcloud		Amazon	
	V	P	V	P	V	P	V	P	V	P	V	P	V	P
BASE (no obscuration)	1164	441	625	448	625	448	123	585	154	429	722	441	471	587
OSPKG	6	11	25	25	625	448	10	23	154	429	6	12	N/A	0
URL	1164	441	625	448	625	448	123	585	154	429	722	441	471	587
LINK	1164	441	625	448	625	448	123	585	154	429	722	441	471	587
DEP	1164	441	625	448	625	448	123	585	154	429	722	441	471	579
PKG	1158	430	619	436	625	448	117	573	148	429	716	429	469	576
ALIAS	1164	441	625	448	625	448	123	585	154	429	722	441	471	587
PACK	1164	441	625	448	625	448	123	585	154	429	722	441	471	587
OS	1164	441	9	448	625	448	6	18	154	429	6	12	471	587
OS + OSPKG	6	11	27	25	625	448	6	23	154	429	6	12	N/A	0
DEP + PKG	1158	430	619	436	625	448	117	573	148	429	716	429	465	568
OS + DEP	1164	441	9	448	625	448	6	18	154	429	6	12	471	579
OS + PKG	1158	430	3	436	625	448	0	6	148	429	N/A	0	469	576
OS + OSPKG + PACK	6	12	27	25	27	25	6	23	0	0	6	12	N/A	0
OS + OSPKG + PKG	0	0	21	13	625	448	0	11	148	429	N/A	0	N/A	0
OS + OSPKG + DEP	6	11	27	25	625	448	6	23	154	429	6	12	N/A	0
OS + OSPKG + DEP + LINK	6	11	27	25	625	448	6	23	154	429	6	12	N/A	0
OS + OSPKG + DEP + PKG	0	0	21	13	625	448	0	11	148	429	N/A	0	N/A	0
OS + OSPKG + DEP + PACK	6	12	27	25	27	25	6	23	0	0	6	12	N/A	0
OS + OSPKG + DEP + ALIAS	6	11	25	24	625	448	6	22	154	429	6	12	N/A	0
OS + OSPKG + DEP + ALIAS + PACK	6	12	25	24	27	25	6	22	0	0	6	12	N/A	0
OS + OSPKG + DEP + ALIAS + PKG	0	0	19	12	625	448	0	10	148	429	N/A	0	N/A	0

V / P: **number of vulnerabilities/packages** detected by each tool on each image

1164: identical number of items suggests **no effect of obfuscation** on the tool

6: lower number of detected items compared to baseline indicates **obfuscation**

Analysis of the results (1/2)

Key findings:

- **Significant impact of obfuscation** across multiple cases

- All tools affected by **at least one** obfuscation technique
  - Combined techniques are even more effective
  - Making some tools to fail entirely, reporting zero vulnerabilities

- Cloud-based scanners reject images missing some specific expected files
  - However images are not required to contain OS or OS package

Analysis of the results (2/2)

Additional observation:

- The number of vulnerabilities and packages significantly vary across tools
  - Highly dependent on package name formatting and on how vulnerabilities are counted
  - **Not reliable indicators** to measure a tool's resistance to obfuscation

KubeCon | CloudNativeCon
Europe 2025

Objective #3
Obfuscation in the wild

Obfuscation in the wild

Analysis of obfuscation on 600 container images from 6 datasets

- For each dataset we selected the **100 most downloaded images**
- From 6 different popular registries (DockerHub and others)
- Used for comparison of OSS tools

Dataset	***Size***	***Description***
DockerHub Official	100	Curated images developed in collaboration with software maintainers.
DockerHub Bitnami	100	Images maintained by Bitnami.
DockerHub Verified	100	Images from trusted software publishers.
DockerHub OSS	100	Images published and maintained by open-source projects sponsored by Docker.
Quay.io	100	Most used container images in Quay.io.
ECR	100	Images published and maintained by Amazon.

Obfuscation in the wild

Number of containers with missing, modified, or deleted information (out of 600 containers)

Observations:

- More than 10% present OS obfuscation
- Almost 20% download software directly from the Internet without using package managers
- OSPKG in more than 50% of tested containers
- Software dependency on 3% of containers and package obfuscation on 15%

Obfuscation type	*Missing*	*Modified*	*Deleted*
OS	49	23	3
OSPKG	55	364	58
URL	187	N/A	N/A
DEP (total)	N/A	15	3
Python	N/A	1	2
Ruby	N/A	2	1
Node.js	N/A	9	0
PHP	N/A	2	0
Go	N/A	1	0
PKG (total)	N/A	69	23
Python	N/A	48	19
Ruby	N/A	5	1
Node.js	N/A	12	2
PHP	N/A	1	0
Go	N/A	3	1

Obfuscation in the wild

Number of containers with missing/modified/deleted information for each registry (out of 100 containers per registry)

Observations:

- Obfuscation present in all container registries
- More present in DockerHub OSS and third-party registries.
- 90% of Bitnami containers install code from the Internet, but keep metadata

DO: Docker Official / DB: Docker Bitnami / DV: Docker Verified / DOSS: Docker OSS / Q: Quay.io / E: ECR.

Obfuscation	*DO*	*DB*	*DV*	*DOSS*	*Q*	*E*
OS	4	3	20	19	20	9
OSPKG	100	3	99	117	93	21
URL	27	94	13	39	9	5
DEP (total)	0	0	0	6	2	10
Python	0	0	0	2	0	1
Ruby	0	0	0	1	0	2
Node.js	0	0	0	2	2	5
PHP	0	0	0	1	0	1
Go	0	0	0	0	0	1
PKG (total)	6	0	4	18	22	38
Python	3	0	2	13	16	33
Ruby	1	0	0	2	1	2
Node.js	2	0	1	3	4	2
PHP	0	0	0	0	0	1
Go	0	0	2	0	1	0

Real-world example (1/2)

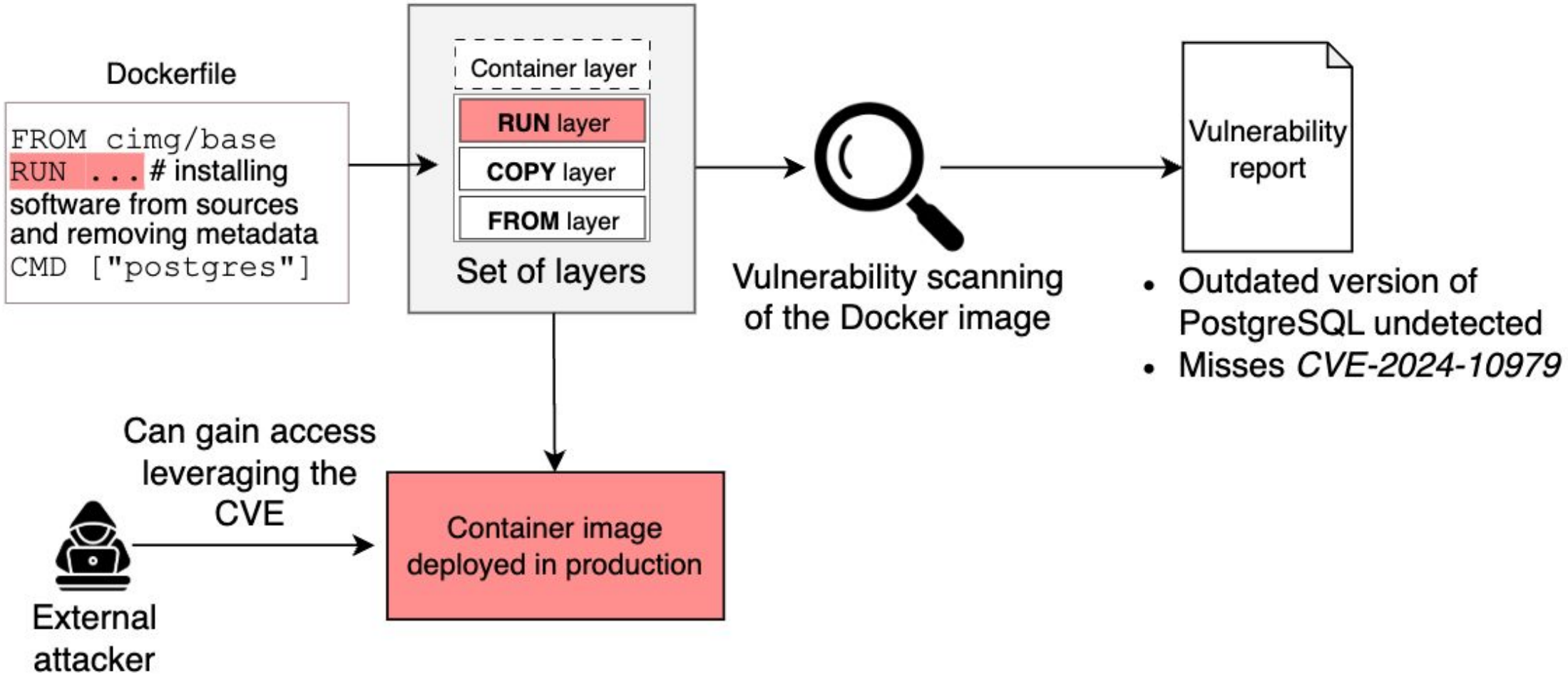
Extract from the CircleCI Dockerfile for PostgreSQL container image

Single `RUN` command: install build dependencies, compile database + plugins from source, and remove installed dependencies.
→ Corresponding layer only stores the executables, with **no trace of installed package manager files to identify dependencies**.

```
RUN BUILD_DEPS="clang \
        dirmngr \
        gnupg \
        libclang-dev \
        libicu-dev \
        libipc-run-perl \
        libkrb5-dev \
        libldap2-dev \
        liblz4-dev \
        libpam-dev \
        libperl-dev \
        libpython3-dev \
        libreadline-dev \
        libssl-dev \
        libxml2-dev \
        libxslt1-dev \
        llvm \
        llvm-dev \
        postgresql-server-dev-all \
        python3-dev \
        tcl-dev \
        uuid-dev" && \
    apt-get update && apt-get install -y --no-install-recommends \
    $BUILD_DEPS \
    && rm -rf /var/lib/apt/lists/* && \
    localedef -i en_US -c -f UTF-8 -A /usr/share/locale/locale.alias en_US.UTF-8 && \
    curl -sSL "https://ftp.postgresql.org/pub/source/v${PG_VER}/postgresql-${PG_VER}.tar.gz" | tar -xz && \
    cd postgresql-${PG_VER} && \
    ./configure
    && make -j $(nproc) world && \
    make install-world && \
    cd .. && rm -rf postgresql-${PG_VER} \
    && git clone --depth 1 https://github.com/citusdata/pg_cron.git /pg_cron && \
    cd /pg_cron && make && PATH=$PATH make install && \
    rm -rf /pg_cron && \
    apt-get purge -y --auto-remove $BUILD_DEPS
```

Real-world example (2/2)

Consequences: Outdated package & associated vulnerabilities remain undetected and attacker can potentially gain access

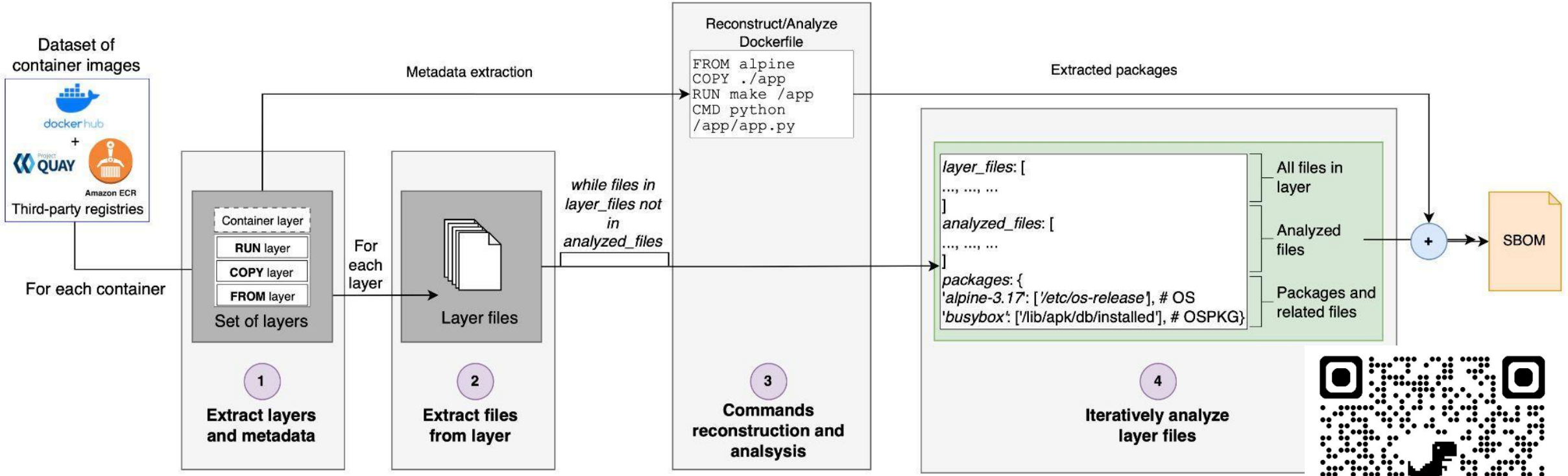


KubeCon | CloudNativeCon
Europe 2025

Objective #4
Mitigate obfuscation

Mitigate obfuscation

We developed an **original methodology** and **open-source tool** to improve package detection.



Available here →

Main features of our tool

1. **Reconstruct** the **Dockerfile with reverse-engineering**

→ Enable to retrieve external downloads and URLs.

1. Examine each container **layer individually**

→ Detect and analyze deleted and modified files from previous layers.

1. Analyze both package metadata and **actual package-related content** files

→ Analyze also binaries, config. files, libraries in case package metadata has been tampered.

1. Analyze both index files and file paths

→ Cross-check package manager indexes, file system structures, and naming conventions.

Insights

/!\ Not all obfuscation techniques can be mitigated:

- Multi-stage builds
- Compressed images

→ **Need for new best-practices**

Current best practices for writing optimized Dockerfiles encourage **multi-stage builds** and the **external download** of binaries and source code.

→ These guidelines should be revised to improve **transparency** and **resilience against obfuscation** of containers.

Thank you!

KubeCon | CloudNativeCon Europe 2025

Agathe Blaise
agathe.blaise@thalesgroup.com

Jacopo Bufalino
jacopo.bufalino@lecnam.net

THALES

Aalto University
School of Science

le cnam