Static composition analysis of containers, virtual machines and other root filesystems

For provenance, license and vulnerabilities

Introduction: Philippe Ombredanne

- Weird facts and claims to fame
  - Signed off the **largest deletion of source lines in the linux kernel** (but these were only comments)
  - Repenting code hoarder (only 20K forks)
- Maintainer of FOSS tools for FOSS code analysis
  - ScanCode and AboutCode
- Co-founder of SPDX, ClearlyDefined
- long time GSoC mentor
- Co-founder and CTO of nexB Inc.
- pom@nexb.com or pombredanne@gmail.com

Agenda

- The problem with containers
- How to solve the issue
- Ideal solution
- Composition analysis pipeline
- What's your linux distro?
- Scan installed system packages
- Scan application packages
- Scan for remaining files
- What about license and vulnerabilities?
- Alternative tools
- Architecture
- Status

The problem with containers (1)

- A container is essentially a kernel-less root filesystem
  - But more than a single rootfs, this is actually many rootfs
  - One for each "layer" in a union filesystem
- Each layer
  - can have similar duplicated or updated packages and files
  - may contain a whole userland
    - with system packages (multiple versions)
    - with application packages (multiple versions)
    - with extra files added and copied from undetermined origins
  - 1000's of these

The problem with containers (2)

- **Many (many) packages and then some more**
  - mostly pre-built binaries
  - base image builders may bypass signature checks for distro packages
  - images binaries are built on top of image binaries built on top of binaries packages
- Not always a clear provenance and license
  - Package metadata are not enough or not present
  - Sometimes doc or metadata are removed to keep things smaller
- Lack of traceability

The problem with containers (3)

- **Using piles of unknown binaries is not ideal**
  - If his is open source code, where is the source?
  - What's the license?
  - What are the known vulnerabilities or bugs?
- With so many pre-built binaries of unknown provenance, then what's to love in that ??
- **Unknown, weirdly-licensed, buggy or vulnerable code will sneak in easily**
- So why do we use containers in the first place then?
  - We, developers, are lazy!
  - Convenience beats everything and this is very convenient

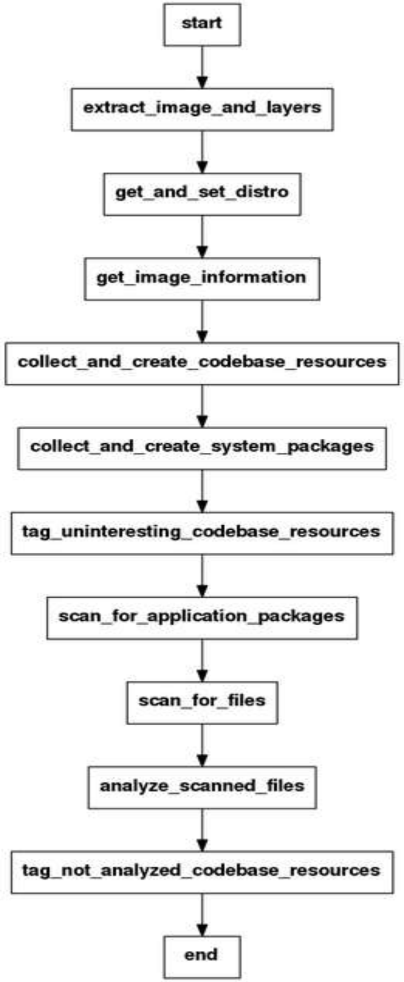
How to solve the issue

- In the future, we will have fully vetted, traceable containers with **reproducible builds**
  - One day, hopefully
- For now, "software composition analysis" is the way
  - Find ALL the **packages**
  - Then, trace back ALL the files to **determine provenance**
  - Then, find the **licenses**.
  - Then, find the **vulnerabilities**.
- Done.

Ideal solution

- **Free, open source and open data of course**
- Guarantee that ALL files in an image are vetted
  - Not a mere inventory of packages and documented licenses
- Scriptable tool that is easy to customize
  - There is no one tool to rule them all so you need to easily include and plugin new tools and scripts
- Bonus: do it **without running containers** with a pure static analysis
  - simpler installation and runtime
  - and avoid the "**observer effect**" by NOT running inside the container you analyze

Composition analysis pipeline



- Prepare image, determine distro
- For each image layer: scan **system packages**
  - Find their file and check if modified
- For remaining files: scan **application packages**
  - All ScanCode-supported package types (ruby, go, npm, maven, composer, etc.)
  - Find their file and check if modified
- For remaining files: scan files
  - All files, including binaries
- Finally, analyze remainder
  - Dispose of temp and transient or log files and more

Layers and Union filesystem

- The layers are slices of rootfs "layered" on top of each other using a union filesystem (AUFS, overlayFS)
- Rather than requiring the availability of the FS drivers for these the approach is to either:
  - Analyze a **squashed image** where the layers are overlayed reproducing the procedure using the union FS, but without the need for a driver
  - Analyze **layer by layer**, and check what was analyzed in the previous layers to avoid duplicate
- Both implemented in the **container-inspector** library

What's your linux distro?

- `/etc/os-release` is the best way
  - Older distro-specific ways are not worth it
- But some containers have no "distro"
  - e.g. minimal busybox-based userland base images and nothing else
  - "distroless" images are more or less based on Debian but are not exactly Debian.
- The discovered distro drives what installed system packages DB are checked for

Scan installed system packages

Read directly installed package databases

- On Debian distros `/var/lib/dpkg/status` and `info/`
  - RFC-822 Email-like format with .md5sums and .list file lists
  - distroless use a partial Debian-like db
- On RPM distros `/var/lib/rpm/Packages`
  - A binary blob in either BDB hash, sqlite DB or own **ndb** dbm-like (SUSE anyone?)
  - Older or new Fedora and derivative and openSUSE each use a different database format
- On Alpine `/lib/apk/db/installed`
  - RFC-822 Email-like, close to but not Debian

System packages of other distro

- **Scan installed system packages for other distro can be derived** easily from existing distro handlers
- For instance, close to home with openSUSE **RPMs**
  - the installed database is using BDB in the past and NDB going forward
  - This will come with the upcoming RPM support using a special librpm build
- For instance, with **archlinux**
  - with pacman, each installed package has a `dest` file with metadata and `mtree+files` listings
  - There are existing parsers

Scan application packages

- Only on the subset of files that are NOT part of system packages
- Use package **manifests,** lock files and package installation conventions to detect installed packages. For instance:
  - **python** `site-packages`
  - **npm** nested node_modules tree
  - **Maven** Jars
  - installed **Rubygems**
  - **etc....**
- Use scancode-toolkit scanners with many parsers
- For each, collect the set of installed files

What if a package lies about its files?

- We should trust but verify
- Verify either with:
  - "built-in" crypto and signatures
  - lookup in a database of known packages and files
- A lookup is easier
  - The open database of all the package files is in the works (a subset focused on licensing is already available through ClearCode project)
  - Lookup by checksum

Scan for remaining files

- Only on the subset of files that are NOT part of system packages or application packages
- Use ScanCode-toolkit scanners for license and origin clues
- For files with no explicit origin and licenses, lookup in a database
- As noted before, an open database of all the package files is in the works (a subset focused on licensing is already available through ClearCode project)
- Lookup by checksum

Finally…

- The leftover subset of files that are neither from system nor application packages and cannot be traced to some known provenance are …**suspicious files!**
- Some are transient database, temp or log files with well known locations, filetype and content
- **The rest need to be subject to extra analysis**
- Introspect binaries for origin clues
  - DWARF symbols, ELF symbols, C++ demangling, Strings or reversing
  - In the future, lookup in a database of symbols, signatures and strings TBD
  - Or YARA rules?

What about license and vulnerabilities?

- License is derived from package metadata and scans of the source code (using best in class **ScanCode-toolkit** scanner)
- Vulnerabilities are found thanks to the new **VulnerableCode** aggregated and open source database of known vulnerabilities
  - lookup is done using **PackageURL**s (a project derived from Scancode and VulnerableCode and adopted by OWASP and many more)
  - for system and application packages (and more than just the NVD)
  - possibly YARA rules too in the future

Architecture

- Server to host pipelines execution and data storage:
  - Python, Django, PostgreSQL
- Each composition analysis is a pipeline
  - Scripting customizable with resume/restart
- Minimal API-only JSON, almost no UI beyond basic CRUD
  - **ScanCode.io + ScanPipe** for end-to-end pipeline scripting and execution
  - **ScanCode** toolkit for license and application package detection
  - **NetFlix's Metaflow** ml/data science workflow engine
  - **container-inspector** library for container image processing
  - **Debut** for Debian, Alpine (and soon **RPM** and distroless) for system package
  - **VulnerableCode** for vulnerabilities lookup
  - **PackageURL** to identify packages

Alternative tools

- Open source with Tern, Trivy, Clair, Anchore
- Several commercial but none with similar feature sets
- Except for Tern (that also uses Scancode and debut) they typically focus only on security and have little or no support for file origin, license and other metadata tracing
- Typically less coverage of application packages and little or limited support to trace which file belong to a package
- Typically require to mount the image as a union filesystem and/or to run the original package managers in a container. Most of them require Docker to be installed and run themselves inside a Docker image too. This requires a more involved setup and runtime.

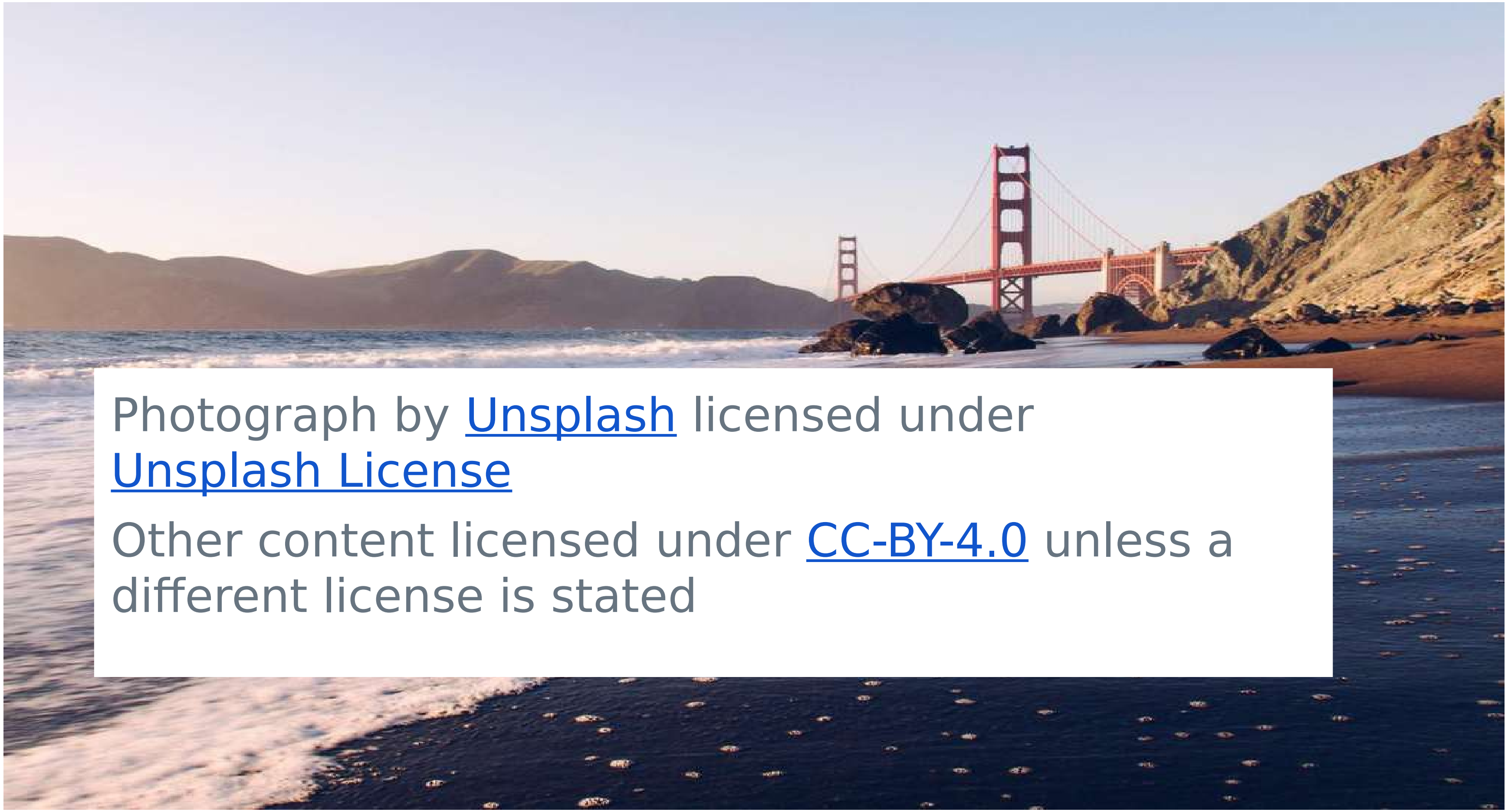
Status

- Base architecture is in place ~ 70% complete
  - For Debian, Ubuntu and Alpine done, RPM-based, distroless distros are next
- **container-inspector** library for images complete
- **debut** library for Debian parsing complete
- **rpm-inspector** library for RPM under development
- **scancode-toolkit** support for installed Debian & alpine WIP
- **scancode-toolkit** parsers for application packages complete
- **vulnerablecode** DB is WIP, about 70% complete
- **PackageUrl** library complete

About nexB

- Focused on FOSS compliance since 2007
- Hybrid solution for FOSS governance
  - Business applications for Legal/Business
  - Open source tools for Developers
  - APIs in-between
- Overview of our FOSS projects at www.aboutcode.org
- Our FOSS tools are at https://github.com/nexB
- Co-founders of SPDX - http://spdx.org/
- Co-founders of ClearlyDefined - https://clearlydefined.io/

Credits



Finish

Thank You