Integrating OSSelot curation data into OpenEmbedded

Presenting meta-osselot

Jasper.Orschulko@iris-sensing.com

Licensing and Links

The content of this presentation is licensed under Creative Commons BY 4.0 and available at https://github.com/Jasper-Ben/presentation-meta-osselot-cool.

Here, you will also be able to access all links mentioned within the presentation.

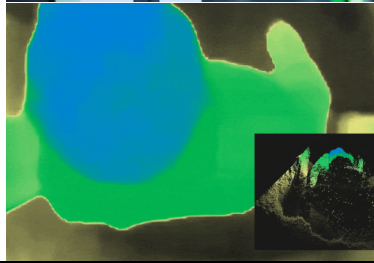
Feel free to reuse and share in accordance to the license terms. See the Repository README for more details.

$ whoami

- Jasper Orschulko, anno 1994
- Bachelor of Computer Science
- At Iris since December 2019 as DevOps Engineer
- My main topics are:
  - Infrastructure Management and Container Orchestration
  - Product Maintenance, Build & Release Management
  - Unifying dev environment
- I would describe myself as ”the tooling guy”
- Automating processes make me go 😍
- Manual labour makes me go 😰

And a bit about my employer: iris-GmbH

- Berlin (Germany) based with >150 employees
- We do passenger counting in public transport utilizing TOF sensors with >99% accuracy
- Data used for optimizing schedules, train sizes and ticket sales distribution, ...
- Since 1991, approximately 250,000 of our counting systems have been sold worldwide and installed in more than 80,000 vehicles
- Heavily relying on (and contributing to) open source software
- We are hiring!



Obligatory Disclaimer ⚖

- IANAL (I am not a lawyer), take any of my “legal advice” with a grain of salt!
- When in doubt, consult with a lawyer of your confidence before releasing software into the wild. We have a legal department for this
- Note section 5 of CC BY SA 4.0.

What are OpenEmbedded & Yocto?

Since you are here, you probably already know what these are. If not, here the brief summary:

- Community projects that allow building custom-tailored Linux-based Operating Systems
- Funded by the non-profit Linux Foundation
- Useful for purpose-built, embedded devices with limited resources, think POS-system, Printer, Router, `re.compile('smart(?!phone).*')`
- A lot of popular software already packaged) for OE, well tested and continuously improved by the community
- Allows developers to focus more on the core functionality of a product and contributing to existing projects rather than reinventing the wheel

The “Hidden” Costs of (F)OSS Software

(F)OSS software usually reduces costs, as shown in the results of recent survey commissioned by the LF) but it is not free of costs, e.g.:

- (Usually) no warranties, guarantees or support, thus:
  - Might require more “hands-on” than commercial offerings, thus requiring skilled software engineers or partner companies at hand
  - Might not fulfil certification requirements or certification might need to be paid out of own pocket

  → These are topics OSADL might be able to help you with 🙂
- **Fulfilling license obligations, aka License compliance can be a lengthy and expensive process**

  ⇒ This is what we will be talking about today with a focus on OpenEmbedded Linux

(OE) License Compliance obligations (over)simplified

For everything that is shipped to customers:

1. Provide original Source Code for copyleft (e.g. GPL*) licensed Software
2. Provide modifications to Source Code for copyleft licensed Software
3. Provide Copyright notices from Licenses & Source Code
4. Provide License notices from Licenses & Source Code
5. Provide build tooling and scripts ("[...] complete source code means all the source code [...] **plus the scripts used to control compilation and installation of the executable**." - GPL* License)

Using OpenEmbedded's Built-In Compliance Tooling

The "old" workflow: Using the `archiver.bbclass` for creating open-source (/copyleft) source-code archives

→ Downsides:

- Uses only the licenses declared in OE recipes which usually only contain the "top-level" package license, not licenses defined throughout the project's source code
- Copyright statements for non-copyleft software is missing (unless providing source code for all open-source software)
- Static linked build-time dependencies not included
- Sheer size of these archives
- Does not provide a workable SBOM

Using OpenEmbedded's Built-In Compliance Tooling

The "new" workflow: Using `create-spdx.bbclass` for creating a SPDX SBOM and optionally source-code archives

- Fixes a lot of issues with the "old" workflow
- Too complex to discuss in detail here, see Joshua Watt's in-depth video for the nitty-gritty

→ However, some issues remain:

- Most notably, "bad" license data due to oversimplified license detection (only checking for SPDX headers)
- Not easy to fix "in code" due to variety in which licenses and copyrights are declared throughout source code

Taking License Compliance a Step Further

For many use-cases (e.g. consumer-grade products) the OE approach is a good balance between effort and results. Providing the SPDX, together with (F)OSS source code will probably™ be enough to not get sued.

However, sometimes the results from OE's built-in mechanisms are not enough:

- Increased demand for exhaustive SBOMs and License Compliance Clearing Documents throughout the industry (especially towards component suppliers)
- You want to go the extra mile to improve confidence in your compliance status

→ In these cases there is no way around inspecting the actual source code files for copyright and license statements

However, manually and repeatedly inspecting source code is tedious work and becomes unmanageable for small and medium sized companies when distributing large software packages, such as an embedded Linux system.

Cue to the Osselot Project

- Relatively new project (started roughly two years ago)
- Re-uses existing FOSS software (e.g. FOSSology, git)
- Trained curators create and validate high quality SPDX SBOMs for popular open-source software
- Curation data licensed under the CC0 1.0 Universal
- SBOMs made available in GitHub and via a REST API

The basic idea: Crowdsource the necessary curation data in a shared effort (true to the open-source spirit)

⇒ The challenge: How to make the data useful in your OE environment?

Why integrate Osselot into the OE build system?

- Tackle the problem at the root
- Build system includes a lot of useful meta-information
- Reuse existing OpenEmbedded mechanisms (sstate cache, task dependencies, etc.)
- Reuse processes familiar to the OE community

The meta-osselot OE layer

- Open-source (MIT licensed) OpenEmbedded layer
- Available at https://github.com/iris-GmbH/meta-osselot
- Mainly consists of three components:
  - `osselot-package-analysis-native.bb`: recipe instructing OE from where to fetch OSSelot curation data
  - `osselot.bbclass`: The logic for matching OpenEmbedded packages against their OSSelot counterparts
  - `*.bbappends`: Various bbappends for amending recipes from upstream layers for Osselot usage

Closer look at osselot-package-analysis-native.bb

- Takes a git source repository at a certain refspec (defaults to OSSelot's repository with latest available data on main branch)
- Clones the repository into a work directory
- Searches for available package/version combinations:
  1. Find versions by searching for folders with a "version-" prefix

  → package version == version folder stripped of "version-" prefix

  2. Identify the package name via the parent folder name

  → package name == package folder name
- Write available packages/versions into JSON file for later consumption

Design choice: Git repository vs. REST API

Why don't we use the REST API provided by OSSelot for data collection?

- Easier to integrate package curation data that is not upstreamed to OSSelot (yet) using a repository fork
- Follow OpenEmbedded best-practices: Separate data fetching steps from build steps. We can still reproduce the build offline!
- Reduce bandwidth usage and runtime: OpenEmbedded git fetcher will only rerun on changes to the git repository
- More flexibility when using raw data: Package name suggestions on similar matches and reuse during version mismatch (see following slides)

Closer look at osselot.bbclass

- Decides whether a package should be validated against the OSSelot database (based on configurable parameters)
- Reads JSON file containing available OSSelot packages/versions from osselot-package-analysis-native
- Attempts to find the OpenEmbedded package in list of OSSelot packages
  - On failure: Suggests close package matches and exits
- Attempts to find the given package version within OSSelot
  - On failure: Will reuse closest available version match
- Calculates checksums of source code files within OE package and compares them to corresponding data available in OSSelot SPDX JSON file
- Writes available OSSelot data and corresponding meta file to the OpenEmbedded deploy directory

Issues matching OE packages to OSSelot data

- Same package might be available under diverging package names / version strings
  e.g., expat 2.5.0 vs. libexpat R_2_5_0
- OE occasionally uses package release tarballs rather than the “original” git code OSSelot uses or might add patches

  → Slight differences in source code possible
- OE ships some packages without point releases, e.g., config files where “source code” is stored directly within the OE layer repository

⇒ How can we address these issues?

Closer look at bbappend files

Extends existing recipes from community layers to work better with OSSelot:

- Automatically applied if relevant layer is used during build (using BBFILES_DYNAMIC)
- Adjust `OSSELOT_NAME` or `OSSELOT_VERSION` variable on naming scheme mismatch between OpenEmbedded package and OSSelot data
- Ignore packages not relevant for license compliance
- Ignore files/globs within packages not relevant for license compliance
- Define checksum equivalence between file versions, e.g., patched files with no changes to license information

⇒ Fixes many (but not all) issues when matching OE packages to OSSelot data

Example bbappend file (dbus_%.bbappend)

```
# SPDX-License-Identifier: MIT
# Copyright 2024 iris-GmbH infrared & intelligent sensors
# The release package for dbus which is used by openembedded already
# contains artifacts from a pre-run configure step.
# These are not relevant for license compliance.
OSSELOT_IGNORE_SOURCE_GLOBS += " \
    configure \
    aminclude_static.am \
    Makefile.in \
    config.h.in \
    aclocal.m4 \
    m4/ltversion.m4 \
    m4/ltoptions.m4 \
    m4/libtool.m4 \
    m4/lt-obsolete.m4 \
    m4/ltsugar.m4 \
    doc/Makefile.in \
    test/Makefile.in \
    test/name-test/Makefile.in \
    bus/Makefile.in \
    dbus/Makefile.in \
    tools/Makefile.in \
    build-aux/config.guess \
    build-aux/config.sub \
    build-aux/missing \
    build-aux/install-sh \
    build-aux/tap-driver.sh \
    build-aux/ltmain.sh \
    build-aux/depcomp \
    build-aux/compile \
    cmake/DBus1ConfigVersion.cmake \
    cmake/DBus1Config.cmake \
"
## Define equivalence between unpatched and patched source code files
## Unpatched hashes taken from dbus/1.14.8
# configure.ac
OSSELOT_HASH_EQUIVALENCE += "c84fb14d03d4542d04a34725a41ad28e:7e480ba3a0d09e77c20758a538ddae4d"
```

Integrating meta-osselot in your build

Enabling meta-osselot is as simple as:

1. Adding the meta-osselot layer to your build configuration (use appropriate release branch)
2. Adding `INHERIT += "osselot"` to your local.conf file

→ Every bitbake build command will now automatically include all relevant osselot tasks, e.g.: `bitbake core-image-minimal`

→ Alternatively, only running osselot-relevant tasks:
`bitbake core-image-minimal --runonly=populate_osselot`

For more advanced use-cases see the projects README.

Taking a look at meta-osselots output

Meta-osselot will create folders for all build relevant packages in OEs deploy directory, containing:

1. A package specific meta-file containing information e.g., on:
   a. The OSSelot status on this package (currently: `"ignored"`, `"version_mismatch"`, `"not_found"`, `"found"`)
   b. Ignored source files
   c. Source files checksum mismatches
   d. Applied checksum-equivalence statements
   e. Source files checksum matches
2. If package available: All OSSelot curation files for this particular package

What is next for (meta-)osselot?

- Currently still many OE relevant packages missing from OSSelot database

  $\rightarrow$ We hope to achieve 100% OSSelot coverage for the poky core-image-minimal reference image for LTS releases $\geq$ kirkstone
- Deal with "OE specific" packages
- Evaluate reusing data from `create-spdx.bbclass` or compiler data for further narrowing down of packages and source code files that are included in target image
- Work closer with OE maintainers, ideally get OSSelot integration upstreamed
- Anything in the issue tracker
- As usual: Time constrains is the biggest issue 🥲

How can you help?

- Get involved
  - Contribute curation data to OSSelot
  - Improve meta-osselot
- Spread the word
- Talk to us
  - Feedback is always appreciated!

Demo

Demo time!