APERTIS

Automated License Compliance

Contents

A Linux system such as those assembled by Apertis contain components licensed under many different licenses. These various licenses impose different conditions and it is important to understand to a good degree of fidelity the terms under which each component is provided. We are proposing to implement an automated process to generate software Bills Of Materials (BOMs) which detail both the components used in Apertis and the licensing that applies to them. Licensing isn't static, nor is it always as simple as all the components from a given source package deriving the same license. Packages have been known to change licenses and/or provide various existing or new components under different terms. Either now or at some point in the future, the licenses of some of the components in Apertis may start to be provided under terms that Apertis may wish to avoid[1]. For example, by default Apertis is careful not to include components to be used in the target system that are licensed under the GPL version 3, the licensing terms wouldn't be acceptable in Apertis' target markets.

In order to take advantage of new functionality and support being developed in the software community, Apertis needs to incorporate newer versions of existing software packages and replace some with alternatives when better or more suitable components are created. To ensure that the licensing conditions remain favorable for the use cases targeted by Apertis, it is important to continually validate that the licensing terms under which these components are provided. These licensing terms should be documented in a way that is accessible to Apertis' users.

Debian packages by default track licensing on a per source package level. The suitability of a package is decided at that level before it is included in Debian, which meets the projects licensing goals[2]. Apertis will continue to evaluate licensing before the inclusion of source packages in the distribution, but also wishes to take a more nuanced approach, tracking licensing for each file in each of it's binary packages. By tracking licensing to this degree we can look to exclude components with unsatisfactory licensing from the packages intended for distributed target systems, whilst still packaging them separately so they may be utilized during development. A good example of this situation is the

[1] https://em.pages.apertis.org/apertis-website/policies/license-expectations/
[2] https://www.debian.org/social_contract.html#guidelines

gcc source package and the libgcc1 binary package produced by it. Unlike the other artifacts produced by the GCC source package, the libgcc1 binary package is not licensed under the stock GPLv3 license, a run time exception[3] is provided and it is thus fine to ship it on target devices. The level of tracking we are providing will detect such situations and will offer a straight forward way to resolve them, maintaining compliance with the licensing requirements.

To achieve this 2 main steps need to be taken:

- Record the licensing of the project source code, per file
- Determine the mapping between source code files and the binary/data files in each binary package

These steps have been integrated into our CI pipelines to provide early detection of any change to the licensing status of each package. Extending our CI pipelines also enables developers to learn about new issues and to solve them during the merge request development flow.

FOSSology

FOSSology is an Open Source server based tool which provides a web front-end that is able to scan through source code (and to a degree binaries) provided to it, finding license statements and texts. To achieve this FOSSology employs a number of different scanning techniques to identify potential licenses, including using matching to known license texts and keywords. The scanning process errs on the side of caution, generating false positives over missing potential licensing information, as a result it will be necessary to “clear” the licenses that are found, deciding whether the matches are valid or not. The scanning and clear process during the first time is more time consuming and requires special attention, however, subsequent runs should be much faster as FOSSology is able to use previous decisions to find the license information. Once completed, FOSSology records the licensing decisions and can apply this information to updated scans of the source. It is anticipated that, after an initial round of verification, FOSSology will only require additional clearing of license information should the scan detect new sources of potential licensing information in an updated projects source or when new packages are added to Apertis. It is possible to export and import reports which contain the licensing decisions that have previously been made, if a trusted source of reports can be found then these could also be imported, potentially reducing the work required.

FOSSology is backed by the Linux Foundation, it appears to have an active user and developer base and a significant history and it is the de-facto Open Source Software solution for license compliance. As such, it is felt that this tool is likely to be maintained for the foreseeable future.

As this tool provides a web bases UI, it presents an additional advantage, as it makes it easier for non-technical users, such as auditors or lawyers, to access

[3] https://em.pages.apertis.org/apertis-website/policies/license-exceptions/#gcc8

and manage the reports, allowing a smooth integration in an audit process.

For all the reasons mentioned above we understand this is the best choice for integration into the Apertis workflow.

CI Pipeline integration

In order to avoid manual tasks the license detection should be integrated into the CI process. FOSSology provides a REST API[4] to enable such integration.

FOSSology is able to consume branches of git repositories, thus allowing scanning of the given source code straight from GitLab. This process should be triggered after updating a package from external sources, as in this cases a license change can be introduced. A report will be generated and retrieved, using the REST API, which describes (among other things) the licensing status of each file. The report can be generated in a number of formats, including various SPDX flavors that are easily machine parsable, using DEP5[5] as the preferred option. It is suggested that each component should require a determination of the licensing to have been made for every file in the project. Due to the large volume of licensing matches that will result from the initial licensing scan, we recommend that the absence of license information initially generates a warning. In some cases, to achieve the fine grained licensing information desired, the licensing of some files may need to be clarified with the components author(s). Once an initial pass of all Apertis components had been made we would expect missing license information to result in an error, as such errors would be as a result of new matches being found, which would need to be resolved in FOSSology before CI would complete without an error. The generated report should be saved in the Debian metadata archive so that it is available for the following processing.

The adoption of FOSSology will be gradual and in parallel with the current license scanning process in order to compare the results and improve the workflow. Once the process is fully reviewed and tested with all the packages in the target repository FOSSology will be the default scanner.

Binary to source file mapping

Binaries are built from many different source files, but the exact list of them depends on build options. For this reason a reliable mechanism needs to be put in place to extract this list after the build process in order to determine the license information.

Compilers store information in the binaries it outputs, that can be used by a debugger to pause execution of a process at a point corresponding to a selected line of source code. This information provides a mapping between the lines of

[4] https://www.fossology.org/get-started/basic-rest-api-calls/

[5] https://dep-team.pages.debian.net/deps/dep5/

source code and the compiled machine code operations. Executable binaries in Linux are generally stored in the Executable and Linkable Format[6] (ELF), the associated DWARF[7] debugging data format is generally used to store this debugging information inside the ELF in specific "debug" sections.

The tool `dwarf2sources` parses this information and extracts the name of the source files that were used to generate each binary, generating a `json` file that can easily be parsed later. Combining this with the licensing information provided in the licensing report, a mapping can be made between each binary and it's associated licenses.

CI Pipeline integration

Apertis uses the Open Build Service (OBS) platform to build the binary packages in a controlled manner across several architectures and releases. OBS utilizes `dpkg-buildpackage` behind the scenes to build each package. This utility has access to the source licensing report as it is contained in the Debian metadata archive. As well as the source licensing, the Debian metadata archive contains configuration to help `dpkg-buildpackage` determine how to build the source. This is typically done with the help of `debhelper`[8], which provides helpers that simplify this process.

Apertis extended `debhelper` by including a new command `dh_dwarf2sources` to perform the source file name extraction using `dwarf2sources` as described above. Typically the binaries are striped (using a debhelper command called `dh_strip`) prior to packaging, removing the debug symbols from the binary and reducing its size. For this reason `dh_dwarf2sources` is placed before this step in the `dh` sequence. Whilst the debug symbols are kept, packaged separately in the `dbgsym` package, it's easier to perform the mapping before this is done. The result is stored in the binary package under `/usr/share/doc/<package>/`.

Following this same idea, Apertis also extends `debhelper` command `dh_installdocs` to install the license report generated by FOSSolgy in the binary under `/usr/share/doc/<package>/copyright_report`.

Despite that this solution should work for most packages, some of them might need special handing as may override default rules. These special cases will be covered with further improvements.

There may be packages in Apertis that do not make use of debhelper, these packages need special handling to ensure that the required steps are completed.

As these reports are provided by each binary package, the reports from installed packages can be accessed at image build time and amalgamated into an image wide report at that point should it be required. As a binary can be built from multiple sources, each with differing licenses, it is necessary for the report to

[6] https://en.wikipedia.org/wiki/Executable_and_Linkable_Format

[7] https://en.wikipedia.org/wiki/DWARF

[8] https://manpages.debian.org/jessie/debhelper/debhelper.7.en.html

detail each file that is used to create each binary and the licensing under which it is provided. In some circumstances dual licensed source code may allow for a binary to be effectively licensed under the terms of a single license, that is the user has the option to pick a license that results in the whole binary being able to be provided under the terms of a single license. Where dual licensed source code isn't used, the terms of all applicable licenses should be declared. The terms of the various licenses may be considered compatible[9], allowing the binary to effectively be managed under the terms of the more restrictive license. For example, a binary derived from source code licensed with the GPLv2 license and other source code licensed with the MIT license, the terms of both apply to the binary, though as the terms of the MIT license will be met if the binary is used in accordance with the terms of the GPLv2, then handling the binary as though it was licensed under the GPLv2 will ensure the terms of both are met. Not all possible combinations of licenses work out this way and thus why it is important to ensure that licensing is properly tracked.

Binary Licensing Reporting

The approach each project using Apertis takes with regards to the reporting of licensing information should be driven by how this information is to be utilized, i.e. some projects may wish to parse the license information and present it in a single BOM file in HTML, XML or human readable text.

For the images provided by the Apertis project, the script `generate_bom.py` combines the reports saved in `/usr/share/doc/<package>/`, which consists in a `json` per package and a `DEP5` file per source package into a single `json` file which is provided with the image. This file can be generated with different levels of verbosity allowing to list licenses per image, package, binary or source file.

This same scripts also issues a warning in case a problematic license is found.

CI Pipeline integration

Apertis utilizes Debos[10] in its image generation pipeline, which provides a very versatile way of customizing them. During the final stage of the image creation, the script `generate_bom.py` is used to build the BOM file with the license information of the image and export it as an additional artifact. Finally as both `minimal` and `target`images should not shipped extra data, the contents of `/usr/share/doc/` are dropped from the image.

Step-by-step process

This is a description of the steps in the process as currently implemented:

[9] https://en.wikipedia.org/wiki/License_compatibility

[10] https://github.com/go-debos/debos

1. when a package is imported from Debian to Apertis the `scan-license` job in the packaging pipeline[11] will call `ci-license-scan`[12] to submit the sources to the scanner, be it `scan-copyright`, FOSSology or any other tool
2. metadata in `debian/apertis/copyright.yml`[13] can be used to override things where the scanner gives the wrong results, which should no longer be needed once the switch to FOSSology is completed and the correct licensing information is stored in its database
3. the output is committed in the `debian/apertis/copyright` YAML files in the sources[14]
4. if some files have problematic licenses but they do not really affect us for any reason, the reason is documented in `debian/apertis/copyright.whitelist`[15]
5. for packages meant to be installed on production devices, the packaging pipeline will fail if problematic licenses are detected and the affected files are not whitelisted
6. when the sources are submitted to OBS, the `dh_dwarf2sources` subcommand for Debhelper[16] calls the `dwarf2sources` tool[17] to generate a mapping from binaries to the source files used to build them
7. the output is included in the same `.deb` file as the processed library/executable, under `/usr/share/doc/$packagename/copyright_report.gz`
8. for each installed `.deb` package the `/usr/share/doc/$packagename/copyright_report.gz` files get unpacked during image generation
9. the `generate_bom.py` script[18] is invoked at the end of each image recipe[19], loading all the `/usr/share/doc/*/copyright_report.gz` files producing a JSON report[20] alongside each produced image, using the source→license and binary→source mappings above to match each installed library and executable to the licenses of the sources used to build them
10. human-readable reports in any format can be generated by the JSON data describing the licenses that apply to the libraries and executables shipped in the image itself

[11] https://gitlab.apertis.org/infrastructure/ci-package-builder/-/blob/c2c59e28/ci-package-builder.yml#L313

[12] https://gitlab.apertis.org/infrastructure/apertis-docker-images/-/blob/6bc2a375/package-source-builder/overlay/usr/bin/ci-license-scan

[13] https://gitlab.apertis.org/pkg/gnutls28/-/blob/dae6f34d/debian/apertis/copyright.yml

[14] https://gitlab.apertis.org/pkg/gnutls28/-/blob/dae6f34d/debian/apertis/copyright

[15] https://gitlab.apertis.org/pkg/gnutls28/-/blob/dae6f34d/debian/apertis/copyright.whitelist

[16] https://gitlab.apertis.org/pkg/debhelper/-/blob/8abfd8a5/dh_dwarf2sources

[17] https://gitlab.apertis.org/pkg/dwarf2sources/

[18] https://gitlab.apertis.org/infrastructure/apertis-image-recipes/-/blob/283bcd3f/scripts/generate_bom.py

[19] https://gitlab.apertis.org/infrastructure/apertis-image-recipes/-/blob/283bcd3f/image-uboot.yaml#L150

[20] https://images.apertis.org/release/v2022dev2/v2022dev2.0/arm64/minimal/apertis_v2022dev2-minimal-arm64-rpi64_v2022dev2.0.img.licenses.gz