OpenChain Compliance Automation

Automating the generation of OpenChain compliance artefacts

Alexander Murphy

2023-09-11

OpenChain

International standard (ISO/IEC 5230) lists requirements for a quality open source licence compliance program. The project homepage is here: https://www.openchainproject.org/; and the specificaiton and workgroup manage the development publicly on GitHub here: https://github.com/OpenChain-Project/License-Compliance-Specification.

The standard provides the following definition:

> 2.1 - compliance artifacts a collection of artifacts that represent the output of a compliance program and accompany the supplied software
>
> Note: The collection may include (but is not limited to) one or more of the following: attribution notices, source code, build and install scripts, copy of licenses, copyright notices, modification notifications, written offers, open source component bill of materials, and SPDX documents.

These data are what we will automate in this demonstration (attribution notices and licence texts specifically, other artefacts may be required but this is not an exhaustive exercise, regardless, it is highly unlikely that the steps in this example will map exactly to your own use-case).

Tools

OpenChain is a non-prescriptive standard. To (ab)use computer science terminology, OpenChain is declarative in nature, it tells you *what* to do, but you decide *how* to go about it. This example is a generalised method (rather, a collection of tools and processes) used frequently at Orcro to generate compliance artefacts.

Spelling

Artefacts - British engligh, artifacts, american english. Former used throughout.

Scancode toolkit

Our favorite OSS SCA tool (https://github.com/nexB/scancode-toolkit). Fully featured, well supported, active community. Straightforward to integrate into existing pipelines (run Python script direct from the repo).

Runs comprehensive battery of regex patterns on source code for licence/copyright matches. Huge test suite.

R

Statistical computing programming language. This is the author's personal preference (a dark art) although Python would work, and bash scripts would also work. No libraries are required, just Base R.

Scripts

These are the "glue" that holds togheter the pipeline. These will all be written in R.

Git

The code we review will be located in a Git repository. *this* repo (OpenChain...) contains a submodule to our demonstration project, OpenBLAS (more on this project below).

Overview

We begin by identifying what data we need to automate. This step is usually resource-intensive, so an arbitrary (and vetted) software product will be used for this example.

Then we break-down the full pipeline into "components", and semi-manually run these to illustrate what the tools are doing. I.e., we run through producing the compliance artefacts once step-by-step, and see what happens when we run a "generate compliance artefacts" command in CI.

The output of the process (the compliance artefacts) will be shown. These will be in the form of plain text files, which can be compressed if required.

Setting expectations

Implementing the process on a CI system is not covered.

This only covers generation of compliance artefacts for the application itself. It is likely that additional FOSS is used in any shipped product (Docker container, GPU drivers, etc.) but this process is for the application layer only.

We won't consider legal requirements, such as how the jurisdiction your code is shipped to may affect your obligations.

We won't consider engineer considerations, such as caching SCA results.

The sample project

OpenBLAS

BLAS (basic linear algebra subprograms) is a widely used library for matrix and vector calculations. It's open source, the reference implementation (from https://www.netlib.org/blas/) isn't "licensed" per se, rather it has some unusual wording about being free to use but to add notices if anything is modified and to provide acknowledgement for the authors.

OpenBLAS has the BSD-3-Clause out-licence, and the full source (which we look at here) is available on GitHub (https://github.com/xianyi/OpenBLAS).

Identifying the requried artefacts

Questions to ask

What is *distributed*?

Are there any *dependencies*?

How will the code be shipped?

Are there any *snippets* present?

In this case, all of the source code is provided in this monorepo. We can run our tools here without requiring any additional downloads.

Recall that build tools won't end up in the distributed binary, therefore you have no need to concern yourselves with licences for them (typically, but never seen a case where this is not the case).

Licences are BSD-3-Clause, therefore our obligations are simply: Provide copyright notices (all of them!) and the licence text. * another obligation is to not use the name of the project in marketing, but this does not impact the compliance artefacts r.e. openchain

We also see that it doesn't matter whether the code is distributed as source or in binary form. The artefacts are required* note that if distributed as source code, then the compliance materials *will be provided automatically as they are contained within the source itself!*

For this we assume the binary is distributed, and providing source code is unsuitable for some reason.

Note: providing the full source code for components licensed permissively is not *required* but it may be one way to comply! Although, if you have more than one component with different obligations then things can get messy quickly, hence, automation...

Initial SCA

It is prudent to manually run a SCA before implementing any automation. Like shopping for bug-spray, it is important to know *what will work* before spending any money (or developer time, resources, etc.) on the thing. It would be disastrous to implement compliance automation and a year down-the-line realise that it wasn't identifying issues appropriately (or even worse, it *was* identifying them but you didn't notice)!

Scancode can be run from the command-line. So a developer, who is working inside the development repo, can simply use:

```
pwd
/home/demo/OpenChain-Artefact-Automation/OpenBLAS/
scancode -clp -n 18 --csv ScanOut.csv .
```

which will run scancode on the source repo. This assume that scancode has been installed (added to path) et cetera (see Appendix A for simple scancode installation instructions).

As this is the first time that we have extracted copyright information using scancode, we should take a look at the data it has provided us. It is typically of high quality, but there are sometimes things that are easy to check and verify.

There are various ways of doing this, if you run the `--csv` output format then you can even load up the data into Excel and look at it that way. Here though, we'll use R.

```
copyright_data <- read.csv("ScanOut.csv") # load the scan results
licences <- unique(copyright_data$license_expression) # summarise 'unique' licence (so we only see one
licences # display them
```

```
##  [1] ""                                "bsd-2-clause-views"
##  [3] "bsd-new"                         "blas-2017"
##  [5] "bsd-new AND bsd-2-clause-views" "bsd-simplified"
##  [7] "cc0-1.0"                         "bsd-3-clause-open-mpi"
##  [9] "blas-2017 AND bsd-new"           "proprietary-license"
## [11] "mit"                             "apache-2.0"
```

This output is using Scancode's internal naming system, but it is based on SPDX identifiers. Look at `apache-2.0` this is simply the SPDX ID with a lower case `a`.

There doesn't appear to be too much to review here, these are all permissive or variants of, except for that results of `proprietary-license`. You should not be too alarmed at these results (at least in the first instance) because Scancode is quite *greedy* with its results. It will generate more false positives than omissions (it's generally better this way, as you can manually review the results, you cannot manually review what you do not know exists)!

Let's double check this result, to see how often it appears, and what is triggering this result in the scan.

```
copyright_data[copyright_data$license_expression == "proprietary-license", ]["path"] # selecting result
```

```
##                                         path
## 15409 OpenBLAS/lapack-netlib/LAPACKE/README
## 15410 OpenBLAS/lapack-netlib/LAPACKE/README
```

```
# and then from the results, extract the path of the file (*where* the scan found it)
```

So the `proprietary-license` results came from the LAPACKE readme. We'll assume that this check went well for the purposes of this demonstration, but there are a few more notes on this in Appendix 2, including the README file itself.

It would be good practice to verify all of the licence results, at least by sample, but as everything is permissive here there's unlikely to be any major issues so we'll proceed with generating the artefacts.

Generating artefacts

Now that the scan results have been collected and reviewed for problems, we can use whatever means at our disposal to generate appropriate artefacts. This is an inherently 'wooly' statement, because it depends on the licences as to what artefacts must be provided. There are also various way in which the artefacts *could* be provided for any particular licence. What follows is appropriate for permissive licences, OpenChain compliance, and note that no source code provision is required.

Some scripts have been developed by Orcro for this particular task, and are available under open source licences for you to freely use yourselves if you wish: https://github.com/galacticalex/compliance-artefact-generator, they are also as a submodule in this demo repository, we use some functions from these now.

Appendix A: Scancode installation

`cd` to the installation directory, then

```
git clone --depth=1 https://github.com/nexB/scancode-toolkit.git
cd ./scancode
./scancode --help
```

Running `./scancode --help` will setup scancode for first use. There are other methods in the documentation (https://scancode-toolkit.readthedocs.io/en/stable/) but the above is, clearly, very simple. The above can be copied into a `.sh` script, Docker `RUN` directive, et cetera, as required.

Appendix B: Intel's `proprietary-license` result

The full README is here:

```
cat OpenBLAS/lapack-netlib/LAPACKE/README
```

```
## -----------------------------------------------------------------------------
##                              C Interface to LAPACK
##                                    README
## -----------------------------------------------------------------------------
## Introduction
## -----------------------------------------------------------------------------
##
## This library is a part of reference implementation for the C interface to
## LAPACK project according to the specifications described at the forum for
## the Intel(R) Math Kernel Library (Intel(R) MKL):
## http://software.intel.com/en-us/forums/showthread.php?t=61234
```

```
##
## This implementation provides a native C interface to LAPACK routines available
## at www.netlib.org/lapack to facilitate usage of LAPACK functionality
## for C programmers.
## This implementation introduces:
## - row-major and column-major matrix layout controlled by the first function
##   parameter;
## - an implementation with working arrays (middle-level interface) as well as
##   without working arrays (high-level interface);
## - input scalars passed by value;
## - error code as a return value instead of the INFO parameter.
##
## This implementation supports both the ILP64 and LP64 programming models,
## and different complex type styles: structure, C99.
##
## This implementation includes interfaces for the LAPACK-3.2.1 Driver and
## Computational routines only.
##
## ---------------------------------------------------------------------------
## Product Directories
## ---------------------------------------------------------------------------
##
## The installation directory of this package has the following structure:
##
## src                - C interface source files
## utils              - C interface auxiliary files
## include            - header files for C interface
##
## ---------------------------------------------------------------------------
## Installation
## ---------------------------------------------------------------------------
##
## The reference code for the C interface to LAPACK is built similarly to the
## Basic Linear Algebra Subprograms (BLAS) and LAPACK. The build system produces
## a static binary lapacke.a.
##
## You need to provide a make.inc file in the top directory that defines the
## compiler, compiler flags, names for binaries to be created/linked to. You may
## choose the appropriate LP64/ILP64 model, convenient complex type style,
## LAPACKE name pattern, and/or redefine system malloc/free in make.inc. Several
## examples of make.inc are provided.
##
## After setting up the make.inc, you can build C interface to LAPACK by typing
##
## make lapacke
##
## ---------------------------------------------------------------------------
## Handling Complex Types
## ---------------------------------------------------------------------------
##
## The interface uses complex types lapack_complex_float/lapack_complex_double.
## You have several options to define them:
##
## 1) C99 complex types (default):
```

```
##
## #define lapack_complex_float    float _Complex
## #define lapack_complex_double   double _Complex
##
## 2) C structure option (set by enabling in the configuration file):
## -DHAVE_LAPACK_CONFIG_H  -DLAPACK_COMPLEX_STRUCTURE
##
## typedef struct { float real, imag; } _lapack_complex_float;
## typedef struct { double real, imag; } _lapack_complex_double;
## #define lapack_complex_float  _lapack_complex_float
## #define lapack_complex_double _lapack_complex_double
##
## 3) C++ complex types (set by enabling in the configuration file):
## -DHAVE_LAPACK_CONFIG_H -DLAPACK_COMPLEX_CPP
##
## #define lapack_complex_float std::complex<float>
## #define lapack_complex_double std::complex<double>
##
## You have to compile the interface with C++ compiler with C++ types.
##
## 4) Custom complex types:
## -DLAPACK_COMPLEX_CUSTOM
##
## To use custom complex types, you need to:
## - Define lapack_complex_float/lapack_complex_double types on your own.
## - Optionally define lapack_make_complex_float/lapack_make_complex_double_real
##   functions if you want to build the testing suite supplied. Use these
##   functions for the testing system. Their purpose is to make a complex value of
##   a real part re, imaginary part im. The prototypes are as follows:
##
##    lapack_complex_float lapack_make_complex_float( float re, float im );
##    lapack_complex_double lapack_make_complex_double( double re, double im );
##
## ----------------------------------------------------------------------------
## Choosing ILP64 Data Model
## ----------------------------------------------------------------------------
## To choose ILP64 data model (set by enabling in the configuration file), use the
## following options:
##
## -DHAVE_LAPACK_CONFIG_H  -DLAPACK_ILP64
##
## ----------------------------------------------------------------------------
## Using Predicate Functions
## ----------------------------------------------------------------------------
##
## The functions
##
## lapacke_?gees/lapacke_?gees_work
## lapacke_?geesx/lapacke_?geesx_work
## lapacke_?geev/lapacke_?geev_work
## lapacke_?geevx/lapacke_?geevx_work
##
## require the pointer to a predicate function as an argument of a predefined type
## such as:
```

```
##
## typedef lapack_logical (*LAPACK_S_SELECT2) ( const float*, const float* );
##
## The purpose and format of these predicate functions are described in the LAPACK
## documentation. This interface passes the pointer to the corresponding LAPACK
## routine as it is.
##
## Be cautious with return values of the logical type if you link against LAPACK
## compiled with Fortran compiler. Whereas all non-zero values are treated as TRUE
## generally, some Fortran compilers may rely on a certain TRUE value, so you will
## have to use the same TRUE value in the predicate function to be consistent with
## LAPACK implementation.
##
## ------------------------------------------------------------------------------
## Implementation Details
## ------------------------------------------------------------------------------
##
## The current C interface implementation consists of wrappers to LAPACK routines.
## The row-major matrices are transposed on entry to and on exit from the LAPACK
## routine, if needed. Top-level interfaces additionally allocate/deallocate
## working space on entry to and on exit from the LAPACK routine.
##
## Because of possible additional transpositions, a routine called with
## this interface may require more memory space and run slower than the
## corresponding LAPACK routine.
##
## ------------------------------------------------------------------------------
## Disclaimer and Legal Information
## ------------------------------------------------------------------------------
##
## INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL(R)
## PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO
## ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT.  EXCEPT
## AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS,
## INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS
## OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS
## INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR
## PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR
## OTHER INTELLECTUAL PROPERTY RIGHT.  UNLESS OTHERWISE AGREED IN WRITING
## BY INTEL, THE INTEL PRODUCTS ARE NOT DESIGNED NOR INTENDED FOR ANY
## APPLICATION IN WHICH THE FAILURE OF THE INTEL PRODUCT COULD CREATE A
## SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR.
##
## Intel may make changes to specifications and product descriptions at
## any time, without notice. Designers must not rely on the absence or
## characteristics of any features or instructions marked "reserved" or
## "undefined." Intel reserves these for future definition and shall have
## no responsibility whatsoever for conflicts or incompatibilities
## arising from future changes to them. The information here is subject
## to change without notice. Do not finalize a design with this
## information.
##
## The products described in this document may contain design defects or
## errors known as errata which may cause the product to deviate from
```

```
## published specifications. Current characterized errata are available
## on request.
##
## Contact your local Intel sales office or your distributor to obtain
## the latest specifications and before placing your product order.
## Copies of documents which have an order number and are referenced in
## this document, or other Intel literature, may be obtained by calling
## 1-800-548-4725, or go to http://www.intel.com/design/literature.htm
##
## Intel processor numbers are not a measure of performance. Processor
## numbers differentiate features within each processor family, not
## across different processor families. See
## http://www.intel.com/products/processor_number for details.
##
## This document contains information on products in the design phase of
## development.
##
## BunnyPeople, Celeron, Celeron Inside, Centrino, Centrino Atom,
## Centrino Atom Inside, Centrino Inside, Centrino logo, Core Inside,
## FlashFile, i960, InstantIP, Intel, Intel logo, Intel386, Intel486,
## IntelDX2, IntelDX4, IntelSX2, Intel Atom, Intel Atom Inside, Intel
## Core, Intel Inside, Intel Inside logo, Intel. Leap ahead., Intel. Leap
## ahead.  logo, Intel NetBurst, Intel NetMerge, Intel NetStructure,
## Intel SingleDriver, Intel SpeedStep, Intel StrataFlash, Intel Viiv,
## Intel vPro, XScale, IPLink, Itanium, Itanium Inside, MCS, MMX, Oplus,
## OverDrive, Intel PDCharm, Pentium, Pentium Inside, skoool, Sound Mark,
## The Journey Inside, VTune, Xeon, and Xeon Inside are trademarks of
## Intel Corporation in the U.S. and other countries.
##
##
## * Other names and brands may be claimed as the property of others.
##
## Copyright (C) 2011, Intel Corporation. All rights reserved.
```

This requires some further analysis, however, it is highly likely that in the “INTEL TERMS OF SERVICE” that such code is licensed freely with the disclaimer.