Software Bill of Materials and Supply Chain with the Yocto Project

Joshua Watt
Embedded Linux Conference
June 23, 2022
About Me

- Worked at Garmin since 2009
- Using OpenEmbedded & Yocto Project since 2016
- Member of the OpenEmbedded Technical Steering Committee (TSC)
- Joshua.Watt@garmin.com
- JPEWhacker@gmail.com
- IRC (OFTC or libera): JPEW
- Twitter: @JPEW_dev
- LinkedIn: joshua-watt-dev
Yocto Project and OpenEmbedded

**OpenEmbedded**
- Community project
- OpenEmbedded core layer
- Build system (bitbake)

**Yocto Project**
- Linux Foundation project
- Poky reference distribution
- Runs QA tests
- Manages release schedule
- Provides funding for personnel
- Documentation
Outline

- Software Supply Chain
- OpenEmbedded Build Flow
- Software Bill of Materials
- SPDX Contents
- Reproducible Builds
- Buildtools Tarball
Software Supply Chain
Why is the Software Supply Chain Important?

- What's in my Software?
  - Where did it come from?
  - What version is it?
- Am I complying with Software Licenses?
- Has it been tampered with?
- Is it vulnerable to exploits?
- Can deliverables be traced back to their code?
OpenEmbedded Build Flow
Build Images from Source Code

Source

Metadata

Policy

bitbake

Target Image

Widget
Simplified Build Flow

Host Tools

Source

Recipe Metadata

Recipe Metadata

Recipe Metadata
Simplified Build Flow

Host Tools → Recipe Metadata → Native tools & Cross Compiler

Source

Recipe Metadata

Recipe Metadata

Recipe Metadata
Simplified Build Flow

- Host Tools
- Source
- Recipe Metadata
- Native tools & Cross Compiler
- Target Packages
- Recipe Metadata

Diagram showing the flow from Host Tools to Source, then to Recipe Metadata, Native tools & Cross Compiler, and finally to Target Packages, with connections back to Source and Recipe Metadata.
Simplified Build Flow

Host Tools

Recipe Metadata

Source

SHA256

Native tools & Cross Compiler

Recipe Metadata

Target Packages

Target Image

Recipe Metadata

Source

SHA256

Recipe Metadata

SHA256

Recipe Metadata

SHA256
Simplified Build Flow

- Host Tools
- Source
  - SHA256
- Recipe Metadata
- Native tools & Cross Compiler
  - SHA256
- Recipe Metadata
- Target Packages
  - SHA256
- Recipe Metadata
- Target Image
Simplified Build Flow

Host Tools → Recipe Metadata → Native tools & Cross Compiler → Source

Target Packages → Recipe Metadata → Target Image

SHA256 connects the dots, tracing the target image back to the code (and metadata)
Software Bill of Materials
**"Nutrition Information" for Software**

<table>
<thead>
<tr>
<th>Ingredient(s)</th>
<th>bash, Linux, u-boot, sshd, openssl, busybox</th>
</tr>
</thead>
</table>

## SBoM Facts

| Serving Size | 1 |

<table>
<thead>
<tr>
<th>CVEs Patched</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2019-18276</td>
<td></td>
</tr>
<tr>
<td>CVE-2014-0160</td>
<td></td>
</tr>
<tr>
<td>Patches Applied</td>
<td>30</td>
</tr>
</tbody>
</table>

An SBoM is a method of describing the information about a Software Supply Chain using a standardized encoding that allows for easy exchange of data.

Multiple different SBoM formats may describe the same Software Supply Chain.
What is an SBoM?

Source: NTIA's Framing Software Component Transparency: Establishing a Common Software Bill of Material (SBOM)
Simplified Build Flow

Host Tools

Source

Recipe Metadata

Native tools & Cross Compiler

Recipe Metadata

Source

Recipe Metadata

Target Packages

Recipe Metadata

Target Image
Simplified Build Flow

Host Tools

Source

Recipe Metadata

Source

Recipe Metadata

Recipe Metadata

Target Packages

Target Image

Carol's Compression Engine v3.1

Bob's Browser v2.2
Recipe Metadata

Recipes already contain much of the data desired in a SBoM

- Version
- Source code URL
- Licenses
- Build time dependencies
- Run time dependencies
- CVEs patched
- Source Files
- Package Files
- ...

All of this information is authoritative (no guessing)
Generating SBoMs

OpenEmbedded has support for generating SBoMs in SPDX JSON format:

$ . oe-init-build-env
$ echo 'INHERIT += "create-spdx"' >> conf/local.conf
$ bitbake core-image-minimal
SPDX Generation

Host Tools

Recipe Metadata

Source

Recipe Metadata

Source

Native tools & Cross Compiler

SPDX

Target Packages

SPDX

Target Image

SPDX Archive
Yocto SPDX Features

- Declared License
  - With License Text if not a known SPDX license
- Homepage URL
- Download URL(s)
- CVEs fixed
- CPE
- Summary
- Description
- Source File Listing with Checksums
- Source file SPDX licenses
- Packages
- Package files with Checksums

- Package file GENERATED_FROM (from debug data)
- Build time dependencies
- Runtime dependencies
- Source code archive for analysis by other tools (e.g. Fossology)
What can we generate SPDX documents for?

TL; DR - Anything we can build

- "On target" C/C++/Fortran etc.
- "native" build tools & cross compiler
- Linux Kernel
- Target images
- SDKs
- Container Images
- VM Images
- Rust
- Go
Configuration Knobs

- **SPDX_INCLUDE_SOURCES = "1"**
  - Includes patched source files from "s" in Recipe SPDX with a "CONTAINS" relationship
  - Off by default because the SPDX is huge when turned on

- **SPDX_ARCHIVE_SOURCES = "1"**
  - Creates a tarball of the sources, useful for running against other tools (e.g. fossology)
  - Off by default

- **SPDX_ARCHIVE_PACKAGED = "1"**
  - Creates a tarball of the packaged output files, useful for running against other tools
  - Off by default

- **SPDXPRETTY = "1"**
  - Make output more human readable (master branch only)
Publishing Results on the Internet

- Set:
  - SPDX_SUPPLIER = "Organization: My Company" (1)
  - SPDX_NAMESPACE_PREFIX = "http://my.company.com/spdx/" (2)
  - SPDX_UUID_NAMESPACE = "my.company.com"

[1]: https://spdx.github.io/spdx-spec/package-information/#75-package-supplier-field
SPDX Contents
Generated SPDX Files

$ ls -l tmp/deploy/images/qemux86-64/*.spdx.*
core-image-minimal-qemux86-64.spdx.json
core-image-minimal-qemux86-64.spdx.tar.zst
core-image-minimal-qemux86-64.spdx.index.json
$ ls -l tmp/deploy/images/qemux86-64/*.spdx.*
core-image-minimal-qemux86-64.spdx.json
core-image-minimal-qemux86-64.spdx.tar.zst
core-image-minimal-qemux86-64.spdx.index.json

The SPDX JSON file for the image itself
Generated SPDX Files

$ ls -l tmp/deploy/images/qemux86-64/* .spdx.*
core-image-minimal-qemux86-64.spdx.json
core-image-minimal-qemux86-64.spdx.tar.zst
core-image-minimal-qemux86-64.spdx.index.json

Compressed Tarball containing all of the SPDX documents for the image itself, all packages that were installed in the image, all recipes that generated those packages, and the index file.
Generated SPDX Files

$ ls -l tmp/deploy/images/qemux86-64/*.*.spdx.*
core-image-minimal-qemux86-64.spdx.json
core-image-minimal-qemux86-64.spdx.tar.zst
core-image-minimal-qemux86-64.spdx.index.json

Index file that lists all of the SPDX JSON files in the SPDX archive
SPDX Archive Contents

$ tar -tvf core-image-minimal-qemux86-64.spdx.tar.zst
  core-image-minimal-qemux86-64-20220614012543.spdx.json
  util-linux-1sblk.spdx.json
  runtime-util-linux-1sblk.spdx.json
  util-linux-unshare.spdx.json
  runtime-util-linux-unshare.spdx.json
  recipe-util-linux.spdx.json
  ...
  index.json
$ tar -tvf core-image-minimal-qemux86-64.spdx.tar.zst
core-image-minimal-qemux86-64-20220614012543.spdx.json
util-linux-lsblk.spdx.json
runtime-util-linux-lsblk.spdx.json
util-linux-unshare.spdx.json
runtime-util-linux-unshare.spdx.json
recipe-util-linux.spdx.json
...
index.json

Image SPDX file (from before)
SPDX Archive Contents

$ tar -tvf core-image-minimal-qemux86-64.spdx.tar.zst
    core-image-minimal-qemux86-64-20220614012543.spdx.json
    util-linux-lsblk.spdx.json
    runtime-util-linux-lsblk.spdx.json
    util-linux-unshare.spdx.json
    runtime-util-linux-unshare.spdx.json
    recipe-util-linux.spdx.json
    ...
    index.json

Archive Index file (from before)
SPDX Archive Contents

$ tar -tvf core-image-minimal-qemux86-64.spdx.tar.zst
   core-image-minimal-qemux86-64-20220614012543.spdx.json
   util-linux-1sblk.spdx.json
   runtime-util-linux-1sblk.spdx.json
   util-linux-unshare.spdx.json
   runtime-util-linux-unshare.spdx.json
   recipe-util-linux.spdx.json
   ...
   index.json

SPDX file describing packages installed in the image
SPDX Archive Contents

$ tar -tvf core-image-minimal-qemux86-64.spdx.tar.zst
  core-image-minimal-qemux86-64-20220614012543.spdx.json
  util-linux-1sblk.spdx.json
  runtime-util-linux-1sblk.spdx.json
  util-linux-unshare.spdx.json
  runtime-util-linux-unshare.spdx.json
  recipe-util-linux.spdx.json
  ...
  index.json

SPDX file describing runtime dependencies of packages installed in the image
SPDX Archive Contents

$ tar -tvf core-image-minimal-qemux86-64.spdx.tar.zst
  core-image-minimal-qemux86-64-20220614012543.spdx.json
  util-linux-lsblk.spdx.json
  runtime-util-linux-lsblk.spdx.json
  util-linux-unshare.spdx.json
  runtime-util-linux-unshare.spdx.json
  recipe-util-linux.spdx.json
  ...
  index.json

SPDX file describing the recipe and source code used to generate packages
Future Improvements

- Improve Relationships (in talks with upstream SPDX)
- Pull in SPDX/SBoM from upstream source code (e.g. `reuse`)
- More SPDX fields
- Include information about how recipes are built (e.g. CFLAGS, etc.)
Reproducible Builds
Why do we need reproducible builds?

- Resist attack
  - What binaries need more scrutiny?
- Compiler Trust
  - [Diverse Double-Compilation](https://reproducible-builds.org/docs/buy-in/) (David A. Wheeler) requires reproducible builds
- Quality Assurance
  - Rare timing bugs, race conditions, locale dependencies
- Smaller Binary Differences
  - Better delta updates
- Increased Development Speed
  - No need to rebuild if nothing has changed

[https://reproducible-builds.org/docs/buy-in/](https://reproducible-builds.org/docs/buy-in/)
Binary output should associate with recipe hashes

Host Tools

Recipe Metadata

Source

SHA256

Native tools & Cross Compiler

Recipe Metadata

SHA256

Target Packages

Recipe Metadata

SHA256

Target Image

??

??
Reproducibility Testing

- Yocto Autobuilder tests regularly for regressions
- [https://www.yoctoproject.org/reproducible-build-results/](https://www.yoctoproject.org/reproducible-build-results/)
- ~11,000 *target* packages
- 3 Package formats (ipk, deb, rpm)
- Multiple build hosts (Fedora, Ubuntu, CentOS, Debian)
  - Ensures cross-host builds are reproducible!
- Automatic [diffoscope](https://www.bazaarcommunity.org/tools/diffoscope) HTML output for packages that are not reproducible
Extending Quality Assurance Test

- The QA test for reproducibility is designed to be easy to extend and run for testing your own images:

```bash
$ cat lib/oeqa/selftest/cases/myreproducible.py
from oeqa.selftest.cases.reproducible import ReproducibleTests

class MyReproTests(ReproducibleTests):
    images = ['my-image']

$ oe-selftest -r myreproducible
```
Buildtools Tarball
"It's SBoMs all the way down"
Images

Target Image

buildtools
Buildtools replaces Host tools

- Source
- SHA256
- Recipe Metadata
- Native tools & Cross Compiler
- SHA256
- Target Packages
- SHA256
- Target Image
- Recipe Metadata
- Source
- SHA256
Buildtools replaces Host tools

- Source
- SHA256
- Recipe Metadata
- SHA256
- Native tools & Cross Compiler
- Target Packages
- SHA256
- Target Image

buildtools
Special Thanks

- Saul Wold (Linux Kernel SPDX Generation)
- Ross Burton (License Work)
- Andres Beltran (SDK Support)
- Richard Purdie (Yocto Project Technical Lead)
- Many others for various fixes & improvements!
Getting Involved

- Libera IRC
  - #yocto
  - #oe
- **Weekly technical meeting**
  - Every Tuesday at 8:00 AM Pacific Time
- **Weekly Bug Triage**
  - Every Thursday at 7:30 AM Pacific Time
- Happy Hour
  - Last Wednesday of every Month ([Calendar](#))
- Yocto Project Summit
  - Twice yearly
Questions?