

KernelSBOM

How to create the SBOM for the Linux kernel

About myself

Maximilian Huber

Principal Consultant

TNG Technology Consulting GmbH

maximilian.huber@tngtech.com

GitHub: [@maxhbr](https://github.com/maxhbr)

[linkedin.com/in/maxhbr](https://www.linkedin.com/in/maxhbr)



Task:

Create an SBOM for a Linux Kernel Build

The KernelSbom tool

A tool that analyzes a kernel build and produces SBOM documents



<https://github.com/TNG/KernelSbom>

(The name and home might change)

Example Call:

```
$ export SRCARCH=x86  
$ python3 sbom/sbom.py \  
  --src-tree path/to/linux \  
  --obj-tree path/to/kernel_build \  
  --roots arch/x86/boot/bzImage \  
  --generate-spdx
```

After merge:

```
$ make sbom
```

Step 1

Analyze the Build

We start with "roots" and build a graph. Roots are usually the distributable artifacts generated by the build like `arch/x86/boot/bzImage` and `*.ko` files.

Next to output files, the build creates files like `arch/x86/boot/.bzImage.cmd` that describe how the outputs were built. This allows us to reconstruct the build graph.

`.cmd` files look like:

```
savedcmd_${target} := ${make-cmd}
```

or

```
savedcmd_${target} := ${make-cmd}

source_${target} := ${source}

deps_${target} := \
    ${dependency1} \
    ${dependency2} \
    ${dependency3} \
    ...
```

Parsing the savedcmd_{\$target} command

Example: arch/x86/boot/compressed/vmlinux

```
$ cat arch/x86/boot/compressed/.vmlinux.cmd
savedcmd_arch/x86/boot/compressed/vmlinux := \
    ld -m elf_i386 --no-ld-generated-unwind-info
[...]
    arch/x86/boot/compressed/kernel_info.o \
    arch/x86/boot/compressed/head_32.o \
    arch/x86/boot/compressed/misc.o \
    arch/x86/boot/compressed/string.o \
    arch/x86/boot/compressed/cmdline.o \
    arch/x86/boot/compressed/error.o \
    arch/x86/boot/compressed/piggy.o \
    arch/x86/boot/compressed/cpuflags.o \
    -o arch/x86/boot/compressed/vmlinux
```

Depends on:

- arch/x86/boot/compressed/kernel_info.o
- arch/x86/boot/compressed/head_32.o
- ...

Parsing source_\${target} and deps_\${target}

Example: arch/x86/boot/compressed/kernel_info.o

```
$ cat arch/x86/boot/compressed/.kernel_info.o.cmd
savedcmd_arch/x86/boot/compressed/kernel_info.o := gcc -Wp,-MMD,arch/x86/boot/compressed/.kernel_info.o.d -nostdinc -I.

source_arch/x86/boot/compressed/kernel_info.o := ../arch/x86/boot/compressed/kernel_info.S

deps_arch/x86/boot/compressed/kernel_info.o := \
    ../../include/linux/compiler-version.h \
    ../../include/linux/kconfig.h \
    ../../include/linux/hidden.h \
    ../../arch/x86/include/uapi/asm/bootparam.h \
    ../../arch/x86/include/asm/setup_data.h \
    ../../arch/x86/include/uapi/asm/setup_data.h \
[ ... ]
```

Depends on:

- ../../include/linux/compiler-version.h
- ../../include/linux/kconfig.h
- ...

We also search for `.incbin` statements in `.S` files

These dependencies are not transparent in the `.cmd` files. An example is:

```
$ cat arch/riscv/boot/loader.S
/* SPDX-License-Identifier: GPL-2.0 */

.align 4
.section .payload, "ax", %progbits
.globl _start
_start:
.incbin "arch/riscv/boot/Image"
```

This includes the build output `arch/riscv/boot/Image`, which itself has a `.cmd` file which can be followed further.

Some dependencies are hardcoded

Makefiles and Kbuild files are not parsed due to their complex structure. Dependencies only defined in these files need to be hardcoded. The current list that is sufficient for tinyconfig is:

```
HARDCODED_DEPENDENCIES: dict[str, list[str]] = {  
    "include/generated/rq-offsets.h": ["kernel/sched/rq-offsets.s"],  
    "include/generated/bounds.h": ["kernel/bounds.s"],  
    "include/generated/asm-offsets.h": ["arch/{arch}/kernel/asm-offsets.s"],  
    "kernel/sched/rq-offsets.s": ["include/generated/asm-offsets.h"],  
}
```

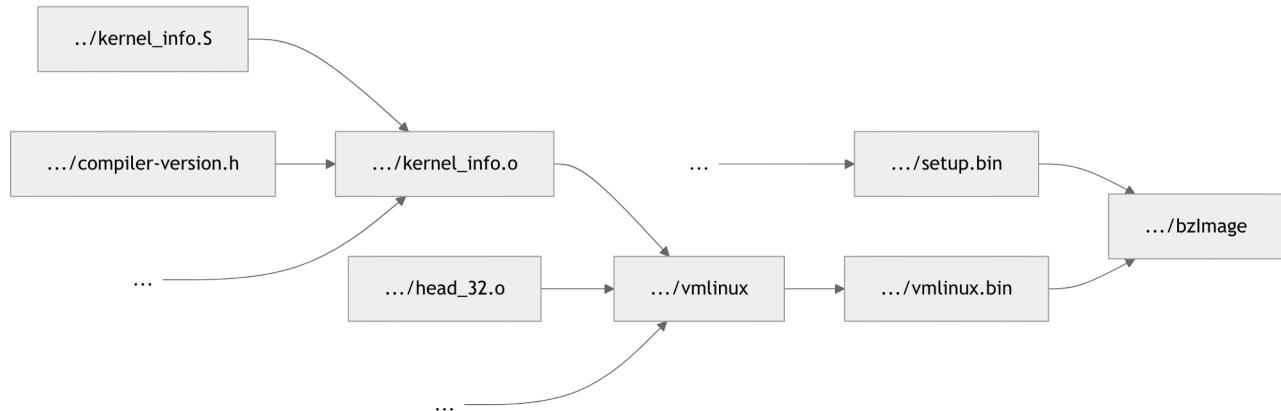
 See #50 for details

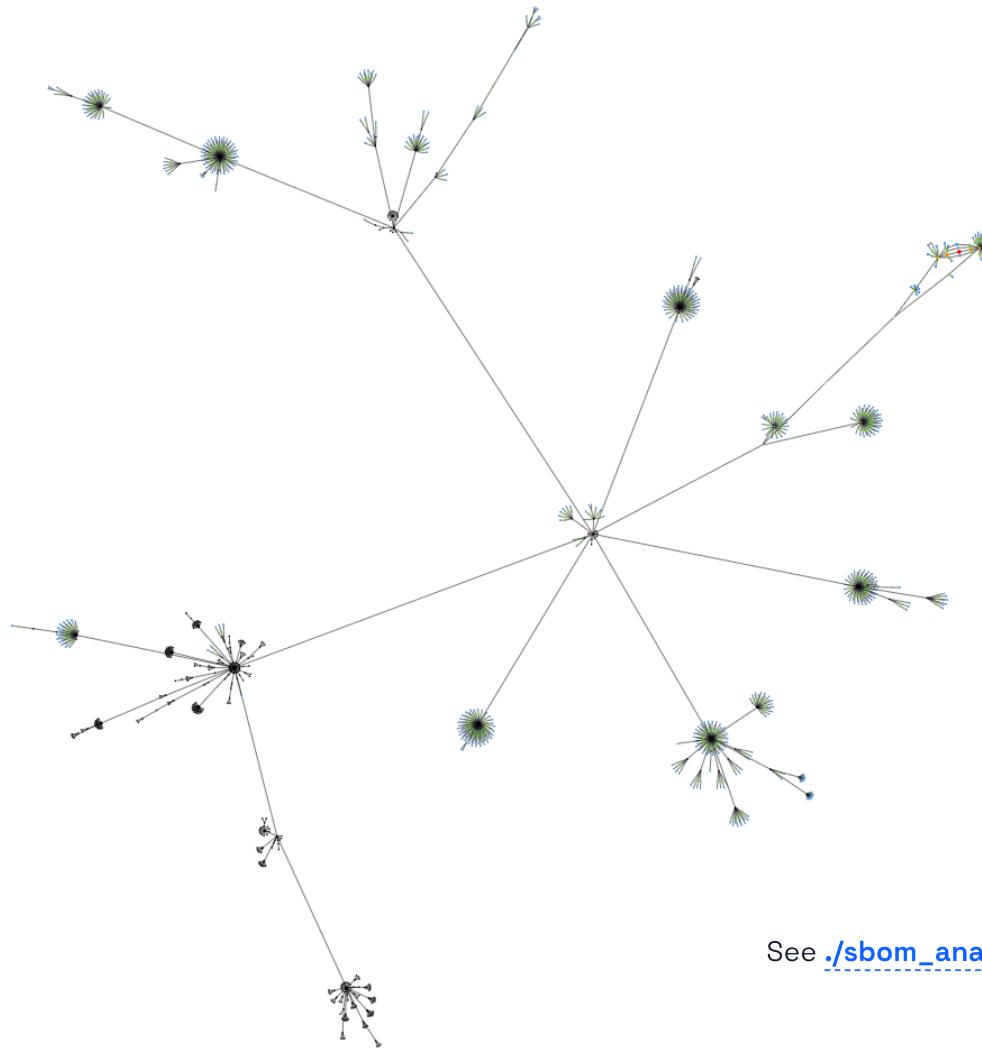
This list is incomplete.

Summary

The graph is created by starting at roots and finding dependencies with the following strategies:

- Parsing the `.cmd` files
 - Parsing the `savedcmd_${target}` command
 - Parsing `source_${target}` and `deps_${target}`
 - Parsing the actual files
 - Parsing `.incbin` statements in `.S` files
 - Applying some hardcoded dependencies





See [./sbom_analysis/cmd_graph_visualization/](#)

Step 2

Serialize as SBOM

Using the SPDX 3.0.1 standard.

- Established and widely used.
- A Linux Foundation project.
- Is targeting ISO again.
- Uses a JSON-LD serialization that we can just treat as JSON.

✉ Home

Copyright

Introduction

1. Scope

2. References

3. Symbols

4. Terms and definitions

5. Conformance

6. Model and serializations

MODEL

⊕ Core

⊕ Software

⊕ Security

⊕ Licensing

⊕ SimpleLicensing

⊕ ExpandedLicensing

⊕ Dataset

⊕ AI

⊕ Build

⊕ Lite

⊕ Extension

ANNEXES

A. RDF model definition and diagrams

B. SPDX license expressions

C. SPDX License List matching guidelines

D. SPDX Lite

E. Package URL specification

LICENSES



GitHub

Next »

/ Home

The System Package Data Exchange® (SPDX®) Specification Version 3.0.1

Copyright © 2010-2024, The Linux Foundation and its Contributors, including SPDX Model contributions from OMG and its Contributors.

With thanks to Adam Cohn, Adolfo García Veytia, Alan Tse, Alexios Zavras, Andrew Back, Ann Thornton, Armin Tänzer, Arthit Suriyawongkul, Ayumi Watanabe, Basil Peace, Bill Schineller, Bradlee Edmondson, Brandon Lum, Bruno Corneç, Ciaran Farrell, Daniel German, David Edelsohn, David Kemp, David A. Wheeler, Debra McGlade, Dennis Clark, Dick Brooks, Ed Warnicke, Eran Strod, Eric Thomas, Esteban Rockett, Gary O'Neill, Gopi Krishnan Rajbahadur, Guillaume Rousseau, Hassib Khanafar, Henk Birkholz, Hiroyuki Fukuchi, Itaru Hosomi, Jack Manbeck, Jaime Garcia, Jeff Licquia, Jeff Luszcz, Jeff Schutt, Jilayne Lovejoy, John Ellis, Jonas Oberg, Joshua Watt, Kamsang Salima, Karen Bennet, Karen Copenhaver, Kate Stewart, Kevin Mitchell, Kim Weins, Kirsten Newcomer, Kouki Hama, Kris Reeves, Liang Cao, Lon Hohberger, Marc-Etienne Vargenau, Mark Gisi, Marshall Clow, Martin Michlmayr, Martin von Willebrand, Mark Atwood, Matija Šuklje, Matt Germonprez, Maximilian Huber, Meret Behrens, Michael J. Herzog, Michel Ruffin, Nicole Pappler, Nisha Kumar, Nobuyuki Tanaka, Norio Kobota, Nuno Brito, Oliver Fendt, Paul Madick, Peter Williams, Phil Robb, Philip Koltun, Philip Odence, Philippo Ombradonna, Pierre Lanciato, Pana

Three interlinked files

Output SBOM: sbom-output.spdx.json

Describes the packages for the Linux kernel and the kernel modules.

Source SBOM: sbom-source.spdx.json

Describes the files that went into the build, with their metadata.

Build SBOM: sbom-build.spdx.json

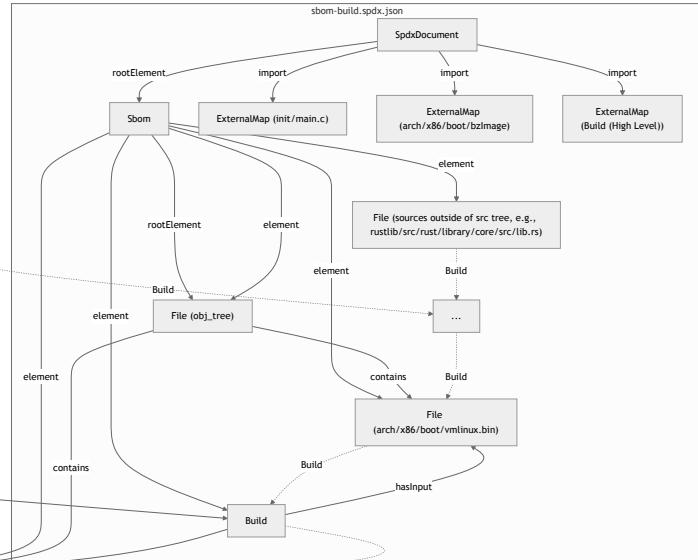
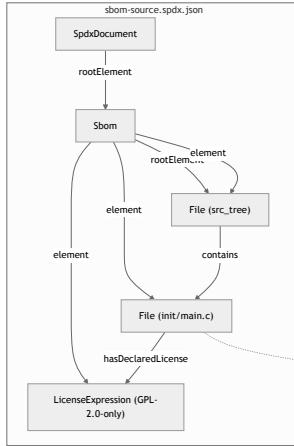
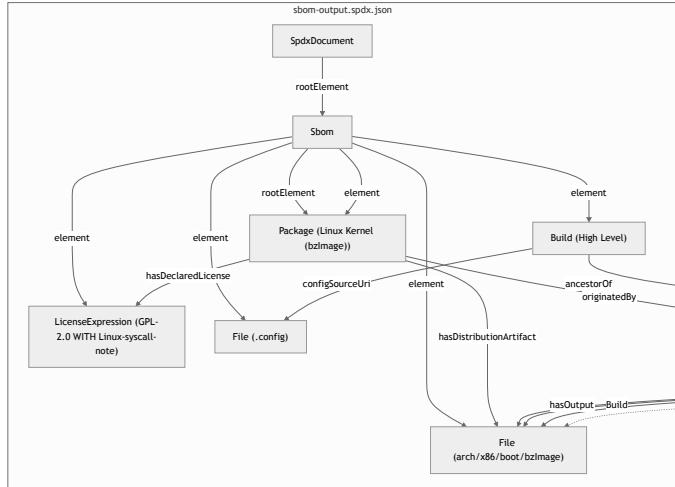
Links the sources to the output artifacts and represents the build graph.



This only works with out-of-tree builds right now

The heuristic to identify which files are source is whether they are in the source directory.

Internal structure:



Source SBOM

Basically a list of the used files together with more metadata that is collected via static analysis

- extract declared license from [SPDX-License-Identifier:](#)
- guess file-type based on file extension and location
- compute file hashes

File

Summary
Description
Metadata
Class hierarchy
Properties
External properties cardinality updates
All properties

Package
Sbom
Snippet
SoftwareArtifact

Properties
Vocabularies
Security
Licensing
SimpleLicensing
ExpandedLicensing
Dataset
AI
Build
Lite
Extension

ANNEXES

A. RDF model definition and diagrams
B. SPDX license expressions
C. SPDX License List matching guidelines
D. SPDX Lite
E. Package URL specification

/ model / Software

/ Classes / File

File

Summary

Refers to any object that stores content on a computer.

Description

Refers to any object that stores content on a computer. The type of content can optionally be provided in the contentType property.

The fileKind property can be set to [directory](#) to indicate the file represents a directory and all content stored in that directory.

Metadata

<https://spdx.org/rdf/3.0.1/terms/Software/File>

Name	File
Instantiability	Concrete
SubclassOf	/Software/SoftwareArtifact

Class hierarchy

Source SBOM

Example entry for `init/main.c`:

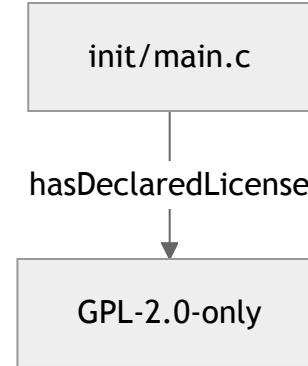
```
{
  "type": "software_File",
  "spdxId": "s:121",
  "creationInfo": "_:creationinfo",
  "name": "init/main.c",
  "verifiedUsing": [
    {
      "type": "Hash",
      "hashValue": "0c8bdff5df4e99c50cf7a7b95c1798206a39d93edd97bd605e7cc3ccfdd58ec9",
      "algorithm": "sha256"
    }
  ],
  "software_primaryPurpose": "source",
  "software_contentIdentifier": [
    {
      "type": "software_ContentIdentifier",
      "software_contentIdentifierType": "gitoid",
      "software_contentIdentifierValue": "fab4f599c035e73483eaeae101adaff4f5d72f2b"
    }
  ]
}
```

Source SBOM

The file from the previous slide ("s:121") has the declared license GPL-2.0-only, which is encoded via a Relationship to a LicenseExpression:

```
{  
  "type": "Relationship",  
  "spdxId": "s:4415",  
  "creationInfo": "_:creationinfo",  
  "relationshipType": "hasDeclaredLicense",  
  "from": "s:121",  
  "to": [  
    "s:4301"  
  ]  
}
```

```
{  
  "type": "simplelicensing_LicenseExpression",  
  "spdxId": "s:4301",  
  "creationInfo": "_:creationinfo",  
  "simplelicensing_licenseExpression": "GPL-2.0-only"  
}
```



Output SBOM

```
{  
  "type": "software_Package",  
  "spdxId": "o:2",  
  "creationInfo": "_:creationinfo",  
  "name": "Linux Kernel (bzImage)",  
  "comment": "Architecture=x86",  
  "originatedBy": [ "p:0" ],  
  "software_primaryPurpose": "application",  
  "software_copyrightText": "...",  
  "software_packageVersion": "6.18.0"  
}
```

```
{  
  "type": "software_Package",  
  "spdxId": "o:3",  
  "creationInfo": "_:creationinfo",  
  "name": "efivarfs.ko",  
  "comment": "Architecture=x86",  
  "originatedBy": [ "p:0" ],  
  "software_primaryPurpose": "module",  
  "software_copyrightText": "...",  
  "software_packageVersion": "6.18.0"  
}
```

ContentDefinition

File

Package

- Summary
- Description
- Metadata
- Class hierarchy
- Properties
- External properties cardinality updates
- All properties
- Sbom
- Snippet
- SoftwareArtifact

Properties

Vocabularies

Security

Licensing

SimpleLicensing

ExpandedLicensing

Dataset

AI

Build

Lite

Extension

ANNEXES

- A. RDF model definition and diagrams
- B. SPDX license expressions
- C. SPDX License List matching guidelines
- D. SPDX Lite
- E. Package URL specification

LICENSES

/ model / Software

/ Classes / Package

Package

Summary

Refers to any unit of content that can be associated with a distribution of software.

Description

A package refers to any unit of content that can be associated with a distribution of software.

Typically, a package is composed of one or more files.

Any of the following non-limiting examples may be (but are not required to be) represented in SPDX as a package:

- a tarball, zip file or other archive
- a directory or sub-directory
- a separately distributed piece of software which another Package or File uses or depends upon (e.g., a Python package, a Go module, ...)
- a container image, and/or each image layer within a container image
- a collection of one or more sub-packages
- a Git repository snapshot from a particular point in time

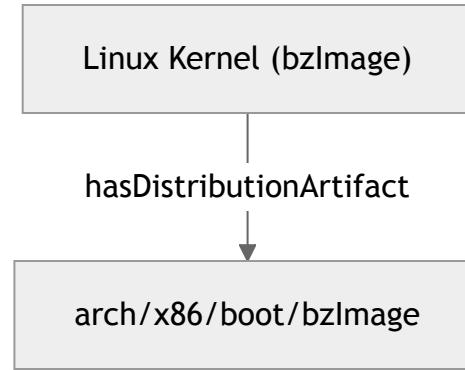
Note that some of these could be represented in SPDX as a file as well.

Output SBOM

The Package is linked to a File as its DistributionArtifact via a Relationship.

```
{  
  "type": "Relationship",  
  "spdxId": "o:12",  
  "creationInfo": "_:creationinfo",  
  "relationshipType": "hasDistributionArtifact",  
  "from": "o:2",  
  "to": [  
    "b:7"  
  ]  
}
```

```
{  
  "type": "software_File",  
  "spdxId": "b:7",  
  "creationInfo": "_:creationinfo",  
  "name": "arch/x86/boot/bzImage",  
  [...]  
}
```



Output SBOM

A Build element contains the high-level view of the build. The Build is linked to the outputs with the `hasOutput` Relationship.

```
{
  "type": "build_Build",
  "spdxId": "o:3",
  "creationInfo": "_:creationinfo",
  "build_buildType": "urn:spdx.dev:Kbuild",
  "build_buildId": "o:3",
  "build_environment": [
    { "type": "DictionaryEntry", "key": "SRCARCH", "value": "x86" },
    { "type": "DictionaryEntry", "key": "srcroot", "value": ".." },
    [ ... ]
  ],
  "build_configSourceUri": [ "o:2" ],
  "build_configSourceDigest": [
    {
      "type": "Hash",
      "hashValue": "837d6caddc50002189ef7dcac28b79a2165166310c7b8a85278534a76dc8abce",
      "algorithm": "sha256"
    }
  ]
}
```

Build SBOM

The Build SBOM contains many low-level Build elements, describing the detailed structure of the Build.

- Every build stores the `savedcmd_content` as comment
- The high level build from the output SPDX is connected to the low level builds via the `ancestorOf` Relationship

This ties the Source SBOM together with the Output SBOM.

4. Terms and definitions
5. Conformance
6. Model and serializations

MODEL

- ⊕ Core
- ⊕ Software
- ⊕ Security
- ⊕ Licensing
- ⊕ SimpleLicensing
- ⊕ ExpandedLicensing
- ⊕ Dataset
- ⊕ AI

Build

Description

- ⊕ Classes
- ⊕ Properties

Lite

Extension

ANNEXES

- A. RDF model definition and diagrams
- B. SPDX license expressions
- C. SPDX License List matching guidelines
- D. SPDX Lite
- E. Package URL specification

LICENSES

- Community Specification License 1.0
Creative Commons Attribution License 3.0 Unported

Build

Profile information

Summary

The Build Profile defines the set of information required to describe an instance of a Software Build.

Description

A Software Build is defined here as the act of converting software inputs into software artifacts using software build tools. Inputs can include source code, config files, artifacts that are build environments, and build tools. Outputs can include intermediate artifacts to other build inputs or the final artifacts.

The Build profile provides a subclass of Element called Build.

It also provides a minimum set of required Relationship Types from the Core profile:

- `hasInput`: Describes the relationship from the Build element to its inputs.
- `hasOutput`: Describes the relationship from the Build element to its outputs.

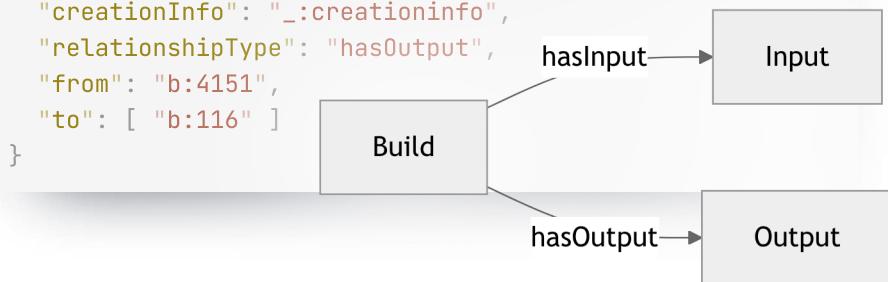
Build SBOM

A Build is linked via Relationships to its inputs and outputs.

```
{  
  "type": "build_Build",  
  "spdxId": "b:4151",  
  "creationInfo": "_:creationinfo",  
  "comment": "arch/x86/boot/compressed/mkpiggy arch/x86/boot/compressed/vmlinux.bin.gz > arch/x86/boot/compressed/piggy",  
  "build_buildType": "urn:spdx.dev:Kbuild",  
  "build_buildId": "urn:spdx.dev:987b9938-c26a-490d-ae00-54630d81262c/output/3"  
}
```

```
{  
  "type": "Relationship",  
  "spdxId": "b:4152",  
  "creationInfo": "_:creationinfo",  
  "relationshipType": "hasInput",  
  "from": "b:4151",  
  "to": [ "b:140" ]  
}
```

```
{  
  "type": "Relationship",  
  "spdxId": "b:4153",  
  "creationInfo": "_:creationinfo",  
  "relationshipType": "hasOutput",  
  "from": "b:4151",  
  "to": [ "b:116" ]  
}
```



Next steps

Get it upstream

The idea is

- to contribute the subdirectory `/sbom` from the KernelSbom project to `/tools/sbom`
- add a config option `SBOM` to enable it
- produce the SPDX files in output directory

Currently in the contribution process:

Get the branch `kernelsbom-integration` from <https://github.com/augelu-tng/linux.git>.

```
$ make defconfig O=kernel_build  
$ make O=kernel_build -j$(nproc)  
$ make O=kernel_build SPDX
```

Messages in this thread

- First message in thread
- Luis Augenstein
 - Luis Augenstein

From Luis Augenstein <>
Subject [PATCH v2 00/14] Add SPDX SBOM generation tool
Date Tue, 20 Jan 2026 12:53:38 +0100

This patch series introduces a Python-based tool for generating SBOM documents in the SPDX 3.0.1 format for kernel builds.

A Software Bill of Materials (SBOM) describes the individual components of a software product. For the kernel, the goal is to describe the distributable build outputs (typically the kernel image and modules), the source files involved in producing these outputs, and the build process that connects the source and output files.

To achieve this, the SBOM tool generates three SPDX documents:

- `sbom-output.spdx.json`
Describes the final build outputs together with high-level build metadata.
- `sbom-source.spdx.json`
Describes all source files involved in the build, including licensing information and additional file metadata.
- `sbom-build.spdx.json`
Describes the entire build process, linking source files from the source SBOM to output files in the output SBOM.

The sbom tool is optional and runs only when CONFIG_SBOM is enabled. It is invoked after the build, once all output artifacts have been generated. Starting from the kernel image and modules as root nodes, the tool reconstructs the dependency graph up to the original source files. Build dependencies are primarily derived from the cmd files

Further development and next steps

- Let it have contact with the real world. We want feedback.
- Support for more architectures, like RISC-V, PowerPC, and s390.
- Integration with other tools and ecosystems.

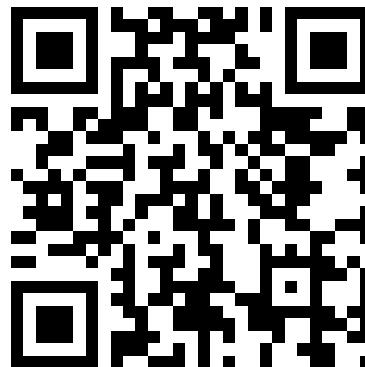
Thank You!

maximilian.huber@tngtech.com

GitHub: [@maxhbr](https://github.com/maxhbr)

[linkedin.com/in/maxhbr](https://www.linkedin.com/in/maxhbr)

KernelSbom



<https://github.com/TNG/KernelSbom>