Surfactant

Automated SBOMs from file systems

Ryan Mast

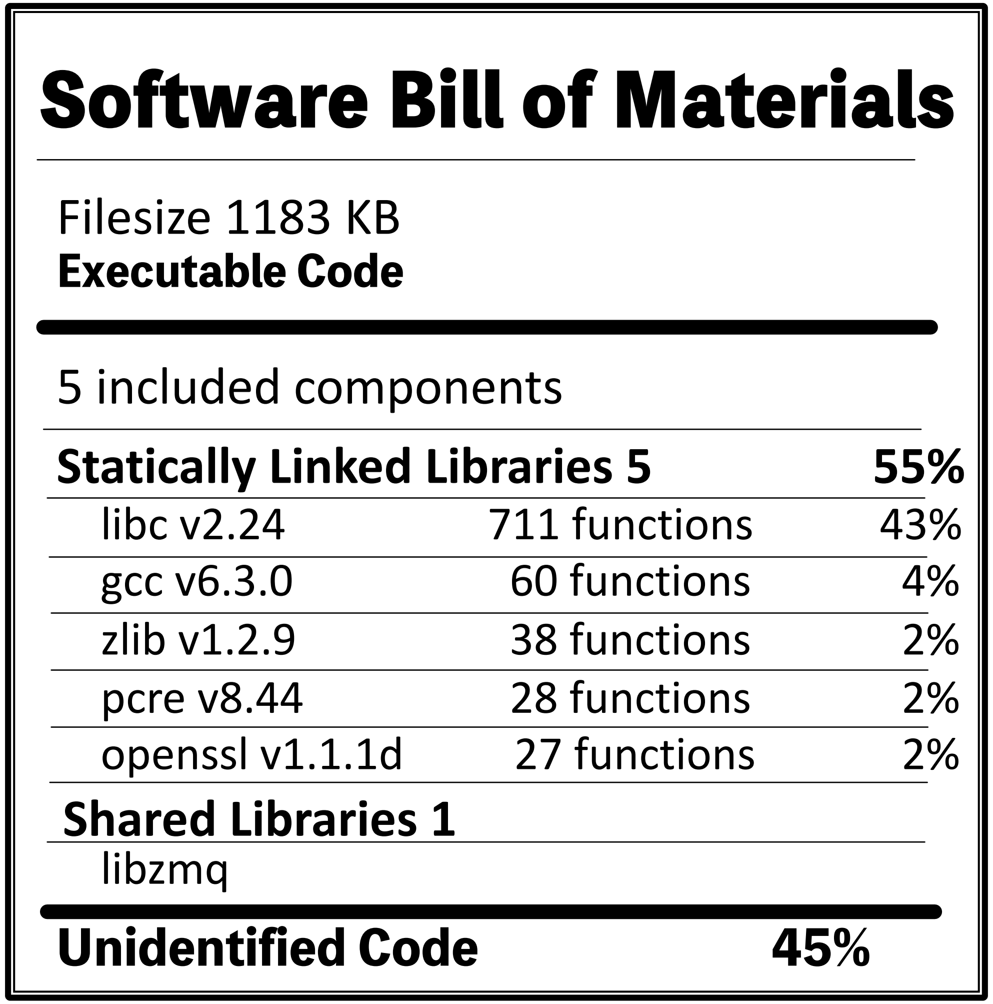
March 20, 2024

LLNL-PRES-859637

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

What is an SBOM?

Nutrition Facts Label for Software



Software Bill of Materials

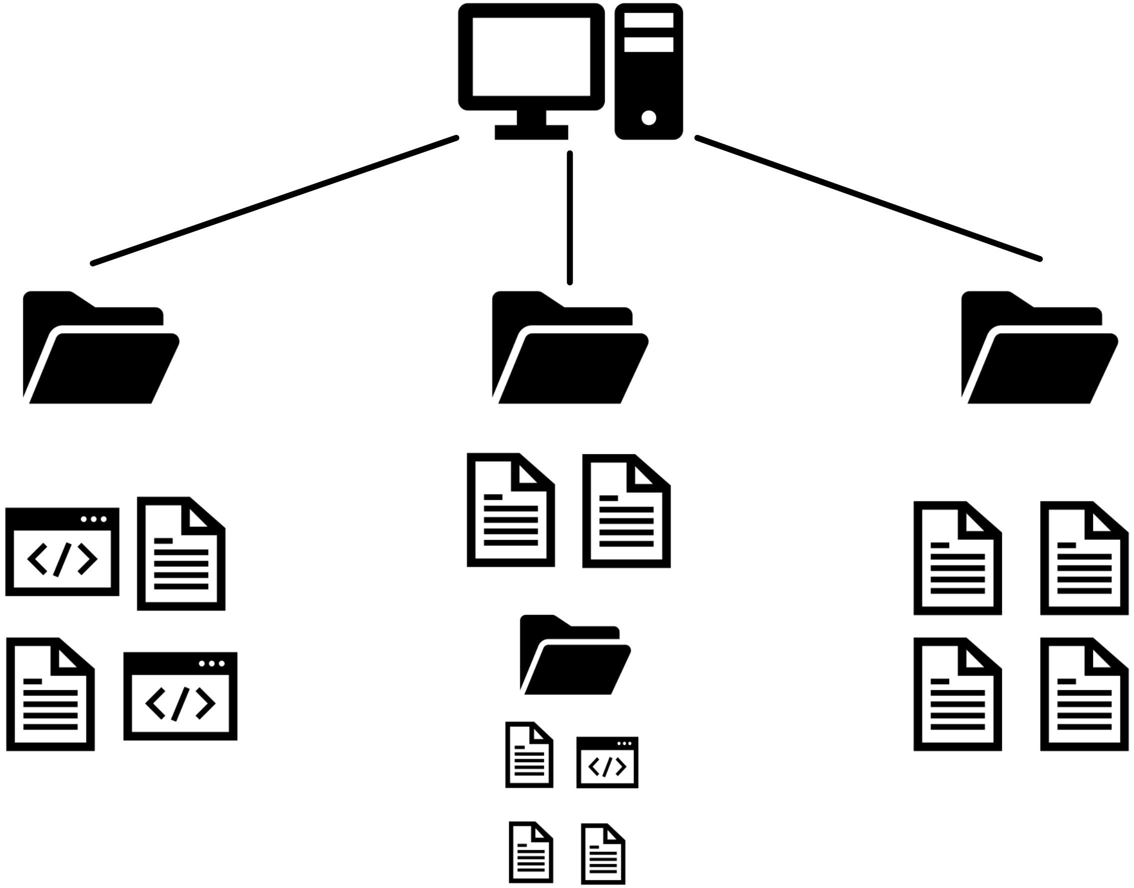
Filesize 1183 KB
Executable Code

5 included components

Statically Linked Libraries 5		**55%**
libc v2.24	711 functions	43%
gcc v6.3.0	60 functions	4%
zlib v1.2.9	38 functions	2%
pcre v8.44	28 functions	2%
openssl v1.1.1d	27 functions	2%
Shared Libraries 1		
libzmq		
Unidentified Code		**45%**

What is an SBOM?

File-level vs Package-level



OpenSSL
Cryptography and SSL/TLS Toolkit
python™
Linux™
NGINX
zlib
APACHE
HTTP SERVER PROJECT
node
JS®
Java™
RapidJSON

The Need

SBOMs from software provided by vendors

- Primarily compiled binaries
  - No source code (often firmware)
  - No BOMs from vendors
  - Custom file formats may be used
- Often a large number of files
  - Received as a compressed archive, filesystem image, or Windows installer
  - Windows configuration/support software (mix of native and .NET/CLR binaries)
  - Embedded Linux device file systems
  - No package manager metadata files
- SBOMs need to have accurate information
  - Relationships between files
  - Support future analysis
    - Do new CVEs apply?

Limitations of Existing Tools

- Depend on source code
- Do not to establish links between files
  - No relationships showing what is loading various shared libraries
  - Unable to capture accurate install paths
- Fail to identify software packages
- Difficult to add support for new file formats
  - Often depend on asking the developer to add support; provide sample files
- Do not support custom SBOM output formats

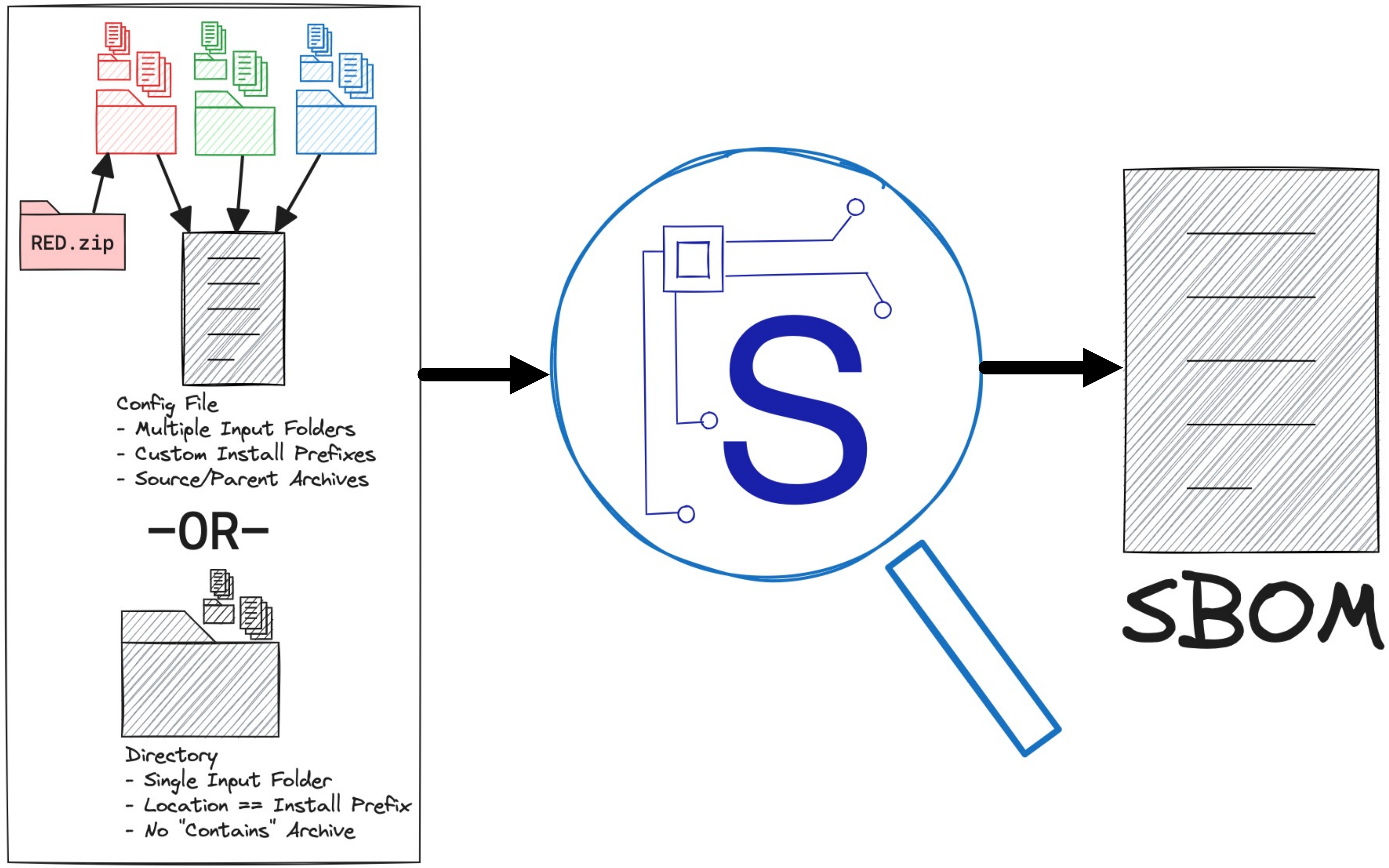
Inspiration

Binary file formats contain a lot of metadata

- Shared libraries to load
  - ELF, PE, and Mach-O
  - .NET/CLR
- Product name, vendor, and version information
  - Windows PE, .NET/CLR, and MSI installers are particularly good
- Embedded dependency lists for auditing
  - Go, Rust, etc
- Humans > AI for recognizing static linked libraries and overall package (for now)
  - Generate an initial automated SBOM, supplement with manual analysis

Our solution: Surfactant

High-level overview



- Directory structures in
  - Gather metadata from files
  - Relationships from metadata
- SBOM out
  - Parent containers
  - Shared libraries used

Our solution: Surfactant

- Open Source! Available at https://github.com/LLNL/Surfactant
  - pip install surfactant
- Used to generate initial automated SBOMs
  - Supplemented with manual analysis
- Modular framework for SBOM generation
  - Recognize new file types
  - Extract interesting metadata for analysis
  - Perform additional analysis on individual files
  - Create additional relationships based on gathered metadata
  - Output SBOM in a variety of formats
    - CyTRICS, CSV, SPDX, CycloneDX, or custom
  - Load SBOM data
    - CyTRICS SBOM or custom formats (SPDX and CycloneDX in progress)

Our solution: Surfactant

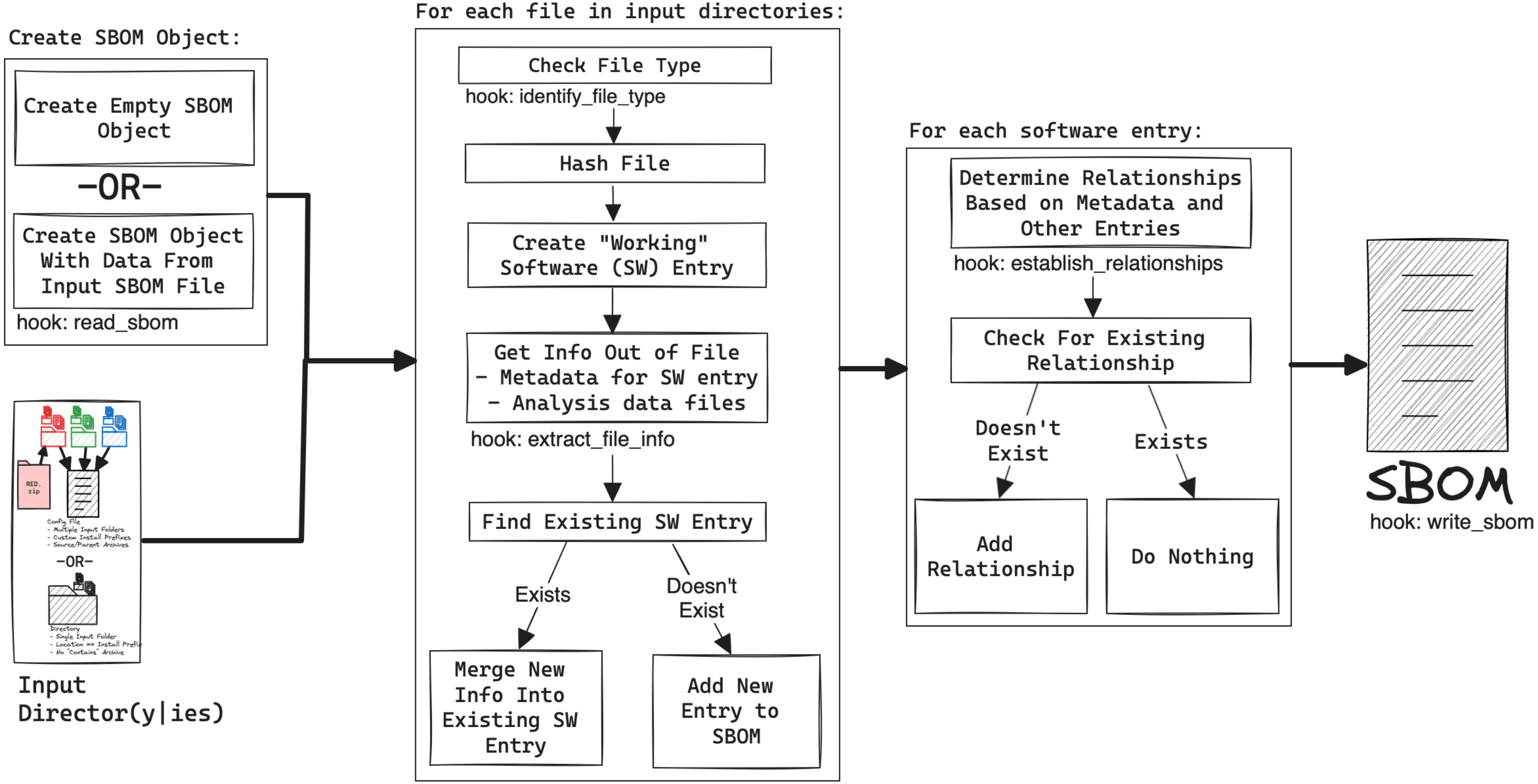
Future work

- Support additional file formats
  - Docker containers
  - Scripts (Python, JavaScript, Shell)
- New analysis passes leveraging static analysis tools
- Enable configuration options for plugins
- UX improvements
  - CLI for SBOM manipulation
  - GUI to reduce command line knowledge required
- Explore ML techniques that could be used to improve output
  - Identify overall package names (e.g. binary is part of git)
  - Identify statically linked libraries

Lawrence Livermore
National Laboratory

Our solution: Surfactant

Inner workings



Create SBOM Object:

Create Empty SBOM Object

-OR-

Create SBOM Object With Data From Input SBOM File

hook: read_sbom

Input Director(y|ies)

For each file in input directories:

Check File Type

hook: identify_file_type

Hash File

Create "Working" Software (SW) Entry

Get Info Out of File
- Metadata for SW entry
- Analysis data files

hook: extract_file_info

Find Existing SW Entry

Exists

Merge New Info Into Existing SW Entry

Doesn't Exist

Add New Entry to SBOM

For each software entry:

Determine Relationships Based on Metadata and Other Entries

hook: establish_relationships

Check For Existing Relationship

Doesn't Exist

Add Relationship

Exists

Do Nothing

SBOM

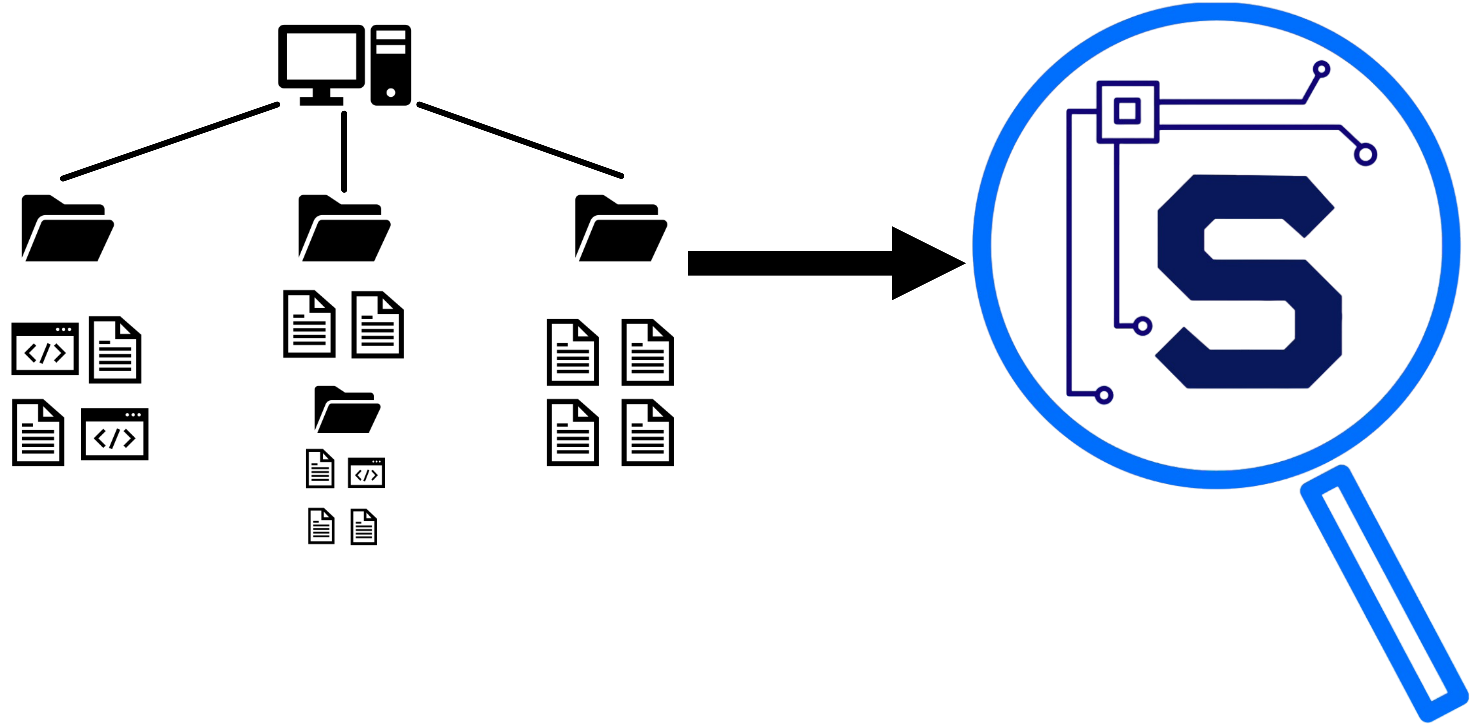
hook: write_sbom

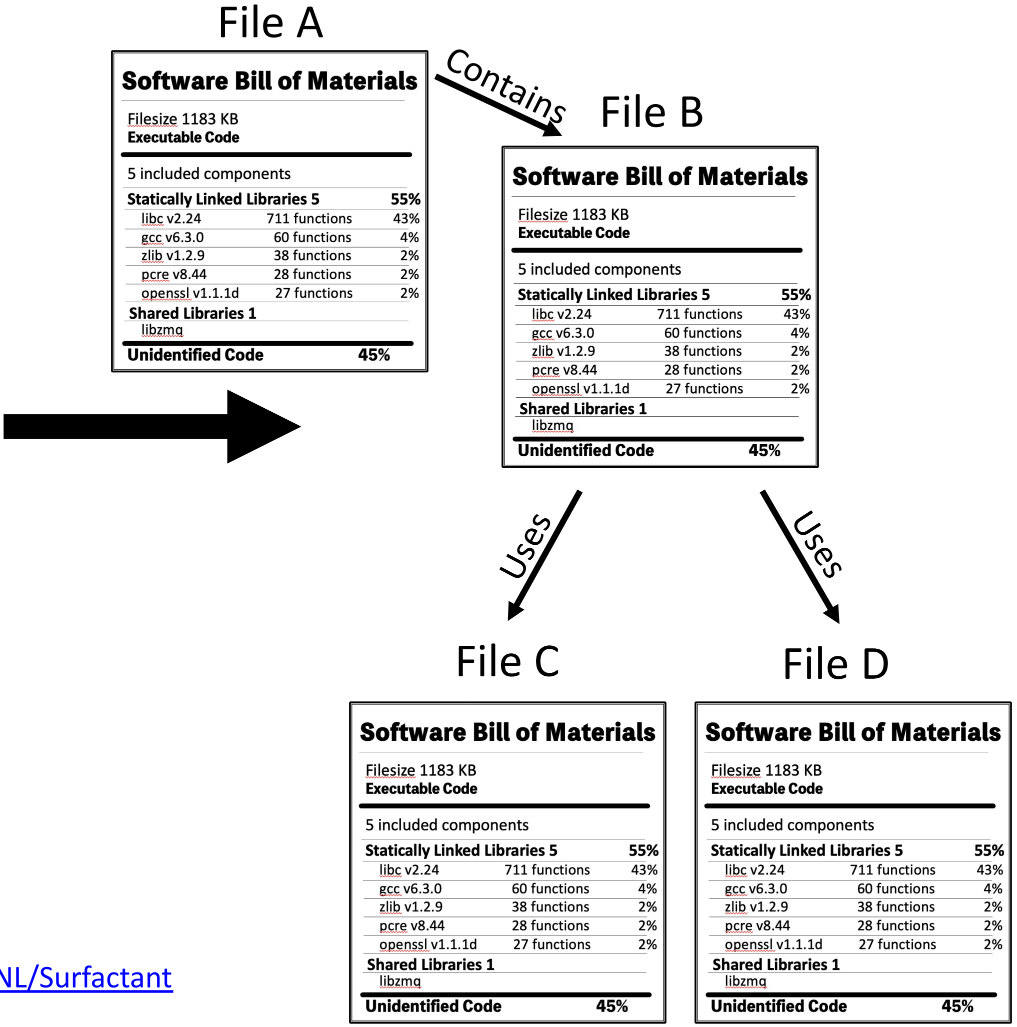
What is an SBOM?

SBOM Formats

Generating an SBOM

Automating file-level SBOM Generation Using Surfactant





File A

Software Bill of Materials

Filesize 1183 KB
Executable Code

5 included components		
Statically Linked Libraries 5		**55%**
libc v2.24	711 functions	43%
gcc v6.3.0	60 functions	4%
zlib v1.2.9	38 functions	2%
pcre v8.44	28 functions	2%
openssl v1.1.1d	27 functions	2%
Shared Libraries 1		
libzmq		
Unidentified Code		**45%**

File B

Software Bill of Materials

Filesize 1183 KB
Executable Code

5 included components		
Statically Linked Libraries 5		**55%**
libc v2.24	711 functions	43%
gcc v6.3.0	60 functions	4%
zlib v1.2.9	38 functions	2%
pcre v8.44	28 functions	2%
openssl v1.1.1d	27 functions	2%
Shared Libraries 1		
libzmq		
Unidentified Code		**45%**

File C

Software Bill of Materials

Filesize 1183 KB
Executable Code

5 included components		
Statically Linked Libraries 5		**55%**
libc v2.24	711 functions	43%
gcc v6.3.0	60 functions	4%
zlib v1.2.9	38 functions	2%
pcre v8.44	28 functions	2%
openssl v1.1.1d	27 functions	2%
Shared Libraries 1		
libzmq		
Unidentified Code		**45%**

File D

Software Bill of Materials

Filesize 1183 KB
Executable Code

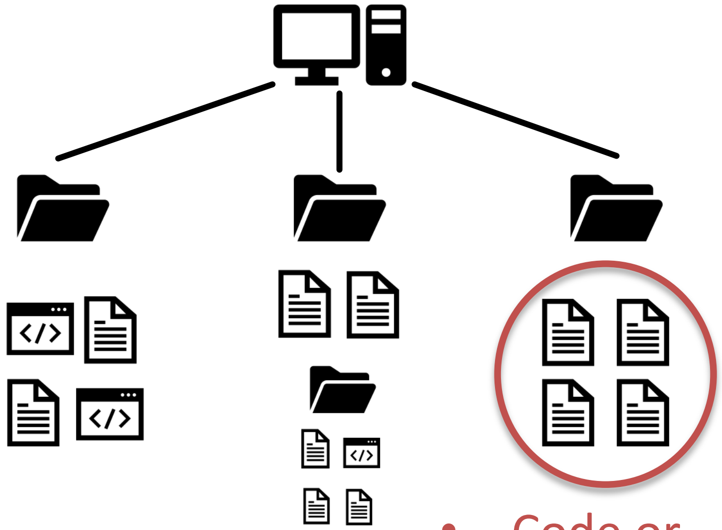
5 included components		
Statically Linked Libraries 5		**55%**
libc v2.24	711 functions	43%
gcc v6.3.0	60 functions	4%
zlib v1.2.9	38 functions	2%
pcre v8.44	28 functions	2%
openssl v1.1.1d	27 functions	2%
Shared Libraries 1		
libzmq		
Unidentified Code		**45%**

Open Source, available at: https://github.com/LLNL/Surfactant
Or: pip install surfactant

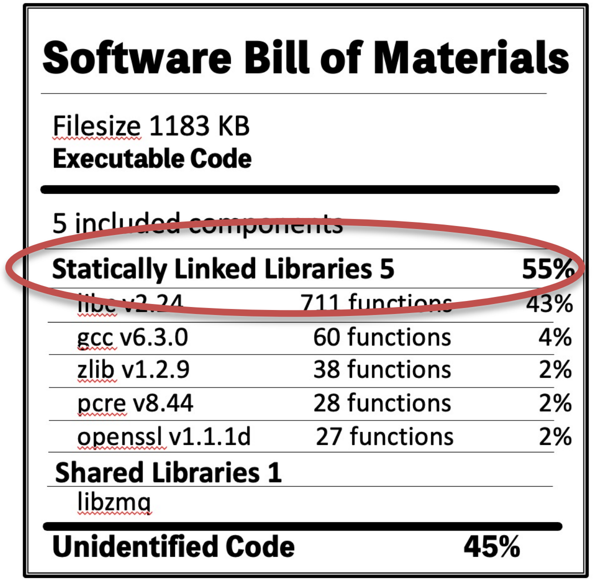
Generating an SBOM

Challenges

- Legality of analyzing binaries made by others?



- Code or Data?
- (Nested) Archives?



Software Bill of Materials

Filesize 1183 KB
Executable Code

5 included components

Statically Linked Libraries 5		**55%**
libc v2.24	711 functions	43%
gcc v6.3.0	60 functions	4%
zlib v1.2.9	38 functions	2%
pcre v8.44	28 functions	2%
openssl v1.1.1d	27 functions	2%

Shared Libraries 1
libzmq

Unidentified Code **45%**

- Recognizing statically-linked or header-only libraries
- Limited (or no) metadata giving name, vendor, version info
- Determining dynamic run-time relationships

OpenSSL
Cryptography and SSL/TLS Toolkit
python™
Linux
NGINX
zlib
APACHE
HTTP SERVER PROJECT
node
JS
Java
RapidJSON

- Determining higher-level packages
- Tying packages to specific files

March 20, 2024

Hannah Pearson-Kleinheider
Idaho National Laboratory

(with many thanks to Robert Erbes for 80% of the slides)

SBOM Lessons Learned

5 years of generating and using BOMs for CyTRICS

Battelle Energy Alliance manages INL for the
U.S. Department of Energy's Office of Nuclear Energy

CyTRICS™

Cyber Testing for Resilient Industrial Control Systems

INL Idaho National Laboratory

Lawrence Livermore National Laboratory

Pacific Northwest NATIONAL LABORATORY

OAK RIDGE National Laboratory

Sandia National Laboratories

NREL NATIONAL RENEWABLE ENERGY LABORATORY

cytrics.inl.gov

Generating BOMs

- There are many ways to make a BOM
  - When and how it is compiled (downloads, updates, versions)
  - How it is formatted and organized
  - The details
- Function guides form
  - CyTRICS focuses on supply chain illumination and vulnerability correlation
  - What use case was any given BOM built for?
- The atomic unit of ‘software’ is squishy
  - How deep do you go?
  - Binary vs. Webapp vs. Mobile vs. Script

BOM Tools

- THE DREAM: Automated BOM generation
- THE REALITY: Functionality isn't fully there yet
  - Better suited for some use cases than others
  - Does not have to be perfect to be useful
- Know what the tools you are using can and can not do
  - Metrics and test cases for tool evaluation
- Custom tooling
  - Based on thorough understanding of objectives
  - Consider generation, storage, querying, versioning, sharing, etc.

Using BOMs

- Not all BOMs are created equal
  - The person (or program / company) behind the BOM
  - Vendors rarely have a complete picture of the contents of their products
  - Vendors frequently do not consider their own software in shared BOMs
- For vulnerability risk management and response, BOMs alone are not enough
  - BOMs are the beginning, not the end
  - Still requires understanding the context of your system and environment
- False negatives / positives
  - Simplistic matching of identified software to CVEs is dangerous

Using BOMs Summary

- Generating BOMs
  - Different use cases == Different BOMs
- Using BOMs is Hard
  - Accuracy
  - Completeness
  - Relevance
- Usefulness vs. Compliance

INL

Idaho National Laboratory

Battelle Energy Alliance manages INL for the U.S. Department of Energy's Office of Nuclear Energy. INL is the nation's center for nuclear energy research and development, and also performs research in each of DOE's strategic goal areas: energy, national security, science and the environment.