

New and established tools for software scanning:

Overview of available compliance tools

Caren Kresse

Open Source Automation Development Lab (OSADL) eG

Compliance toolchain: Requirements

External Input:
What information is provided by suppliers?

Contribution:
Compliance information is contributed to public projects (e.g. ClearlyDefined)

Analyzing:
What software is used and how?

Scanning:
Which licenses?
Extract information.

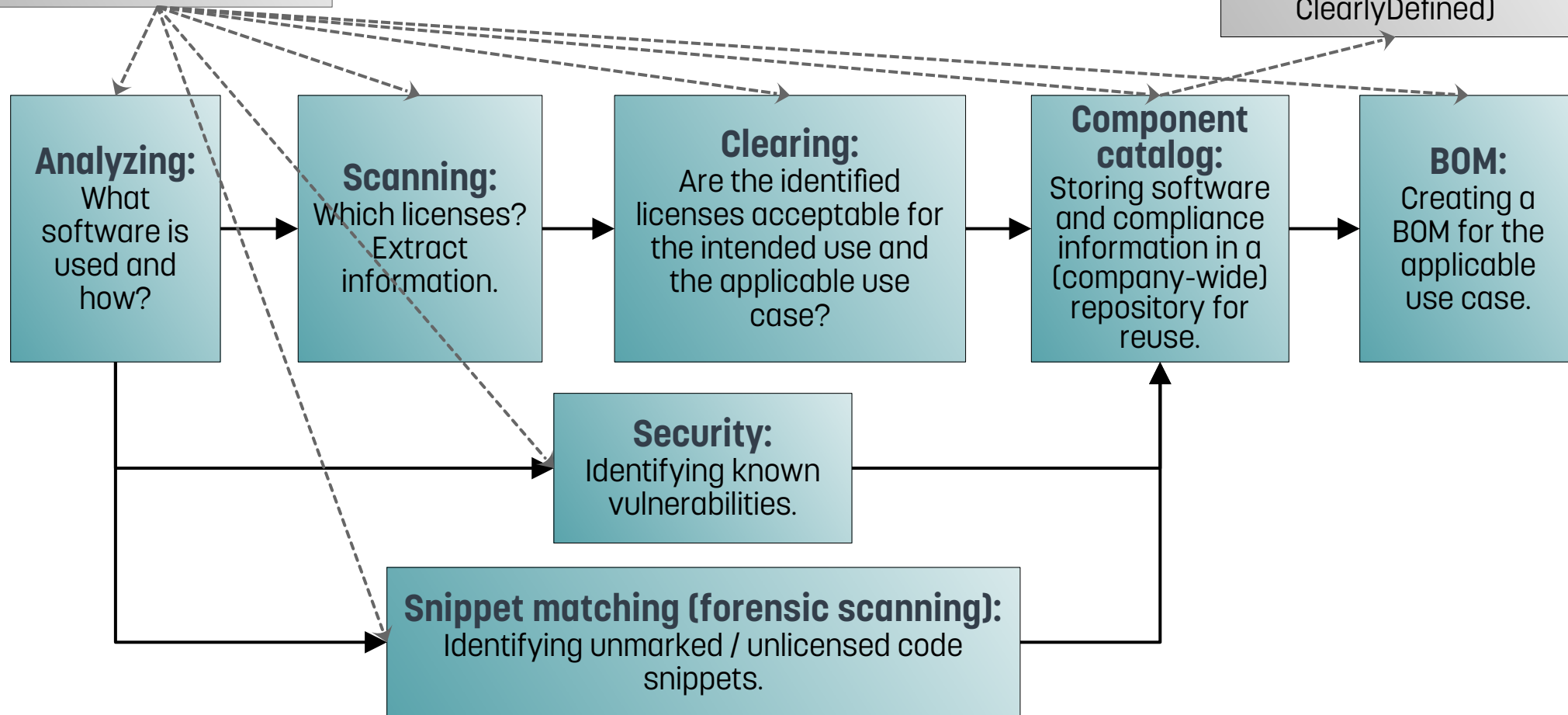
Clearing:
Are the identified licenses acceptable for the intended use and the applicable use case?

Component catalog:
Storing software and compliance information in a (company-wide) repository for reuse.

BOM:
Creating a BOM for the applicable use case.

Security:
Identifying known vulnerabilities.

Snippet matching (forensic scanning):
Identifying unmarked / unlicensed code snippets.



Alphabetical listing of tool selection

- AboutCode
 - AboutCode Toolkit
 - DeltaCode (not maintained)
 - Scancode
 - Scancode Workbench
 - ScanCode.io
 - TraceCode Toolkit
 - VulnerableCode
- BANG
- Barista
- Blackduck Protex / Blackduck Hub
- Callgraph
- CLA Assistant
- ClearlyDefined
- CVE hound (only Linux kernel)
- DeltaScan
- FOSSID / snyk
- FOSSLight
- FOSSology
- License Compatibility Checker
- Licensee.js
- nex/B Container inspector
- nex/B: neues Matching tool
- Ninka (not maintained)
- Opossum-Tool
- OSS Discovery by OpenLogic (no longer maintained)
- OSS Review Toolkit
- Revenera / Flexera
- Pivotal / LicenseFinder
- Quartermaster (no longer active)
- Reuse
- ScanOSS
- SPDX Tools
- SW360
- SW360antenna (no longer active)
- Tern
- vinland-technology/flict
- Whitesource

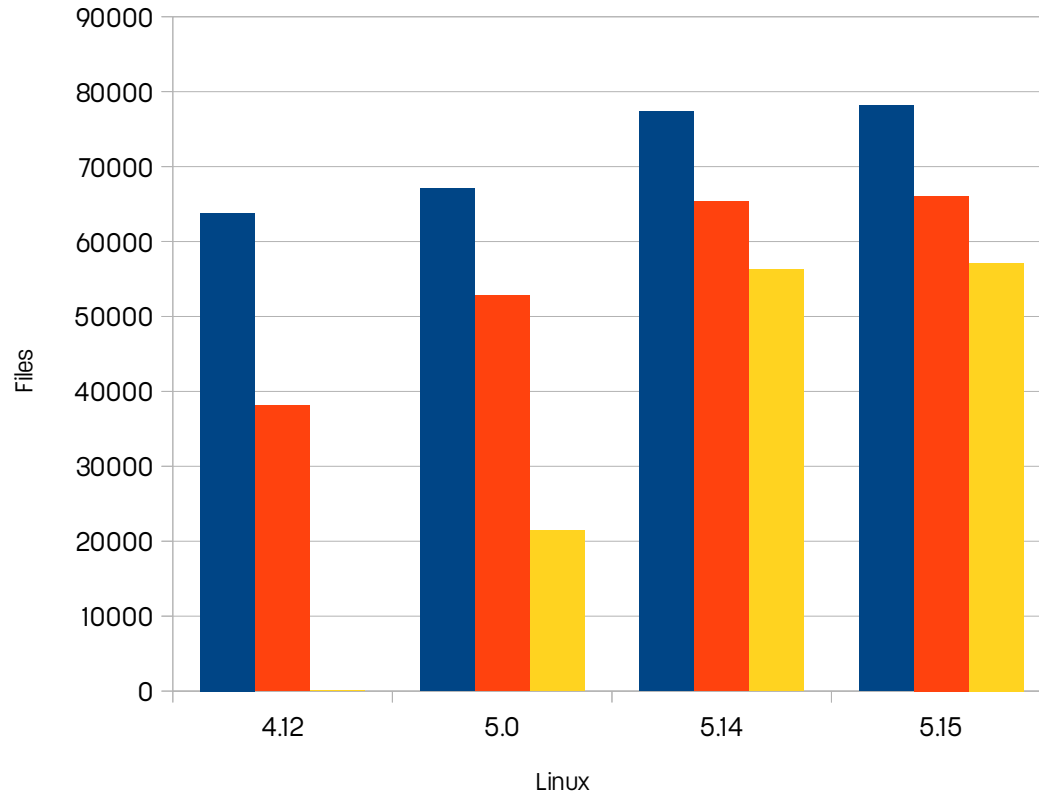
Tooling categories

- Analyzing
- Informational scanning
- Clearing
- Component catalog
- Software BOM, File formats
- Security
- Snippet matching (forensic scanning)

Analyzing

- Which packages?
- What additional / own software?
- Which dependencies?
- How are they integrated?
- Most information from package management system or build tools.
- Supplemented by manual information on additional components and architecture.

Digression: “Root of trust”



- The Linux Kernel (and some other projects) are already in “good shape” when it comes to licensing information.
- A company may decide to trust this information and only look at the delta (*e.g.* custom BSP changes).
- Root of trust is also relevant for sharing compliance information/material, *e.g.* via ClearlyDefined.

Deltascan

<https://github.com/armijnhemel/compliance-scripts/tree/master/osadl-audit/>

- Command line tool to identify such Linux kernel source code files that deviate from the “official” Linux kernel release files provided by kernel.org
- Optionally perform a license scan (with ScanCode and Nomos) on these files.
- Based on comparing hash codes of all files.
- Requires to create a database of kernel.org files (part of the provided scripts).

Deltascan: General information

- Assumption: Files from an official kernel release are licensed correctly ("Root of trust"), only modified or new files must be checked individually.
- Easy to install and run, but initial database creation takes a lot of time and disk space.
- **Input:** Linux kernel source code
- **Output:** Text

Deltascan: Example

```
$ python3 osadlaudit.py -s mytarball/linux-5.10.41-rt42 -c audit.config
```

```
SCANNING 56454 files
```

```
2 FILES NOT FOUND IN DATABASE
```

```
NOT FOUND mytarball/linux-5.10.41-rt42/arch/arm/boot/dts/am335x-wega-bw.dts
```

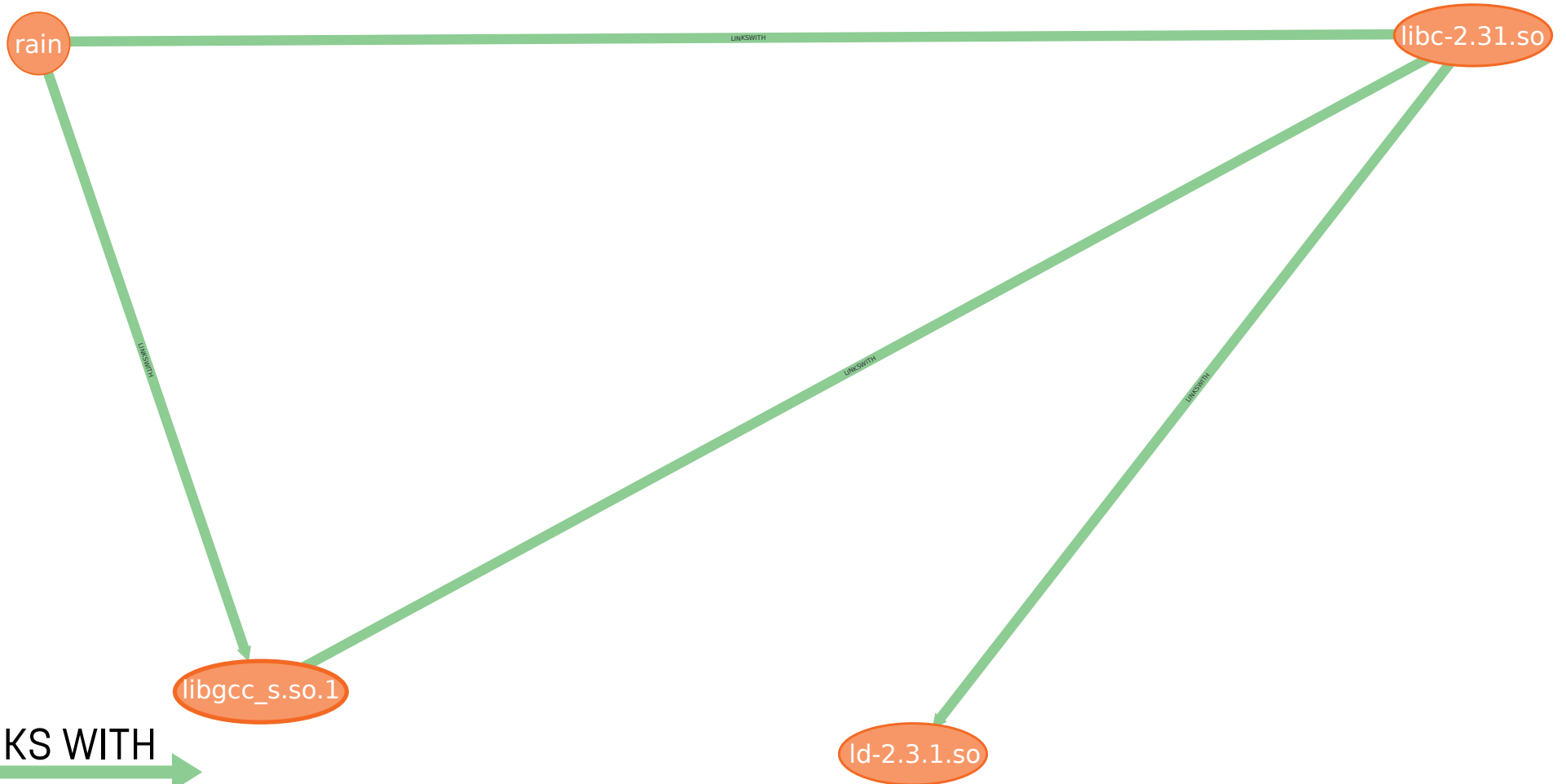
```
NOT FOUND mytarball/linux-5.10.41-rt42/drivers/misc/weather.c
```

Callgraph

<https://www.osadl.org/Callgraph>

- Command line tool creating linking graphs for ELF files to discover software components connected via function call (*i.e.* forming derivative works).
- Only works for files with ELF headers (not for interpreter languages).
- **Input:** Root filesystem and binary programs to be evaluated.
- **Output:** linking information in various formats (text, gv, cypher, gexf).

Callgraph example: Linking graph for "rain"



Callgraph example: Text

```
/opt/rain/bin/rain LINKSWITH /lib/libc-2.31.so  
/opt/rain/bin/rain LINKSWITH /lib/libgcc_s.so.1  
/lib/libgcc_s.so.1 LINKSWITH /lib/libc-2.31.so  
/lib/libc-2.31.so LINKSWITH /lib/ld-2.31.so
```

nexB container-inspector

<https://github.com/nexB/container-inspector>

- Command line tool to inspect Docker images, Dockerfiles, root filesystems, and virtual machine images.
- Extracts meta data and content of each layer to represent the runtime rootfs.
- Easy to install.
- **Input:** Docker images
- **Output:** JSON, CSV, rootfs content

container-inspector: Example

```
# ./venv/bin/container_inspector --help
```

```
Usage: container_inspector [OPTIONS] IMAGE_PATH
```

Find Docker images and their layers in IMAGE_PATH. Print information as JSON by default or as CSV with `--csv`. Optionally extract images with `extract-to`. Output is printed to stdout. Use a `>` redirect to save in a file.

Options:

`--extract-to PATH`

`--csv`

Print information as CSV instead of JSON.

container-inspector: Extracted layers

- Dockerfile

```
FROM osadl/ubuntu-docker-base-image:focal-amd64-211215-bin  
  
RUN apt-get update \  
    && apt-get upgrade --yes
```

- Running container-inspector on the container image created from this Dockerfile exports:
 - Layer 1: complete rootfs of the base image.
 - Layer 2: only files that have changed, in the same directory structure.

Informational scanning

ScanCode

- Standalone command line tool
<https://github.com/nexB/scancode-toolkit/>
- Part of the AboutCode project
<https://aboutcode.readthedocs.io/en/latest/aboutcode-project-overview.html>
- Simple and fast installation
- Easy integration into CI / CT environment
- **Input:** Source code
- **Output:** Extracted compliance information in many different file formats, *e.g.* JSON, HTML, SPDX, yaml, Debian copyright, CSV

ScanCode: HTML

path	start	end	what	value
busybox-1.34.1/LICENSE	1	6	license	gpl-2.0
busybox-1.34.1/LICENSE	9	348	license	gpl-2.0
busybox-1.34.1/LICENSE	12	12	copyright	Copyright (c) 1989, 1991 Free Software Foundation, Inc.
busybox-1.34.1/LICENSE	260	261	copyright	copyrighted by the Free Software Foundation
busybox-1.34.1/Makefile	1197	1197	license	gpl-2.0
busybox-1.34.1/Makefile	1211	1211	license	gpl-2.0
busybox-1.34.1/applets/applet_tables.c	6	6	copyright	Copyright (c) 2007 Denys Vlasenko <vda.linux@googlemail.com>
busybox-1.34.1/applets/applet_tables.c	8	8	license	gpl-2.0-plus
busybox-1.34.1/applets/applets.c	5	5	copyright	Copyright (c) 2007 Denys Vlasenko <vda.linux@googlemail.com>
busybox-1.34.1/applets/applets.c	7	7	license	gpl-2.0-plus
busybox-1.34.1/applets/individual.c	3	3	copyright	Copyright 2005 Rob Landley rob@landley.net
busybox-1.34.1/applets/individual.c	5	5	license	gpl-2.0-plus
busybox-1.34.1/applets/Kbuild.src	3	3	copyright	Copyright (c) 1999-2005 by Erik Andersen <andersen@codepoet.org>

ScanCode: JSON

```
{
  "path": "busybox-1.34.1/archival/ar.c",
  "type": "file",
  [...]
  "licenses": [
    {
      "key": "gpl-2.0-plus",
      "score": 100.0,
      "name": "GNU General Public License 2.0 or later",
      "short_name": "GPL 2.0 or later",
      "category": "Copyleft",
      "is_exception": false,
      [...]
      "spdx_license_key": "GPL-2.0-or-later",
      "spdx_url": "https://spdx.org/licenses/GPL-2.0-or-
later",
      "start_line": 9,
      "end_line": 9,
      [...]
      "matched_text": "* Licensed under GPLv2 or later,
see file LICENSE in this source tree."
    }
  ]
}
```

```
[...]
  "copyrights": [
    {
      "value": "Copyright (c) 2000 by Glenn McGrath",
      "start_line": 5,
      "end_line": 5
    },
    {
      "value": "Copyright (c) 2010 Nokia Corporation",
      "start_line": 12,
      "end_line": 12
    }
  ],
  [...]
  "authors": [
    {
      "value": "Alexander Shishkin",
      "start_line": 13,
      "end_line": 13
    }
  ],
  [...]
},
```

Clearing

FOSSology

<https://github.com/fossology/fossology/>

- Web-based multi-user tool for license scanning and clearing.
- Comes with integrated scanners Nomos, Monk and Ojo.
- Some effort to install and use but extensive functionality.
- See OSADL workshops (2019: <https://www.osadl.org/?id=3250>, 2021: <https://www.osadl.org/?id=3613>, Member login required)
- **Input:** Source code and expertise
- **Output:** Extracted compliance information as text, Debian Copyright file, SPDX

Opossum

<https://github.com/oopossum-tool/oopossumUI>

User guide:

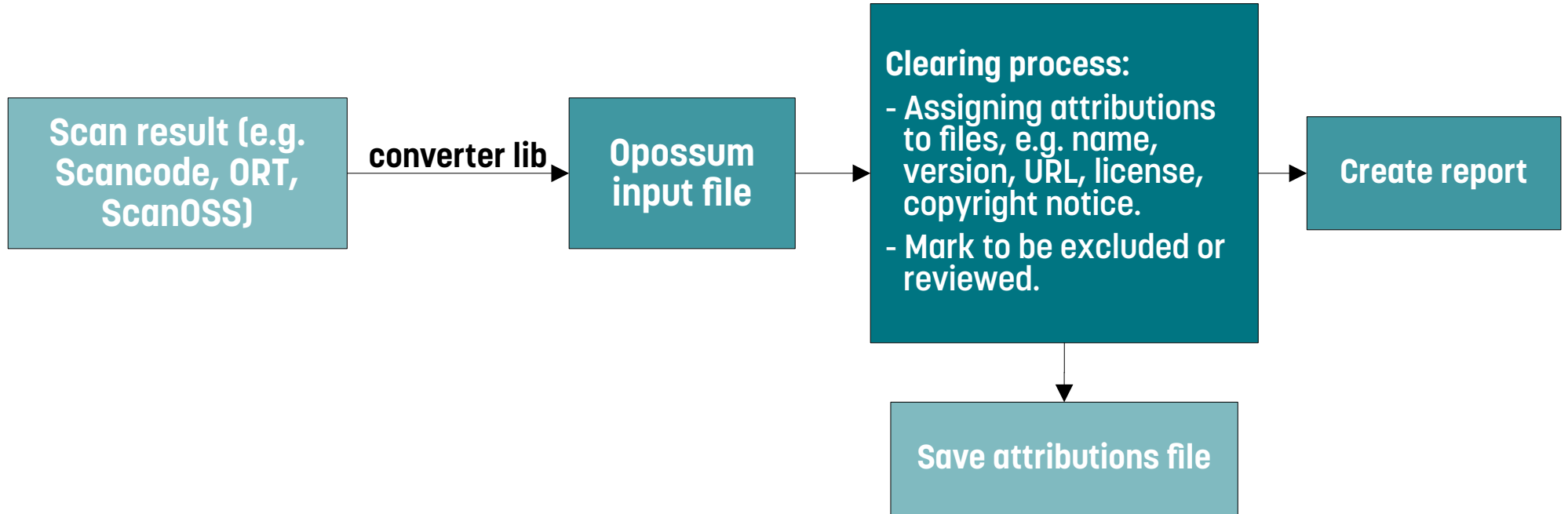
https://github.com/oopossum-tool/OopossumUI/blob/main/USER_GUIDE.md

- Graphical tool to manually visualize, review and edit compliance data created by external tools and to create a BOM.
- Lightweight app and uncluttered interface but workflow is not straightforward.
- Very small degree of automation.

Opossum: General information

- Standalone app, no installation required.
- **Input:** JSON, YAML (among others from ORT, ScanCode, SPDX-2.2, SCANOSS); must be converted with additional Opossum tool that is still in development and currently not documented very well (<https://github.com/opossum-tool/opossum.lib.hs>)
- **Output:** JSON, SPDX-2.2, CSV

Opossum: Workflow



Opossum: ScanCode JSON file imported

The screenshot displays the Opossum web interface with the following components:

- File Explorer (Left):** Shows a tree view of the file system. The selected path is `/busybox-1.35.0/applets/applet_tables.c`.
- Header:** Includes a menu bar (File, Edit, View, About) and a breadcrumb trail: `./././busybox-1.35.0-scancode.json`. On the right, there are tabs for `AUDIT`, `ATTRIBUTION`, `REPORT`, and a timestamp `OpossumUI-2021-12-23`.
- Main Content Area:** Displays the file path `/busybox-1.35.0/applets/applet_tables.c`. It has two tabs: `SIGNALS & CONTENT` (active) and `ALL ATTRIBUTIONS`. Under the active tab, there is a `Signals` section with a `Scancode` entry: `+ Copyright (c) 2007 Denys Viasenko <vda.linux@googlegmail.com> GPL-2.0-or-later`.
- Right-Hand Panel:** A form for editing attribution details:
 - `Name`: Input field.
 - `Version`: Input field.
 - `PURL`: Input field.
 - `URL`: Input field with a refresh icon.
 - `Copyright`: Text area.
 - `License Name`: Dropdown menu.
 - `Confidence`: A section with checkboxes for `1st Party`, `Follow-up`, and `Exclude From Notice`, followed by a `Confidence` dropdown set to `High (80)`.
 - `Comment`: Large text area.
 - `SAVE`: Button at the bottom right.

Opossum: Clearing process

The screenshot displays the Opossum web interface for editing a file's metadata. The browser window title is `././busybox-1.35.0-scancode.json`. The interface includes a menu bar (File, Edit, View, About) and navigation tabs (AUDIT, ATTRIBUTION, REPORT, OpossumUI-2021-12-23).

The left sidebar shows a file tree for `busybox-1.35.0`, with `applets` expanded to show `applet_tables.c`.

The main content area is titled `/busybox-1.35.0/applets/applet_tables.c` and features three tabs: **Attributions**, **SIGNALS & CONTENT**, and **ALL ATTRIBUTIONS**.

The **Attributions** tab shows the following information:

- Name:** Copyright (c) 2007 Denys Viasenko <vda.linux@googlemail.com>
GPL-2.0-or-later
- PURL:** (empty)
- URL:** (empty)
- License Name:** GPL-2.0-or-later
- Confidence:** High (80)

The **SIGNALS & CONTENT** tab shows the following information:

- Scancode:** Copyright (c) 2007 Denys Viasenko <vda.linux@googlemail.com>
GPL-2.0-or-later

The right-hand panel contains a **Comment** text area and a **SAVE** button.

Opossum: Report

../busybox-1.35.0-scancode.json

File Edit View About

AUDIT ATTRIBUTION REPORT OpossumUI-2021-12-23

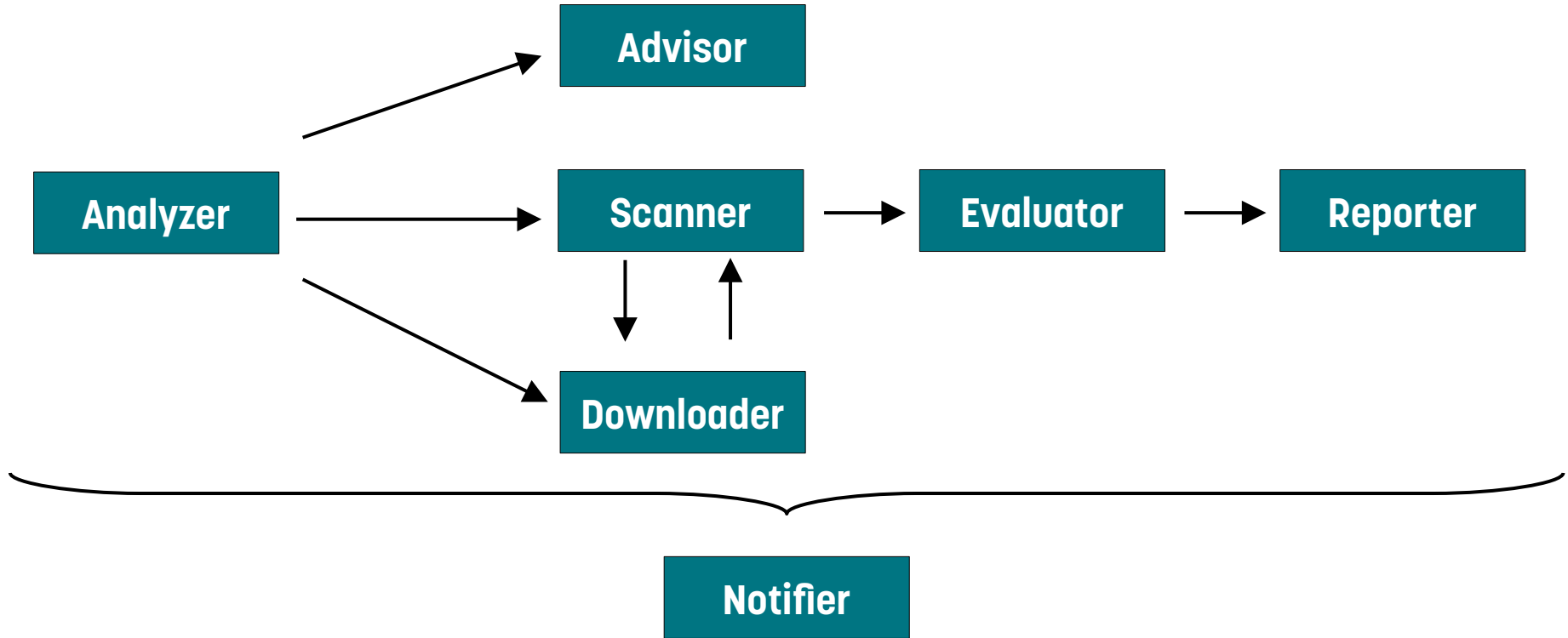
Filter

Name	Version	License	License Text	URL	Resources	Copyright	Confidence	Comment	Follow-up	Excluded	First Party
<input checked="" type="checkbox"/>		GPL-2.0-or-later			/busybox-1.35.0/applets/		80		No	No	No
<input checked="" type="checkbox"/>		GPL-2.0-or-later			/busybox-1.35.0/	Copyright (c) 1999-2004 by Erik Andersen <erik@ragnarok.com>	80		No	No	No
<input checked="" type="checkbox"/>		GPL-2.0-or-later			/busybox-1.35.0/	Copyright (c) 2003 Manuel Novoa III <mjn3@octopus.org>	80		No	No	No
<input checked="" type="checkbox"/>		GPL-2.0-or-later			/busybox-1.35.0/	Copyright (c) 1999-2005 by Erik Andersen <erik@ragnarok.com>	80		No	No	No
<input checked="" type="checkbox"/>		LGPL-2.1-or-later			/busybox-1.35.0/	Copyright (c) 2004 Kay Slevers <kay.slevers@openbsd.org>	80		No	No	No
<input checked="" type="checkbox"/>		GPL-2.0-or-later			/busybox-1.35.0/	Copyright (c) 2008 by Vladimir Dronnikov <v.dronnikov@openbsd.org>	80		No	No	No
<input checked="" type="checkbox"/>		GPL-2.0-or-later			/busybox-1.35.0/	Copyright (c) 2007 by Manuel Novoa III <mjn3@octopus.org>	80		No	No	No

Open Source Review Toolkit (ORT)

- Builds a “pipeline” of tools:
 - Analyzing dependencies (**Analyzer**)
 - Downloading dependencies (**Downloader**)
 - (informational) Scanning: generic API for different scanning tools, currently supports ScanCode and FOSSID (**Scanner**)
 - Retrieving Security Advisories (**Advisor**)
 - Evaluating license information and apply policy rules (**Evaluator**)
 - Creating a Bill of Material (**Reporter**)
 - Sending notifications (**Notifier**)
- Storage Backends to save and re-use scanning results (Local File, HTTP, PostgreSQL, ClearlyDefined)

ORT: Workflow



ORT: General information

- Nice getting started guide, but documentation quite limited
- Installation not straightforward
- **Input:** Source code
- **(Intermediate) output:** JSON, YAML in separate directories for each step together with input file

flict (FOSS License Compatibility Tool)

<https://github.com/vinland-technology/flict>

- Command line tool to verify license compatibility based on data provided externally (*e.g.* from the OSADL compatibility matrix).
- Can also suggest suitable leading license for a list of different licenses.
- Currently working on commit 5d17b262b3a74e35f854685cddfdbad224ee6027 as afterwards the OSADL matrix is imported via module and there is no documentation on how to do so.
- **Input:** list of licenses SPDX-ID or "common non SPDX ways to write licenses (e.g GPLv2)"
- **Output:** JSON, markdown, text, dot (graphical)

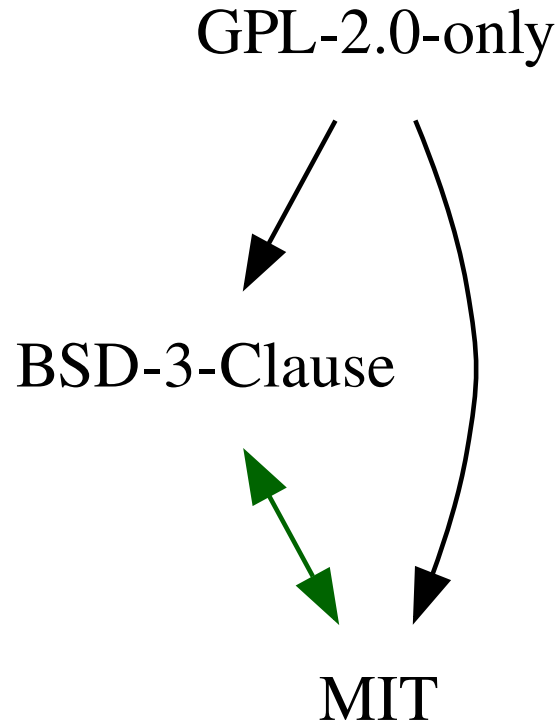
flict: Example JSON

```
flict display-compatibility BSD-3-Clause MIT GPL-2.0-only \
> compatibility.json
```

```
{
  "compatibilities": [
    { "license": "MIT",
      "licenses": [
        { "license": "GPL-2.0-only",
          "compatible_right": "true",
          "compatible_left": "false" },
        { "license": "BSD-3-Clause",
          "compatible_right": "true",
          "compatible_left": "true" } ] },
    { "license": "GPL-2.0-only",
      "licenses": [
        { "license": "MIT",
          "compatible_right": "false",
          "compatible_left": "true" },
        { "license": "BSD-3-Clause",
          "compatible_right": "false",
          "compatible_left": "true" } ] },
    { "license": "BSD-3-Clause",
      "licenses": [
        { "license": "MIT",
          "compatible_right": "true",
          "compatible_left": "true" },
        { "license": "GPL-2.0-only",
          "compatible_right": "true",
          "compatible_left": "false" } ] } ] } ] }
```


flict: Example graphical

```
flict -of dot display-compatibility BSD-3-Clause MIT GPL-2.0-only \  
> compatibility.dot
```



Component catalog

“Homebrew”

- For small to medium sized projects, it might be sufficient to collect information manually, *e.g.*
 - Create hashes of already scanned and cleared source code files.
 - Store these in a database together with compliance information / material and optionally additional information (*e.g.* vulnerabilities).
 - If there are new files, also create hashes and compare with database to reuse the information.
 - Also use proximity hashes (see following bonus talk by Armijn Hemel).

SW360

<https://github.com/eclipse/sw360>, <https://www.eclipse.org/sw360/>

Wiki: <https://github.com/eclipse/sw360/wiki>

- Extensive multi-user software component management tool and database for various aspects of software clearing:
 - Collect information from external tools like license scanner, clearing tools (in particular FOSSology), code quality checker, security vulnerability scanner, forensic scanner and the source code itself.
 - Group components by release to produce material required for particular use cases.
 - Organize clearing workflows, enforce policies and create and maintain project BOM.
 - Assign attributes to releases and tasks to different types of users.

SW360: General information

- Server application; the GUI can be accessed via browser.
- Ongoing work on integrating OSADL License Obligation Checklists.
- **Difficult to set up!**
- Screenshots:
<https://www.eclipse.org/sw360/screenshots/>

Software BOM

- Various tools can output the compliance information as a software BOM, *e.g.*
 - Opossum
 - ORT
 - SW360

Security

- Workflow and processes are similar for managing known security vulnerabilities and managing license compliance.
- A lot of ongoing work on combining both aspects, *e.g.* <https://www.openchainproject.org/security-guide>
- ORT and SW360 enable integration of external information on vulnerabilities.

CVEhound

<https://github.com/evdenis/cvehound>

- Command line tool to find unfixed (fix not available or not applied) known CVEs in Linux kernel source code.
- Includes patterns to find known CVEs; regularly updated.
- Easy to install and use but some difficulties with version compatibility (Python, coccinelle).
- **Input:** Linux kernel source code and CVE patterns
- **Output:** Text, JSON report with info on CVEs

CVEhound: Example

```
$ cvehound -k /usr/src/kernels/linux-5.4.1
```

```
Found: CVE-2020-9391
```

```
Found: CVE-2020-14331
```

```
Found: CVE-2021-27363
```

```
Found: CVE-2021-3715
```

```
Found: CVE-2020-27830
```

```
Found: CVE-2019-19332
```

```
Found: CVE-2020-10732
```

```
[...]
```

Snippet matching (Forensic scanning)

General information

- Mainly proprietary, commercial tools.
- Do NOT use forensic scanners for informational scanning!
- Be aware that the interpretation of scan results is very time consuming: Can result in a significant budget overhead for a project.

ScanOSS

<https://www.scanoss.com/> (two informative Whitepapers available on the website)

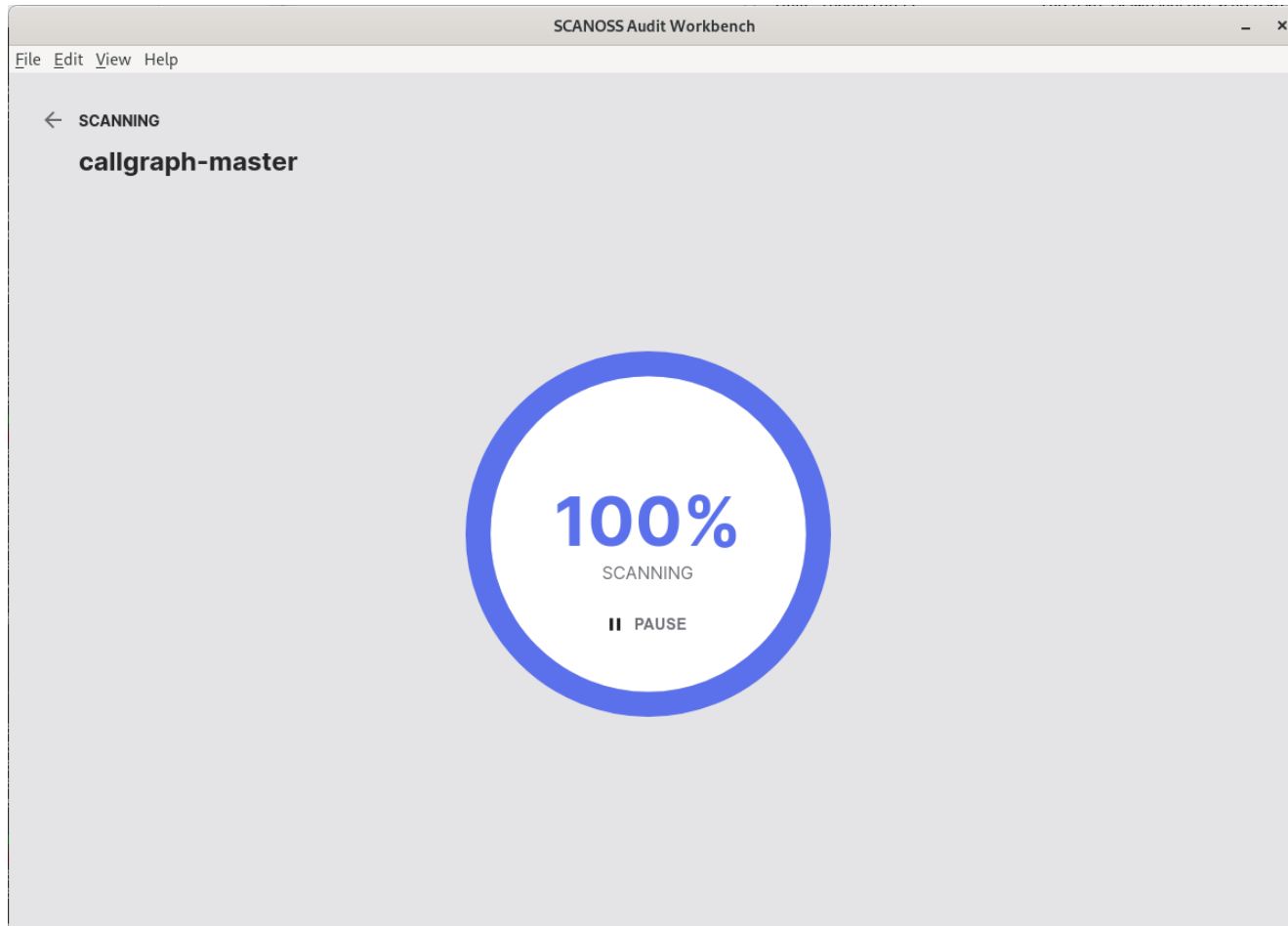
<https://github.com/scanoss>

- Software Composition Analysis tool for creating and maintaining a BOM during the development process.
- Performs component, file and snippet analysis of source code (third-party and own development) and compares to the project's Open Source Knowledge Base (<https://osskb.org/>).
- Lists actual licenses of matched files/snippets and licenses that are incompatible with these.

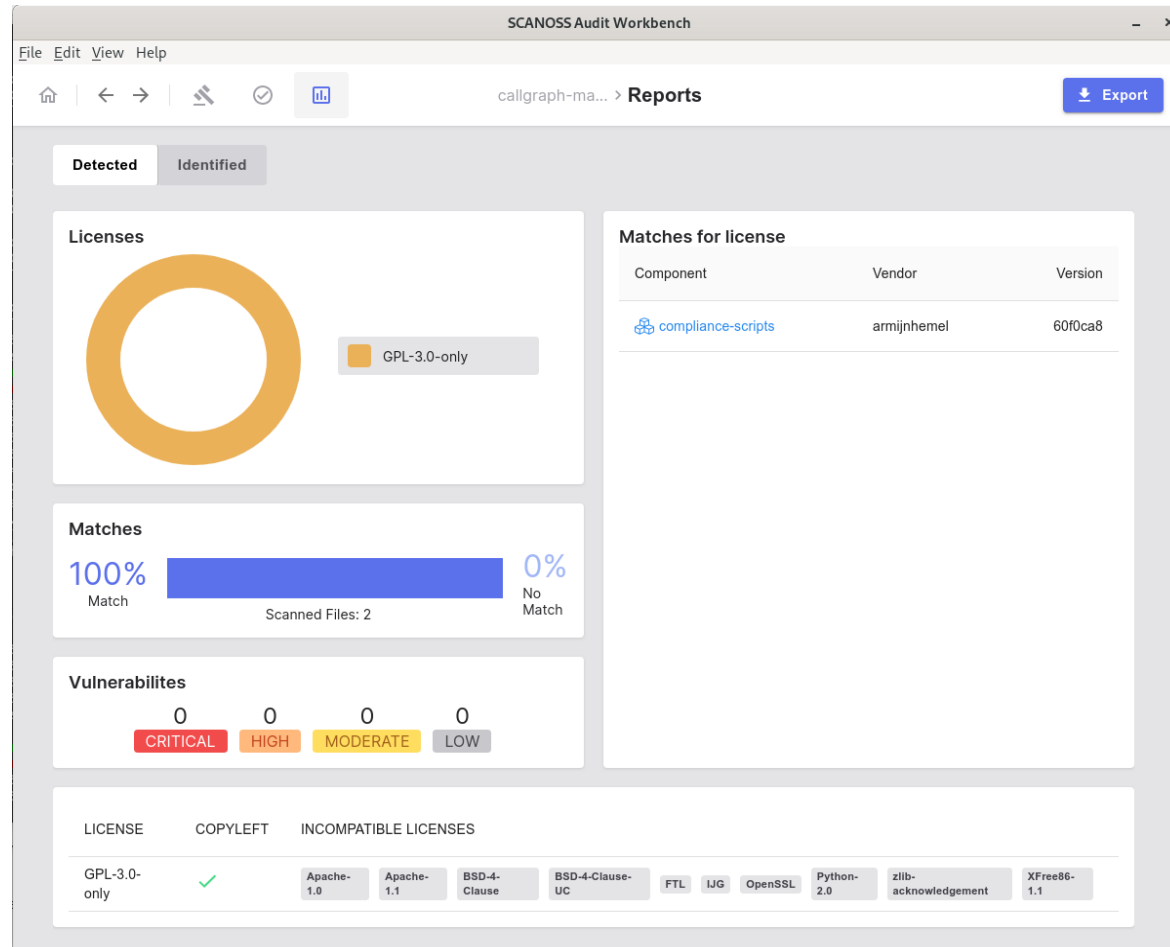
ScanOSS: General information

- Only fingerprints of local source code are sent online.
- Connection to OSSKB via RESTful API from client app (ScanOSS Audit Workbench), webhook or command line interface for automation or integration into CI/CD.
- Workbench is available as standalone binary image, no installation needed.
- **Input:** Source code
- **Output:** CSV, SPDX-light, raw JSON

ScanOSS Audit Workbench Example: callgraph



ScanOSS Audit Workbench: Detected



ScanOSS Audit Workbench: Matched file

The screenshot displays the ScanOSS Audit Workbench interface. The window title is "SCANOSS Audit Workbench". The menu bar includes "File", "Edit", "View", and "Help". The breadcrumb navigation shows "callgraph-master" > "Matches" with a 0% progress indicator. The left sidebar shows a file tree for "callgraph-master" with subfolders "graphics", "pics", and files ".gitignore", "LICENSE", "README.md", "example.cypher", "generatocypher.py", and "graph.config". The main area shows a detected file match for "example.cypher". A badge indicates "compliance-scripts" with a "DETECTED File" status. Below this, there are sections for "Source File" and "Component File", both pointing to "example.cypher". The main content area displays the source code of the file, which is a list of system paths in a structured format.

```
1 CREATE (TghCtaic:ELF {name: '/usr/sbin/download'}),
2 (ajgdnlom:ELF {name: '/lib/libthreadutil.so'}),
3 (JmejdSde:ELF {name: '/usr/sbin/mini_httpd'}),
4 (sHVAGoSU:ELF {name: '/usr/sbin/netbiosname'}),
5 (AcrdpuFk:ELF {name: '/lib/libgcc_s.so.1'}),
6 (RQLPyINH:ELF {name: '/lib/libc.so.0'}),
7 (jNQYwDv:ELF {name: '/bin/busybox'}),
8 (xAppyUrr:ELF {name: '/usr/sbin/wpa_supplicant'}),
9 (qPxBUDqW:ELF {name: '/lib/libaplog.so'}),
10 (AAwDdabB:ELF {name: '/usr/sbin/pb_ap'}),
11 (YaxFIgid:ELF {name: '/lib/libutil.so.0'}),
12 (UyajGqYq:ELF {name: '/lib/libxml.so'}),
13 (MAKH00Fc:ELF {name: '/lib/libthreadutil.so.0'}),
14 (Lxxkg10B:ELF {name: '/usr/sbin/wscupnpd'}),
15 (NMtUAEJK:ELF {name: '/lib/ld-uclibc.so.0'}),
16 (LPEELOId:ELF {name: '/lib/libcrypt-0.9.28.so'}),
17 (VaQ0SCCX:ELF {name: '/lib/libclic-0.9.28.so'}),
18 (sVxsxJbn:ELF {name: '/usr/sbin/vconfig'}),
19 (RFcwTGGj:ELF {name: '/usr/sbin/iwconfig'}),
20 (TuHBUJZD:ELF {name: '/usr/sbin/udhcpd'}),
21 (eYywoiW:ELF {name: '/usr/sbin/apmgr'}),
22 (UjpAq0SA:ELF {name: '/lib/libfwupld.so'}),
23 (YnMnZJni:ELF {name: '/lib/libresolv-0.9.28.so'}),
24 (NfVpTzRB:ELF {name: '/lib/libresolv.so.0'}),
25 (ZZWco0qc:ELF {name: '/lib/libm.so'}),
26 (WtFvRezN:ELF {name: '/sbin/rc'}),
27 (UhlncZLX:ELF {name: '/usr/sbin/nvram'}),
28 (AostVjOl:ELF {name: '/usr/www-ap/setup.cgi'}),
29 (SmsoxrCr:ELF {name: '/usr/www-ap/download.cgi'}),
30 (gdPqwFPi:ELF {name: '/lib/libeditapcfg.so'}),
31 (AKGIkky:ELF {name: '/usr/sbin/scsyslogd'}),
32 (LZKwIpb:ELF {name: '/usr/sbin/ld2'}),
33 (tIndkmVl:ELF {name: '/lib/libm-0.9.28.so'}),
34 (dmbheCTN:ELF {name: '/lib/libmatrixssl.so'}),
35 (zYZCaRZI:ELF {name: '/lib/ld-uclibc-0.9.28.so'}),
36 (Fj1VEYdt:ELF {name: '/usr/sbin/snmptap'}),
37 (iRjxlLn:ELF {name: '/lib/libdl.so.0'}),
38 (mIBxjSY:ELF {name: '/usr/sbin/hostapd'}),
39 (ZTLZtPD:ELF {name: '/usr/sbin/cfgTest'}),
40 (KERBSceU:ELF {name: '/lib/libutil.so'}),
41 (EZFBqXLT:ELF {name: '/lib/libgcc_s.so'}),
42 (ciFsA0Un:ELF {name: '/lib/libapcfg.so'}),
```


ScanOSS Audit Workbench: Matched snippet

The screenshot displays the ScanOSS Audit Workbench interface. At the top, the window title is "SCANOSS Audit Workbench". Below the title bar is a menu bar with "File", "Edit", "View", and "Help". A navigation bar shows a home icon, navigation arrows, a search icon, and a status bar with "callgraph-ma... > Matches" and a 50% progress indicator.

The main content area is divided into three panels:

- Left Panel:** A file tree showing the project structure. The file "generatecypher.py" is selected and highlighted in blue.
- Center Panel:** Displays the source file "generatecypher.py". A blue box highlights a "DETECTED Snippet" with the identifier "compliance-scripts 60f0ca8".
- Right Panel:** Shows the "Component File" for "generatecypher.py".

The source file content is as follows:

```
1 #!/usr/bin/env python3
2
3 # This script walks a directory of files extracts symbols from ELF files,
4 # records dependencies (taking symbolic links and RPATH into account) and
5 # generates different types of output.
6
7 # The method works as follows:
8
9 # 1. walk a directory of files and store:
10 # a) names of dynamically ELF files
11 # b) symbols defined by the ELF files (including visibility,
12 # type and binding)
13 # c) symbols exported by the ELF files (including binding, type, and so on)
14 # d) dependencies declared in dynamically linked files,
15 # possibly indirect (symbolic links)
16
17 # 2. for each group of binaries (architecture, endianness, etc.) it
18 # will then generate output files with all the information from 1.
19
20 # The typical use case would be a firmware of an embedded system that
21 # has been unpacked first into a separate directory.
22
23 # Background material about the method can be found here:
24
25 # https://lwn.net/Articles/548216/
26 # https://github.com/armijnhemel/conference-talks/tree/master/fsfe2013
27
28 # ELF background information can be found in public sources here:
29
30 # https://en.wikipedia.org/wiki/Executable_and_Linkable_Format
31 # https://en.wikipedia.org/wiki/Weak_symbol
32 # https://refspecs.linuxbase.org/elf/elf.pdf
33 # https://android.googlesource.com/platform/art/+master/runtime/elf.h
34 # https://docs.oracle.com/cd/E19683-01/016-1386/chapter6-43405/index.html
35
36 # Licensed under the terms of the General Public License version 3
37
38 # SPDX-License-Identifier: GPL-3.0-only
39
40 # Copyright 2018-2019 - Armijn Hemel, Tjaldar Software Governance Solutions
41 # Copyright 2021 - Open Source Automation Development Lab (OSADL) eG, author Carsten Emde
42
43 import argparse
44 import configparser
45 import os
46 import secrets
47 import string
48 import sys
49 import tempfile
50
51 # import pyelftools
52 import elftools.elf.elffile
53 import elftools.elf.dynamic
54 import elftools.elf.sections
55
56 def notarget(filename, limitsearch):
57     if len(limitsearch) == 0:
58         return False
59     filefound = False
60     for file in limitsearch:
61         if filename == file:
62             filefound = True
63         break
64     return not filefound
65
66 def createoutput(outputdir, outputformat, machine to binary, linked libraries,
67                 filename to full path, elf to exported symbols,
```

ScanOSS Audit Workbench: Identify match

The screenshot displays the ScanOSS Audit Workbench interface. The main window shows a file tree on the left with 'generatecypher.py' selected. The central pane displays the source code of this file, which includes a license header and a function definition. A modal window titled 'Identify Component' is open, showing a search for 'compliance-scripts' with version '60f6ca8' and license 'GNU General Public License v3.0 only'. The modal also shows the URL 'https://github.com/armijnhemel/compliance-scripts', the PURL 'pkg:github/armijnhemel/compliance-scripts', and the usage 'Snippet'. The 'Identify' button is highlighted in green.

File Edit View Help

SCANOSS Audit Workbench

callgraph-master > Matches 50%

callgraph-master > generatecypher.py

compliance-scripts 60f6ca8 DETECTED Snippet

Source File generatecypher.py

Component File generatecypher.py

```
1 #!/usr/bin/env python3
2
3 # This script walks a directory of files extracts symbols from ELF
4 # records dependencies (taking symbolic links and RPATH into account)
5 # generates different types of output.
6 #
7 # The method works as follows:
8 #
9 # 1. walk a directory of files and store:
10 # a) names of dynamically ELF files
11 # b) symbols defined by the ELF files (including visibility,
12 # type and binding)
13 # c) symbols exported by the ELF files (including binding, type
14 # d) dependencies declared in dynamically linked files,
15 # possibly indirect (symbolic links)
16 #
17 # 2. for each group of binaries (architecture, endianness, etc.) it
18 # will then generate output files with all the information from
19 #
20 # The typical use case would be a firmware of an embedded system that
21 # has been unpacked first into a separate directory.
22 #
23 # Background material about the method can be found here:
24 #
25 # https://lwn.net/Articles/548216/
26 # https://github.com/armijnhemel/conference-talks/tree/master/fsf
27 #
28 # ELF background information can be found in public sources here:
29 #
30 # https://en.wikipedia.org/wiki/Executable_and_Linkable_Format
31 # https://en.wikipedia.org/wiki/Weak_symbol
32 # https://refspecs.linuxbase.org/elf/elf.pdf
33 # https://android.googlesource.com/platform/art/+/master/runtime/elf
34 # https://docs.oracle.com/cd/E19683-01/816-1386/chapter-43402/index.html
35 #
36 # Licensed under the terms of the General Public License version 3
37 #
38 # SPDX-License-Identifier: GPL-3.0-only
39 #
40 # Copyright 2018-2019 - Armijn Hemel, Tjaldur Software Governance
41 # Copyright 2021 - Open Source Automation Development Lab (OSADL)
42 #
43 import argparse
44 import configparser
45 import os
46 import secrets
47 import string
48 import sys
49 import tempfile
50
51 # import pyelftools
52 import elftools.elf.elffile
53 import elftools.elf.dynamic
54 import elftools.elf.sections
55
56 def notarget(filename, limitsearch):
57     if len(limitsearch) == 0:
58         return False
59     filefound = False
60     for file in limitsearch:
61         if filename == file:
62             filefound = True
63             break
64     return not filefound
65
66 def createoutput(outputdir, outputformat, machine to binary, linked_libraries,
67                 filename_to_full_path, elf to exported symbols,
```

ScanOSS Audit Workbench: Identified matches

SCANOSS Audit Workbench

File Edit View Help

callgraph-ma... > **Identified components** 100%

- callgraph-master
 - graphics
 - pics
 - .gitignore
 - LICENSE
 - README.md
 - example.cypher
 - generatencypher.py
 - graph.config

compliance-scripts

USAGE
File Identified Group

VERSION
60f0ca8

LICENSE
GNU General Public License V3.0 Only

n/a

[View files →](#)

USAGE
Snippet Identified Group

VERSION
60f0ca8

LICENSE
GNU General Public License V3.0 Only

n/a

[View files →](#)


SCANOSS Audit Workbench

File Edit View Help

callgraph-ma... > **Reports** [Export](#)


Detected **Identified**

Identification Progress

100% 


14 total files

Licenses




GPL-3.0-only

Matches for license

Component	Vendor	Version
 compliance-scripts	compliance-scripts	60f0ca8

LICENSE COPYLEFT INCOMPATIBLE LICENSES

GPL-3.0-only  Apache-1.0 Apache-1.1 BSD-4-Clause BSD-4-Clause-UC FTL IJG OpenSSL Python-2.0 zlib-acknowledgement XFree86-1.1

Compliance toolchain: Requirements

External Input:
What information is provided by suppliers?

Contribution:
Compliance information is contributed to public projects (e.g. ClearlyDefined)

Analyzing:
What software is used and how?

Scanning:
Which licenses?
Extract information.

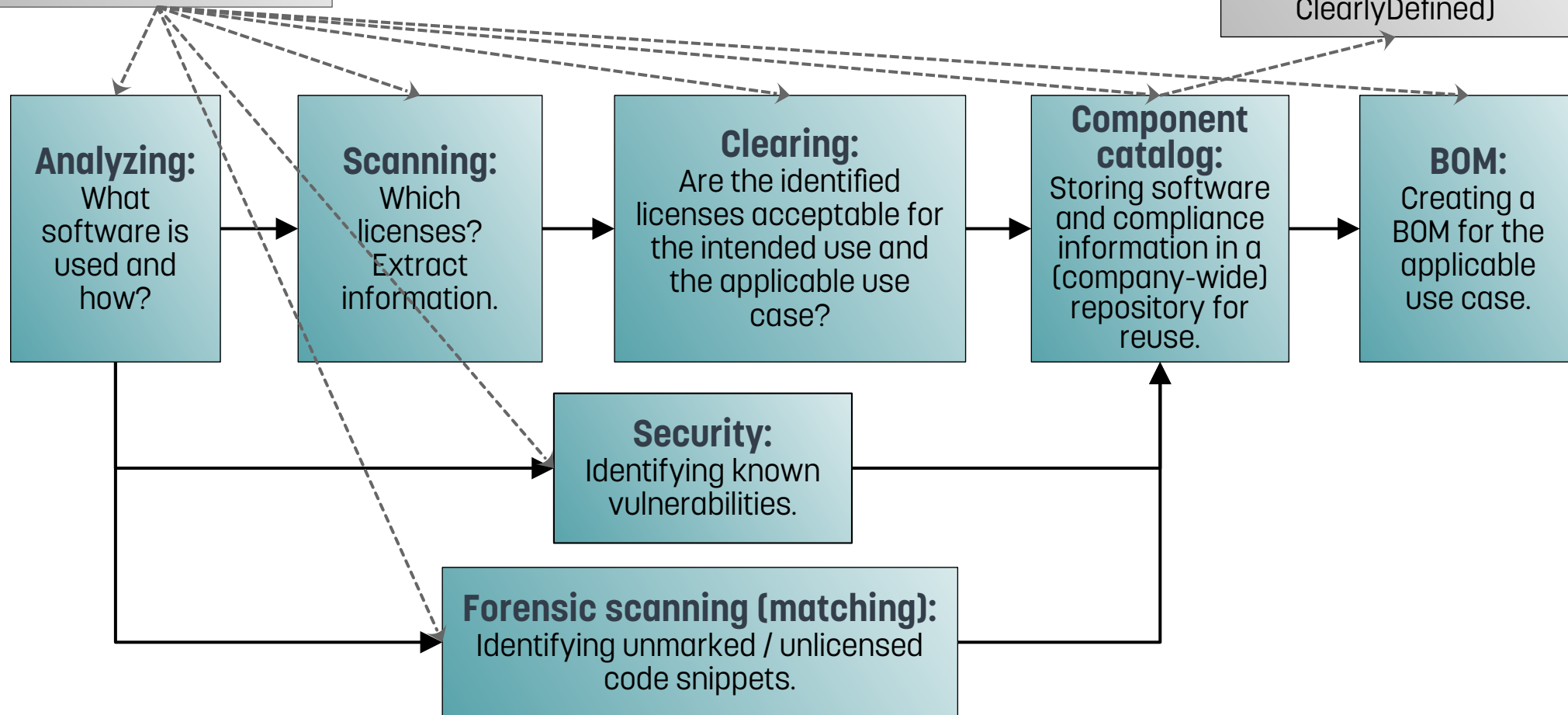
Clearing:
Are the identified licenses acceptable for the intended use and the applicable use case?

Component catalog:
Storing software and compliance information in a (company-wide) repository for reuse.

BOM:
Creating a BOM for the applicable use case.

Security:
Identifying known vulnerabilities.

Forensic scanning (matching):
Identifying unmarked / unlicensed code snippets.



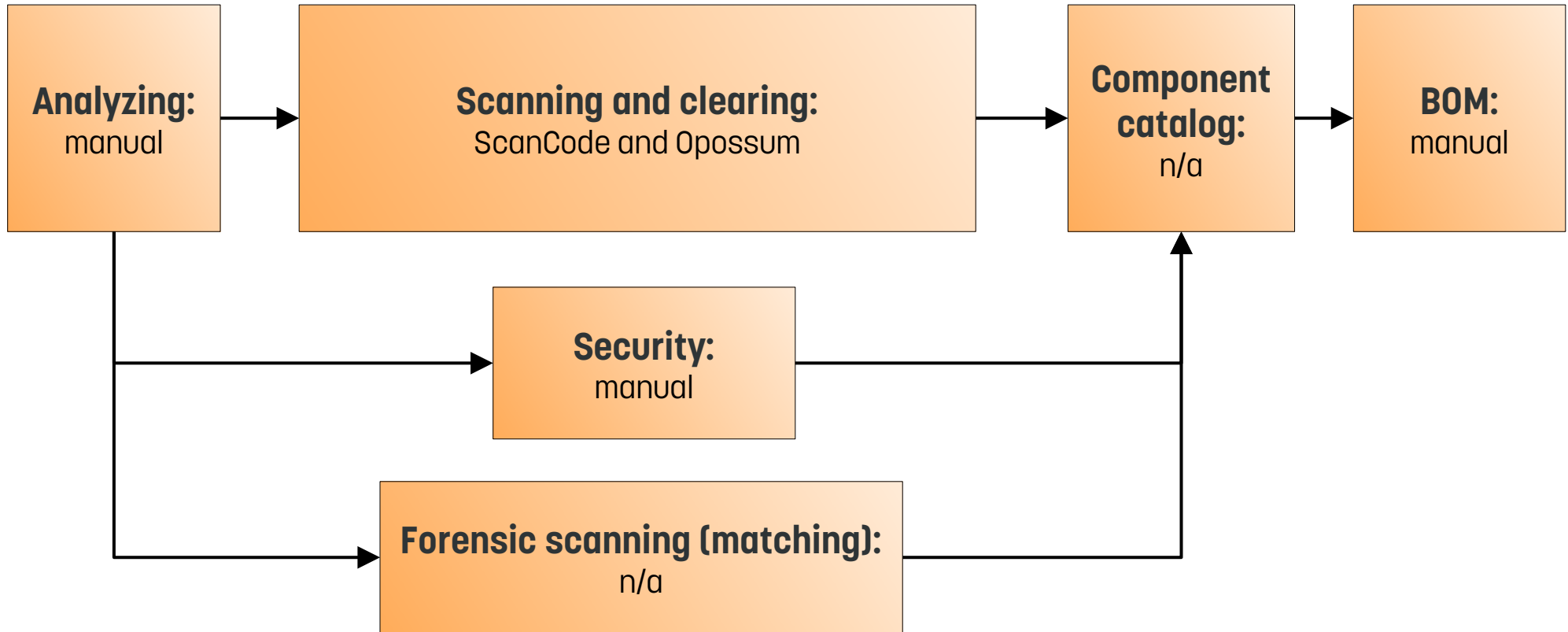
General notes

- Compliance is an **iterative process!**
- Results of the clearing step might influence the original architecture or choice of software.
- The compliance process must start together with the project. As soon as the general architecture is decided on, a first round of the compliance toolchain should be undergone.
- Additional components are cleared as they come along.
- Responsibilities and tasks are to be assigned in accordance with company structures.

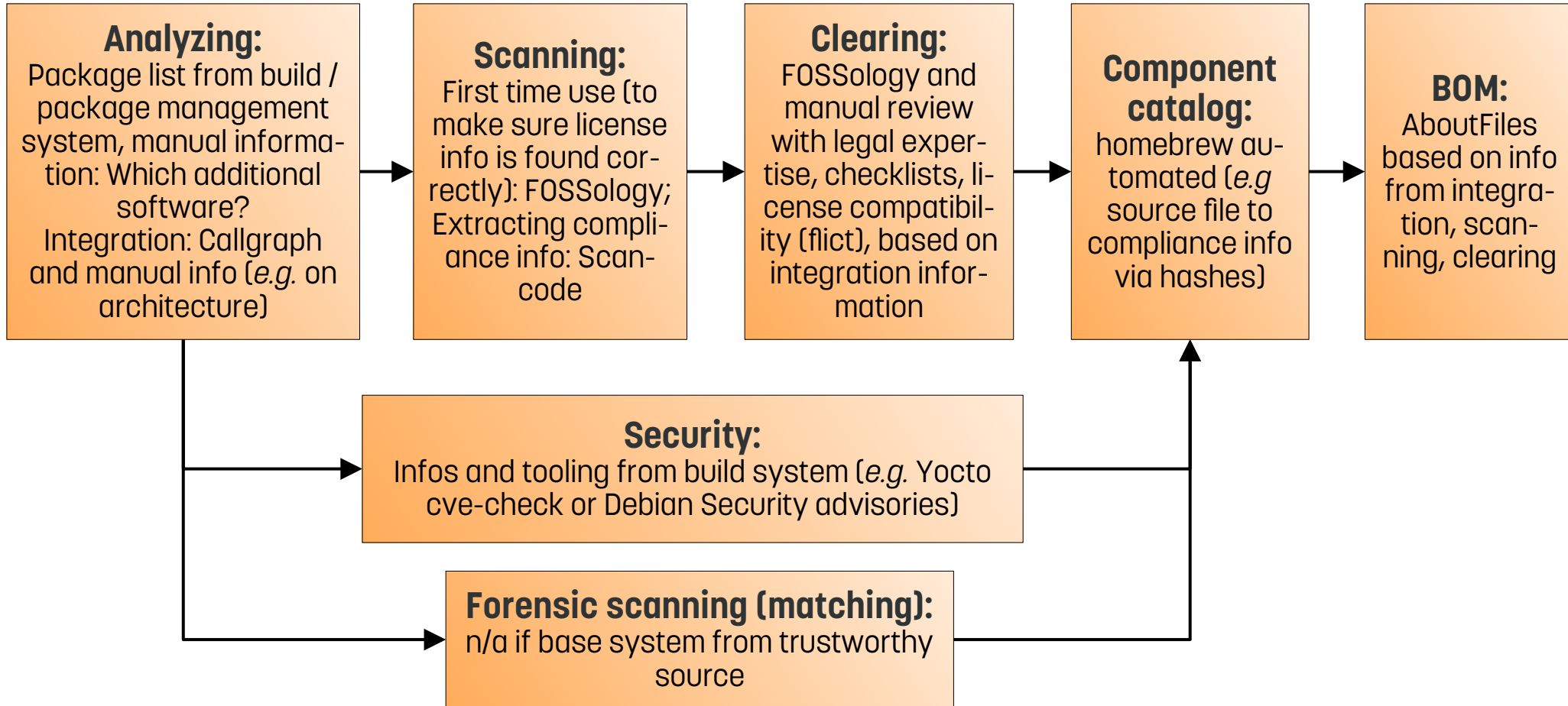
Example cases

- For the following exemplary cases, a selection of suitable tools for each step of the compliance toolchain is suggested.
- It is recommended to implement a toolchain for the most complex applicable case and use this toolchain for all cases.
- Cases:
 - Isolated component
 - Embedded system
 - Platform project (*e.g.* Linux distro with general proprietary components that various users can build their products on)
 - Container
 - *Large project (t.b.d.)*
 - *Whole company (t.b.d.)*

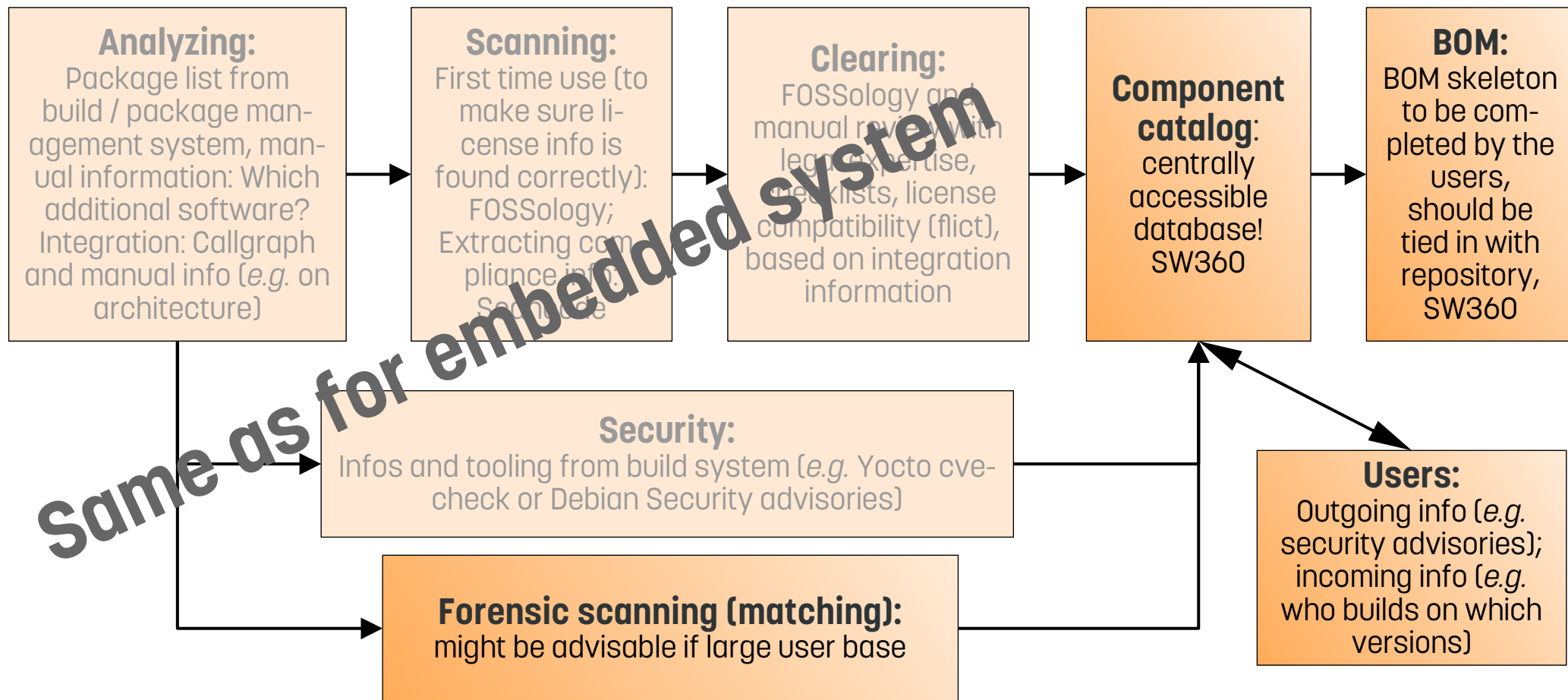
Compliance toolchain – Example 1: Minimal toolchain for isolated components



Compliance toolchain – Example 2: Embedded system



Compliance toolchain – Example 3: Platform project



Compliance toolchain – Example 4: Container

BOM:
Should be tied in with repository, SW360

Analyzing:
Which base image? Which layers? (as each layer must be analyzed separately);
How are components integrated?
Tools:
container-inspector (to separate layers), package management system, Callgraph and manual info (e.g. on architecture)

Extracting:
Obtain CCSC of used software, Package management system and manually

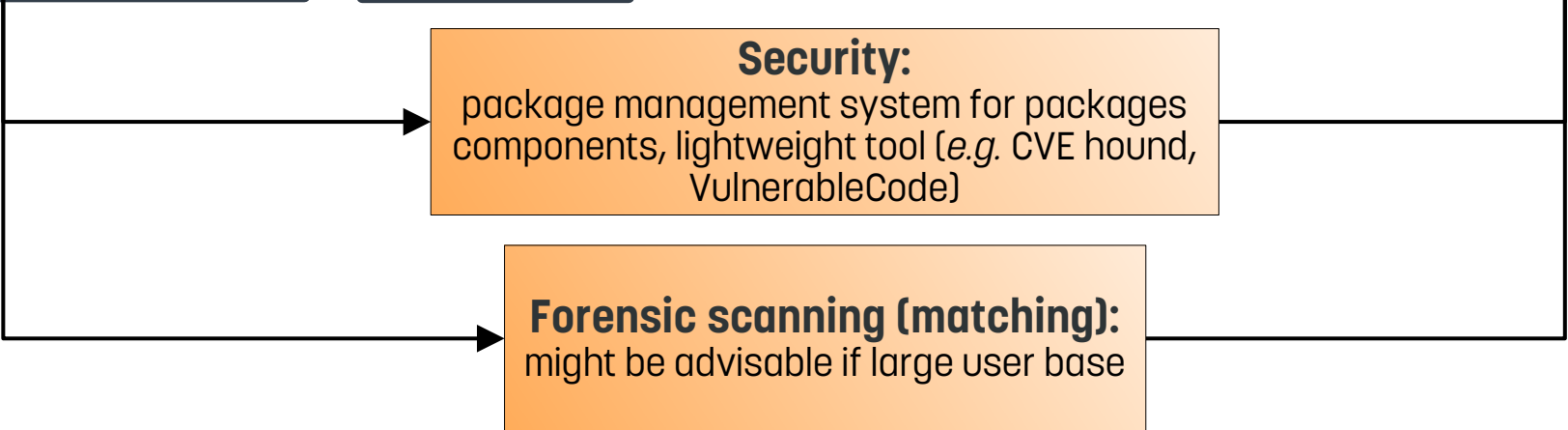
Scanning:
First time use (to make sure license info is found correctly): FOSSology;
Extracting compliance info: Scancode

Clearing:
FOSSology and manual review with legal expertise, checklists, license compatibility (flict), based on integration information

Repository:
centrally accessible database!
SW360

Security:
package management system for packages components, lightweight tool (e.g. CVE hound, VulnerableCode)

Forensic scanning (matching):
might be advisable if large user base



Conclusion

- Integrating FOSS compliance into the development workflow is done in **several stages**.
- There is a large (and growing) number of tools available to support each stage.
- Every tool should be used only for the **intended tasks**.
- It is recommended to combine tools and, if warranted, even use different tools for the same tasks to achieve more reliable results.
- **Not everything can be automated!** Human expertise and some manual labor are always required.