Statutory declaration

I declare that I have composed the master thesis myself and without use of any other than the cited sources and aids. Sentences or parts of sentences quoted literally are marked as such; other references with regard to the statement and scope are indicated by full details of the publications concerned. The thesis in the same or similar form has not been submitted to any examination body and has not been published. This thesis was not yet, even in part, used in another examination or as a course performance. Furthermore I declare that the submitted written (bound) copies of the master thesis and the version submitted in digital format are consistent with each other in contents.

(Place, Date) (Fabian Burth)

Abstract

The importance of software is constantly growing and so is its complexity. Thus, large-scale enterprise software systems are usually not developed completely from scratch. Commonly required functionality like logging or serialization is often provided by already existing *open source software (OSS)*. Hence, modern software systems are composed of several components. These components usually have individual version updates, vulnerabilities, and licenses. Version updates of a component may affect the compatibility with other components. Vulnerabilities in one specific component may compromise the security of the whole software system. Violating license agreements may lead to litigations due to copyright infringement. Furthermore, one must consider that each component itself may be composed of several other components.

So, maintaining and monitoring large-scale enterprise software systems is an important part of the *Application Lifecycle Management (ALM)* and poses a considerable challenge to software companies. The common way to tackle these risks nowadays is by incorporating *Software Composition Analysis (SCA)* tools into the application development process. These tools analyze applications and retrieve information like vulnerabilities, licenses, and *software bill of materials (SBOM)*. But this approach often still has its flaws. The information extracted by these tools is frequently treated like logs and hence of limited value for future usage. Additionally, in larger companies different development teams often come up with point-to-point solutions of integrating the tools tightly coupled to their CI/CD pipeline.

The main objective of this thesis is the design and prototypical implementation of a *Security and Compliance Data Lake (SCDL)*, which provides a standardized way of integrating even multiple different SCA tools loosly coupled to CI/CD pipelines and to store the extracted information. By offering an *Application Programming Interface (API)*, it then should enable consumers to query this information on different levels of aggregation to answer questions that might not even have been known at the time the SCA was performed. A recent and popular example for such a question is "Which components contain log4j?".

This *Security and Compliance Data Lake* will build upon the *Open Component Model (OCM)*, an open standard to describe the SBOM with so called *Component Descriptors*. These *Component Descriptors* also describe how to access sources and resources. It thereby provides an entry point for the execution of SCA tools.

Contents

List of Figures

List of Tables

Acronyms

Chapter 1

Introduction

1.1 Motivation

1.2 Goals

1.3 Environment

1.4 Structure of the Thesis

Chapter 2

Foundations

2.1 Terminology

This section explains several abbreviations and standards which are commonly mentioned alongside the topics *Software Bill of Materials*, *Vulnerability Management* and *Open Source Licensing* and are thus essential for the further understanding.

Common Platform Enumeration (CPE): The CPE specification originally created by MITRE and now maintained by NIST provides a naming scheme for IT assets such as software. It may be used to uniquely determine a specific software and its version. This way a CPE enables cross referencing to other sources of information. The commonly used CPE naming scheme is structured as follows:

```
cpe:2.3: part : vendor : product : version : update : edition :
   language : sw_edition : target_sw : taget_hw : other
```

Listing 2.1: CPE Formatted String Binding

Thereby `part` may be *a* for applications, *o* for operating systems, and *h* for hardware devices. `edition` is a legacy attribute in the current version of the specification and may be omitted where not required for backward compatibility. The attributes after `edition` were newly introduced in this version and are referred to as *extended attributes*. `sw_edition` should characterize a particular market or class of users a product is tailored to (e.g. online), `target_sw` a software computing environment (e.g. linux), `target_hw` the instruction set architecture (e.g. x86), and `language` the language supported in the user interface [1].

Package Information: Maven Coordinates (GAV - groupId:artifactId:version)

NPM Coordinates Package URLs CPE (Common Platform Enumeration): CVE (Common Vulnerabilities and Exposures): CNA (CVE Numbering Authorities): CWE (Common Weakness Enumeration): CVSS (Common Vulnerability Scoring System): NVD (U.S. National Vulnerability Database): CERT/CC Vulnerability Notes Database: Vulnerability: Exposure:

2.2 Software Bill of Materials

A *Software Bill of Materials* is an inventory of the components used in a software. It ideally contains all direct and transitive components and their dependencies, so it is in other words pretty much the dependency graph of a software [2, 3].

2.2.1 Contents and Requirements

As consequence to an *Executive Order on Improving the Nation's Cybersecurity*, the *National Telecommunications and Information Administration (NTIA)* published a document describing the minimum requirements for SBOMs [4, 3]). According to this document, these are:

Data Fields (Metadata)	Baseline information about each component: Supplier, Component Name, Version of the Component, Other Unique Identifiers, Dependency Relationship, Author of SBOM Data, Timestamp of SBOM creation
Automation Support	Automatic generation and machine-readability to allow for scaling across the software ecosystem.
Practices and Processes	Implementation of policies, contracts and arrangements to maintain SBOMs.

Table 2.1: Minimum Elements of a SBOM
Source: [3]

The goal of the *Data fields* is to sufficiently identify the components to track them through the supply chain and map them to other data sources, such as vulnerability and license databases. The *Automation Support* provides the ability to scale across the software ecosystem. The *Practices and Processes* ensure the maintenance by integration into the ALM. SBOMs thereby increase software transparency, providing

those who produce, purchase and operate software the means to perform proper risk assessments [3].

2.2.2 Projects and Data Formats

In the following, four different projects that provide machine-readable data formats for SBOM-description will be introduced. The first three of them are commonly used and also mentioned in the document published by the NTIA. The fourth is an relatively new open standard developed and used within SAP.

Software Package Data eXchange (SPDX):
SPDX is an initiative founded in 2010 and hosted at *The Linux Foundation*. In 2021 the SPDX specification even became an ISO standard [5]. The initiative focuses on solving challenges regarding the licenses and copyrights associated with software packages. SPDX therefore assembles licenses and exceptions commonly found in OSS in the *SPDX License List*. More precisely, this list includes a standardized short identifier, the full name, the license text, and a canonical permanent URL for each license and exception. By incorporating this *SPDX License Identifiers* in source on file level, one enables automation of concise license detection, even if just parts of an OSS project are used. Furthermore, SPDX provides *Matching Guidelines* to ensure that e.g. a "BSD 3-clause" license in a LICENSE file of an OSS project with different capitalization or usage of white space than the master license text included in the *SPDX License List* is still identified as "BSD 3-clause" license.

At the heart of the SPDX initiative are the *SPDX Documents* which leverage the *SPDX License List* and *SPDX License Identifiers* to describe the licensing of a set of associated files, referred to as *Package* in the context of SPDX. A *SPDX Document* provides means to describe information about the document creation, the package as a whole, individual files, snippets of code within an individual file and other licenses that are not contained in the *SPDX License List* but are still relevant for the package, relationships between *SPDX Documents*, and annotations, which in a way are comments within an *SPDX Document*. The concept of relationships is a rather new addition to the specification. It is particularly useful if one has an SPDX Document describing a binary. Explicitly capturing relationships like "generated from" these source files and "dynamically linking" these libraries allows for a complete licensing picture.

These documents may be represented in one of the following five file format: tag/value (.spdx), JSON (.spdx.json), YAML (.spdx.yaml), RDF/xml (.spdx.rdf), and spreadsheets (.xls) [6, 7].

To give a more concrete idea of the basic concepts of *SPDX Documents*, an example from the SPDX GitHub repository will be briefly examined [8]. Therefore figure 2.1 below shows the directory structure of a “Hello World” project in C.

Figure 2.1: C Project Directory Structure
Source: [8]

Listing 2.2 shows a corresponding *SPDX Document*. Some tag:value pairs which are less relevant for the overall understanding are deliberately omitted to contain the length of the example.

```
SPDXVersion: SPDX-2.2
DataLicense: CC0-1.0
SPDXID: SPDXRef-DOCUMENT
DocumentName: hello
DocumentNamespace: https://swinslow.net/spdx-examples/example1/hello-v3
Creator: Person: Steve Winslow (steve@swinslow.net)
Created: 2021-08-26T01:46:00Z

##### Package: hello
PackageName: hello
SPDXID: SPDXRef-Package-hello
PackageDownloadLocation: git+https://github.com/swinslow/spdx-examples.git#example1/content
PackageLicenseConcluded: GPL-3.0-or-later
PackageLicenseInfoFromFiles: GPL-3.0-or-later
PackageLicenseDeclared: GPL-3.0-or-later
PackageCopyrightText: NOASSERTION

FileName: /build/hello
SPDXID: SPDXRef-hello-binary
FileType: BINARY
LicenseConcluded: GPL-3.0-or-later
LicenseInfoInFile: NOASSERTION
FileCopyrightText: NOASSERTION

FileName: /src/Makefile
SPDXID: SPDXRef-Makefile
FileType: SOURCE
LicenseConcluded: GPL-3.0-or-later
LicenseInfoInFile: GPL-3.0-or-later
FileCopyrightText: NOASSERTION

FileName: /src/hello.c
SPDXID: SPDXRef-hello-src
FileType: SOURCE
LicenseConcluded: GPL-3.0-or-later
LicenseInfoInFile: GPL-3.0-or-later
FileCopyrightText: Copyright Contributors to the spdx-examples project.

Relationship: SPDXRef-hello-binary GENERATED_FROM SPDXRef-hello-src
Relationship: SPDXRef-hello-binary GENERATED_FROM SPDXRef-Makefile
```

```
Relationship: SPDXRef-Makefile BUILD_TOOL_OF SPDXRef-Package-hello
```

Listing 2.2: SPDX Document

Most of the tag:value pairs are self-explanatory, but some might require some explanation. The *Concluded License* is the license the SPDX file creator has concluded as the governing license of a package or a file. *License Information from Files* contains a list of all licenses found in a package and the *Declared License* is the license declared by the authors of the package [7]. Additionally, listing 2.2 illustrates how the concept of relationships may be used.

It also worth mentioning that the concept of *Packages* in SPDX as a set of associated files is really rather loose. Thus, describing the project in figure 2.1 as two separate packages, one for source and one for binary, optionally in the same or also in two separate *SPDX Documents* would be completely conform with the specification as well.

Since SPDX has been around for so long and is an accepted ISO standard, there exists a lot of useful tooling. It is therefore quite easy to automate tasks like producing, consuming, transforming and validating *SPDX Documents* [6].

While SPDX is a really good standard to describe licensing and copyright information about packages, it has some flaws when being viewed as a standalone SBOM standard. A main point of criticism is associated to how the SPDX specification handles the mapping to other data sources. Through *SPDX External References* it allows a *Package* to reference an external source of information believed to be relevant to the *Package*. This allows one *Package* to have multiple CPEs or package URLs if these references are believed to be relevant. This provides a lot of flexibility, but also means that SPDX does not necessarily uniquely identify each *Package* in the context of other data sources.

CycloneDX:
-Further requirements for SBOMs -Integrate into build system because of the information from package managers -Ideally incorporate all the ways of creating SBOMs and merge the information somehow
Software Identification (SWID) tags:

Open Component Model (OCM):

2.2.3 Creation

Add a section about supply chain attacks to stress the necessity of SBOMS?

2.3 Vulnerabilities

2.4 OSS Licensing

2.5 Context at SAP Gardener

Builds, Repositories, Gardener Servicelet Pattern (Session mit Uwe)

2.5.1 SAP Gardener

2.5.2 SAP Gardener Deployment Scenario

2.6 Integration into the Application Lifecycle

Chapter 3

State of the Art

3.1 Common Approach of the Industry

3.2 Current Approach at SAP Gardener

3.3 Problems of Existing Approaches

Chapter 4

System Design

4.1 Data Model

4.2 Database

4.3 API

Tool agnostic, how does the api look like to abstract away from different tools

Chapter 5

Implementation

Chapter 6

Results

6.1 Evaluation of the Security and Compliance Data Lake

6.2 Conclusion and Outlook