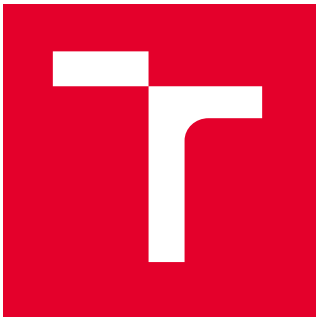


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ

DEPARTMENT OF TELECOMMUNICATIONS

NÁSTROJ PRO SPRÁVU METADAT SBOM

A SOFTWARE BILL OF MATERIAL MANAGEMENT TOOL

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — Bc. Marek Szymutko

AUTHOR

VEDOUCÍ PRÁCE — RNDr. Ing. Pavel Šeda, Ph.D.

SUPERVISOR

BRNO 2025

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní program **Informační bezpečnost**

Ústav telekomunikací

Student: Bc. Marek Szymutko ***ID:*** 231285

Ročník: 2 ***Akademický rok:*** 2024/25

NÁZEV TÉMATU:

Nástroj pro správu metadat SBOM

POKYNY PRO VYPRACOVÁNÍ:

Práce bude zaměřena na práci s metadaty SBOM (Software Bill of Material), které obsahují informace o kompozici software. V rámci teoretické části student objasní význam těchto metadat, jejich legislativní původ a souvislost s metadaty VEX (Vulnerability Exploitability Exchange). Dále pak vyjmenuje a přiblíží možné způsoby specifikace metadat SBOM a VEX. Praktická část diplomové práce se zaměří na vytvoření nástroje pro validaci, známkování a překlad metadat SBOM podle příkladů zveřejněných společností Red Hat Inc. Tento nástroj musí být využitelný v automatizovaných procesech.

DOPORUČENÁ LITERATURA:

[1] Yousefnezhad, N., Costin, A. (2024). Understanding SBOMs in Real-World Systems – A Practical DevOps/SecOps Perspective. In: Shishkov, B. (eds) Business Modeling and Software Design. BMSD 2024. Lecture Notes in Business Information Processing, vol 523. Springer, Cham. https://doi.org/10.1007/978-3-031-64073-5_20

[2] Arora, Arushi, Wright, Virginia L., and Garman, Christina. SoK: A Framework for and Analysis of Software Bill of Materials Tools. United States: N. p., 2022. Web. doi:10.2172/2204407.

Termín zadání: 10.2.2025 ***Termín odevzdání:*** 27.5.2025

Vedoucí práce: RNDr. Ing. Pavel Šeda, Ph.D.

Konzultant: Ing. Aleš Raszka

prof. Ing. Jan Hajný, Ph.D.
předseda rady studijního programu

ABSTRAKT

Metadata SBOM jsou poměrně novým problémem v oblasti DevSecOps. V USA je vydávání těchto metadat vyžadováno vydáním prezidentského nařízení č. 14028, v Evropské Unii je pak jejich vydávání nutné až od roku 2027. Pro vydávání těchto metadat však existuje více standardů a je možné využít více metodologií při jejich tvorbě. Společnost Red Hat tedy vydala vlastní harmonizující návody, na jejichž základě má být dosaženo vydávání jednotných a v praxi využitelných dokumentů SBOM.

Práce se věnuje legislativnímu významu metadat SBOM, jejich úloze při ochraně dodavatelského řetězce a vysvětluje jejich souvislost s metadaty VEX. V rámci práce byla vytvořena aplikace umožňující známkování kvality dokumentů SBOM podle na základě doporučení od společnosti Red Hat a také nástroj pro překlad mezi dvěma nejvíce užívanými formáty metadat SBOM. K tomuto účelu byl v rámci práce vytvořen i jazyk pro vyhledávání v datových strukturách JSON.

Aplikace byla testována, včetně testů na datech poskytnutých společností Red Hat. Počítá se s nasazením této aplikace do CI/CD infrastruktury společnosti Red Hat.

KLÍČOVÁ SLOVA

Bezpečnost, CI/CD, DevSecOps, dodavatelský řetězec, EU, legislativa, Red Hat, SBOM, USA, VEX

ABSTRACT

SBOMs are a new problem in the field of DevSecOps. In the USA, the creation of SBOMs is mandatory per Executive Order 14028, whereas in the European Union issuing SBOMs will be necessary only after 2027. There are multiple standards for SBOM creation and various methodologies can be used for SBOM creation. The company Red Hat issued SBOM guidelines that aim to ensure accurate and useful SBOMs.

This thesis aims to explain SBOM legislation, the role of these documents in the software supply chain, their correlation to metadata VEX. In the scope of this thesis, an application for SBOM grading and format conversion was created based on the Red Hat guidelines. To achieve this functionality a custom query language for searching in JSON files was created.

The application has been thoroughly tested, including testing on data provided by Red Hat. It is expected to include the developed application in the CI/CD infrastructure of Red Hat.

KEYWORDS

CI/CD, DevSecOps, EU, legislation, Red Hat, SBOM, security, supply chain, USA, VEX

SZYMUTKO, Marek. *Nástroj pro správu metadat SBOM*. Diplomová práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2025. Vedoucí práce: RNDr. Ing. Pavel Šeda, Ph.D.

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Bc. Marek Szymutko
VUT ID autora:	231285
Typ práce:	Diplomová práce
Akademický rok:	2024/25
Téma závěrečné práce:	Nástroj pro správu metadat SBOM

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora*

*Autor podepisuje pouze v tištěné verzi.

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu semestrální práce panu RNDr. Ing. Pavlu Šedovi, Ph.D. za odborné vedení, ochotu, trpělivost i vstřícnost.
Také bych rád poděkoval svému konzultantu ve společnosti Red Hat, Ing. Aleši Raszkovi za ochotu, vstřícnost, nápady na zlepšení, zpětnou vazbu k implementaci, zviditelňování tohoto projektu v rámci organizace Red Hat i podněty k testování.
Dále bych chtěl poděkovat Mgr. Martinu Prpičovi z Product Security společnosti Red Hat za námět k práci a zpětnou vazbu k implementaci.

Obsah

Úvod

Tato diplomová práce se zabývá metadaty *Software Bill of Material* (SBOM). Jedná se o poměrně nový fenomén v informačních technologiích zachycující informace o komponentech software. Pro sjednocení formy těchto metadat vydala společnost Red Hat doporučení, jak mají být důležité informace v dokumentech SBOM reprezentovány. Tato doporučení však byla vydána pouze v psané formě a jejich implementace do užitečného nástroje je předmětem právě této práce. Doporučení obsahuje dva formáty metadat SBOM, jimiž jsou SPDX 2.3 a CycloneDX 1.6. Tyto formáty musí být aplikací podporovány.

Výstupem praktické části práce musí být nástroj pro známkování a konverzi dokumentů SBOM na základě dodržování dříve zmíněných doporučení. Tento nástroj musí být využitelný v automatizovaných procesech *Continuous Integration and Continuous Delivery* (CI/CD). To znamená, že je nutné zajištění snadné instalace projektu a zároveň výstup kompatibilní se standardními datovými formáty. Těchto cílů bylo dosaženo implementací nástroje v programovacím jazyce Python a jeho následné zveřejnění na platformě `PyPI`. Nástroj podporuje výstup ve formátech JSON, YAML nebo Markdow.

Téměř všechny zdroje informací o metadatech SBOM jsou vydány pouze v angličtině. Proto je důležité v úvodu práce definovat pojmy, které budou pro popis metadat i pro implementaci praktické části užity. Slovo „pole“ je užíváno ve významu ekvivalentnímu k anglickému výrazu *field* (jinak také „atribut“). Pro překlad slov *array* nebo *list* je zde užíván výraz „seznam“. Výrazy „metadata SBOM“, „kusovníky software“ či „dokumenty SBOM“ jsou zde užity jako synonyma.

V první kapitole této práce jsou zahrnuta veškerá důležitá fakta a souvislosti o dokumentech SBOM včetně jejich legislativního významu a souvislosti s metadaty VEX. V této kapitole jsou také nastíněny potřeby společnosti Red Hat a popis některých již existujících nástrojů pro správu metadat SBOM.

V kapitolách dvě, tři a čtyři je popsán implementovaný nástroj. Druhá kapitola se věnuje popisu jádra aplikace, třetí nastiňuje aplikaci hodnocení kvality metadat a ve čtvrté kapitole je popsána aplikace konverze mezi podporovanými formáty.

Kapitola pátá se věnuje testování aplikace a možnostem jejího nasazení. Je zde popsáno testování na datech poskytnutých společností Red Hat, profilování aplikace z hlediska časové i paměťové náročnosti a srovnání s existujícími nástroji včetně vysvětlení nutnosti vývoje tohoto nástroje.

V přílohách jsou pak nastíněny některé standardy pro vytváření souborů SBOM. Nejvíce prostoru je zde věnováno nejjednodušší z moderních SBOM specifikací, jíž je SPDX 2.3, která je také ve společnosti Red Hat nejvíce využívána.

1 SBOM

Metadata SBOM slouží jako soupis všech komponentů a závislostí, které obsahuje popisovaný software [1]. Jedná se o strojově čitelná data, respektující vzájemnou hierarchii prvků a popisující vztahy mezi nimi. Využívanými datovými formáty jsou například JSON, YAML nebo XML [2].

Jejich význam spočívá ve zjednodušení přístupu k informacím o užívaném software. Jelikož je dnes časté využívání knihoven, které nemusely být napsány koncovým vydavatelem, roste také bezpečnostní riziko, které tyto závislosti mohou představovat [3]. Tyto závislosti navíc mohou být dodávány v binární podobě, což znemožňuje či znesnadňuje jejich čtení člověkem. Pokud je však celý strom závislostí zaznamenán se všemi potřebnými informacemi v souboru SBOM, je přístup k informacím snadný.

SBOM metadata jsou standardizovaná, ale neexistuje pro ně pouze jediný standard [2]. Pro jejich vytváření se využívá metodologií, jež vznikly za různými účely z pohnutek různých institucí [4, 5, 6]. Ani tyto specifikace však nebývají naprosto rigidní a dovolují určitou volnost ve formátu a obsahu dat.

1.1 Legislativní význam metadat SBOM

Dokumenty SBOM jsou v dnešní době užívány zejména ke správě licencí u *Free and Open Source Software* (FOSS) a také k zajištění bezpečnosti software proti útoku na dodavatelský řetězec. Obě tato využití mají legislativní dopad, který bude dále vysvětlen v následujících částech práce.

1.1.1 Správa licencí

Historicky byla SBOM metadata vytvářena jako nástroj pro kontrolu licenčních podmínek u otevřeného software [6]. Z právního hlediska je totiž nutné při využití programu dodržet všechna licenční ustanovení včetně ustanovení z licencí programových závislostí, což vyplývá z autorského zákona [7]. Aby bylo možné všechna licenční ustanovení dodržet, je nutné znát soupis všech závislostí a jejich licencí, nebo ho sestavit pomocí automatizace [8]. Při binární distribuci balíčků je prakticky nemožné dohledat licenční podmínky přímo v datech programu, je tedy vhodné připojit k němu další data s licenčními ujednáními. Standard *System Package Data Exchange* (SPDX), (dále v části 1.3.1,) byl vytvořen jako řešení tohoto problému [6].

V dnešní době jsou k tomuto účelu však vhodné i další standardy, mezi něž patří CycloneDX (popsán v části 1.3.3) nebo *SoftWare IDentification* (SWID) (nastíněn v sekci 1.3.4).

Ačkoli metadata SBOM nejsou výsledky tvůrčí činnosti autora (což vyžaduje český autorský zákon, aby se jednalo o autorské dílo [7]), bylo by možné uvažovat, zdali nenaplňují znaky databáze podle Směrnice Evropského parlamentu a Rady 96/9/ES [9]. I když i tento závěr je velice nepravděpodobný, autoři formátu SPDX předešli všem problémům způsobeným právy duševního vlastnictví tím, že vyžadují deklaraci licence CC0 1.0 [10] v každém dokumentu. Tato licence se vztahuje k datům obsaženým v dokumentu SBOM, nikoliv však k popisovanému softwaru. Pokud by dokument tuto deklaraci neobsahoval, nesmělo by pro něj být užito označení formátu SPDX [11].

1.1.2 Bezpečnost

V dnešní době však SBOM metadata získávají také další význam. Díky soupisu všech komponentů software lze díky nim snadněji sledovat zranitelnosti v programu obsažené, a tím předcházet útokům na softwarový dodavatelský řetězec (anglicky *Supply Chain Attack*) [5, 12].

Tento útok spočívá v zanesení zranitelnosti či „zadních vrátek“ (anglicky *backdoor*) do některé závislosti využívané v jiných projektech [12]. Například na začátku roku 2024 byl do nástroje `xz` zanesen kód umožňující útočníkovi přístup na systémy, které tento nástroj využívají [13]. Knihovna `xz` umožňuje kompresi dat a je využívána ve velké škále dalších projektů i operačních systémů rodiny Linux [13]. Pro zjištění, zda je zranitelný software obsažen v distribuovaném produktu, pomáhají právě metadata SBOM [14].

Legislativní význam v USA

Ve Spojených státech amerických byl vydáním prezidentského nařízení č. 14028 (anglicky *Executive Order 14028*) tento význam zanesen i do oficiálních pravidel, týkajícího se tzv. „kritického software“ využívaného americkou federální vládou. Kritický software je v tomto dokumentu specifikován jako software vykonávající funkce nutné k ustanovení důvěry [15]. Přesné specifikace a implementace tento dokument však neobsahuje, jen ukládá povinnosti jiným subjektům, aby tyto specifikace vytvořily, či zavedly již existující standardy. Je zde však specifikováno, že každý dodavatel kritického software musí k němu dodávat SBOM metadata, či tato metadata zveřejnit na webu [15, 16].

V reakci na toto ustanovení reagoval americký úřad *National Telecommunication and Information Administration* (NTIA) vydáním dokumentu popisující minimální náležitosti metadat SBOM (přehled náležitostí je uveden v tab. 1.1), další rozšiřující nepovinná pole a pole vhodná k budoucí standardizaci [17, 18].

Tab. 1.1: Povinná pole podle NTIA [18]

Datové pole	**Popis**
Název dodavatele	Název entity, která vytváří, definuje a identifikuje komponent.
Název komponentu	Pojmenování vytvořené původním vydavatelem.
Verze komponentu	Identifikátor umožňující specifikovat změny software oproti dříve identifikované verzi.
Další unikátní identifikátory	Další identifikátory určené k jeho identifikaci nebo nalezení v databázi.
Vztahy závislostí	Charakterizují vztah, že komponent X je součástí software Y.
Autor SBOM	Název entity, která vytvořila SBOM data k tomuto komponentu.
Časové razítko	Záznam data a času vytvoření SBOM.

Legislativní význam v Evropě

Spojené státy americké nejsou jediným celkem usilujícím o zakotvení významu metadat SBOM přímo do legislativních předpisů. Evropská unie vydala „Akt o kybernetické odolnosti“ (nařízení EU 2024/2847), který však nabyde účinnost až ke konci roku 2027 [19, 20, 21]. Metadata SBOM jsou v tomto nařízení označena pojmem „softwarový kusovník“ a musí je dodávat každý výrobce „produktů s digitálními prvky“ uvádějící tyto produkty na trh [20]. Dokumenty SBOM musí být dodávány „v běžně používaném strojově čitelném formátu, který obsahuje přinejmenším nejdůležitější závislosti produktu“ [20]. Přesné standardy zde tedy nejsou specifikovány.

Pojem „produkt s digitálními prvky“ označuje jak hardware, tak software [19], a nutnost vytváření metadat SBOM je v dokumentu explicitně zmíněna i pod touto zkratkou [20]. Výrobci mají uloženo tato metadata vytvářet a poskytovat za účelem zvýšení kyberbezpečnosti, avšak nejsou nuceni dokumenty SBOM zveřejňovat [20].

Německo na základě připravovaného nařízení Evropské unie vydalo technickou normu BSI TR-03183, která stanovuje, že SBOM metadata musí splňovat následující parametry: Jedná se o soubor formátu JSON nebo XML, podle standardu CycloneDX ve verzi 1.5 nebo vyšší nebo SPDX ve verzi 2.2.1 nebo vyšší [22].

V České republice však prozatím nebyl vydán žádný závazný dokument, který by nutil vydavatele software dodávat SBOM metadata, nezávisle na účelu software. Když nabyde účinnost Akt o kybernetické odolnosti, budou však muset výrobci produktů s digitálními prvky vydávat dokumenty SBOM od konce roku 2027 i na českém trhu [21].

1.2 Typy SBOM podle NTIA

Americká agentura NTIA definovala šest druhů SBOM dokumentů [23]. Tyto druhy se odvíjejí od stádia komponentu, který popisují. Podle stádia mohou být dostupné různé informace, což ovlivňuje obsáhlost dokumentu. Přehled těchto typů včetně základních informací je uveden v tabulce 1.2.

Tab. 1.2: Typy SBOM podle NTIA[23]

Typ SBOM	**Popis**
Design	SBOM pro plánovaný projekt. Tvoří se ze specifikace nebo konceptu.
Source	Dokument SBOM vytvořený ze zdrojového kódu a závislostí.
Build	SBOM vzniklý při sestavování komponentu k vydání. Může vycházet i z již sestavených souborů.
Analyzed	SBOM, který vznikl analýzou sestaveného komponentu. Nejčastěji vytvořen třetí stranou.
Deployed	SBOM, který může obsahovat i data získaná nasazením komponentu do systému. Může zahrnovat i vztahy k dalším komponentům v systému.
Runtime	SBOM, který zahrnuje i dynamické vazby vzniklé za běhu komponentu.

Z této tabulky vyplývá, že dokumenty SBOM pro samostatné aplikace je vhodné vytvářet ve fázi sestavení aplikace, aby byla tato metadata přesná a podrobná. SBOM tohoto charakteru je přenositelný (neobsahuje informace o platformě), ale zároveň lze pro jeho vytvoření využít všechna fakta, která nemusí být známá v pozdějších fázích životního cyklu aplikace (například zdrojové identifikátory *Uniform Resource Locator* (URL) závislostí nemusí být již po sestavení aplikace dostupné).

1.3 Vybrané standardy

V této části práce jsou nastíněny hlavní využívané standardy pro tvorbu metadat SBOM. Pro jednoduchost jsou zde uvedeny pouze nejvýznamnější standardy. Většina z těchto standardů navíc podporuje více platných formátů, které se od sebe liší jak strukturou dokumentu, tak i názvy jednotlivých polí. Pro přiblížení všech specifikací byl vybrán přístup popisu formátu JSON, který je podporovaný všemi zde popsanými standardy a je využit i společností Red Hat.

1.3.1 SPDX 2.X

Specifikace SPDX je otevřeným standardem patřící nadaci Linux (anglicky *the Linux Foundation*) [4]. Jeho první verze (SPDX 1.0) vyšla již v roce 2011, zatímco nejnovější verze (SPDX 3.0) byla vydána v první polovině roku 2024 [4]. Jak již bylo zmíněno dříve (v sekci 1.1.1), prvotním účelem SPDX bylo ulehčení orientace v licencích otevřeného software, avšak novější verze tohoto standardu jsou využívané i pro bezpečnostní účely.

SPDX verze 2.2.1 existuje i jako mezinárodní standard ISO/IEC 5962:2021 [24], jiné verze však prozatím takto formálně standardizovány nebyly. Jakákoli verze standardu je však dostupná přímo ze stránek projektu [4, 11, 25].

Každá verze standardu se mírně odlišuje deklarovanými poli. SPDX dokument ve verzi 2.3 definuje 8 sekcí, z nichž pouze jediná, sekce s informacemi o vytvoření SPDX dokumentu, je povinná. Výčet podporovaných sekcí dokumentu je následující [11]:

- Informace o vytvoření SPDX dokumentu
- Informace o balíčcích
- Informace o souborech
- Informace o útržcích
- Další detekované informace o licencování
- Vztahy jednotlivých částí dokumentu
- Informace o anotacích
- Informace o recenzích[1]

Jednotlivá pole v každé sekci jsou pak dále přiblížena v příloze A.1, v tabulkách A.1 až A.8. I z tohoto krátkého popisu lze však vyčíst, že se jedná o velice flexibilní specifikaci, což může znesnadňovat práci s daty, na druhou stranu však umožňuje vytvářet metadata popisující velkou škálu software.

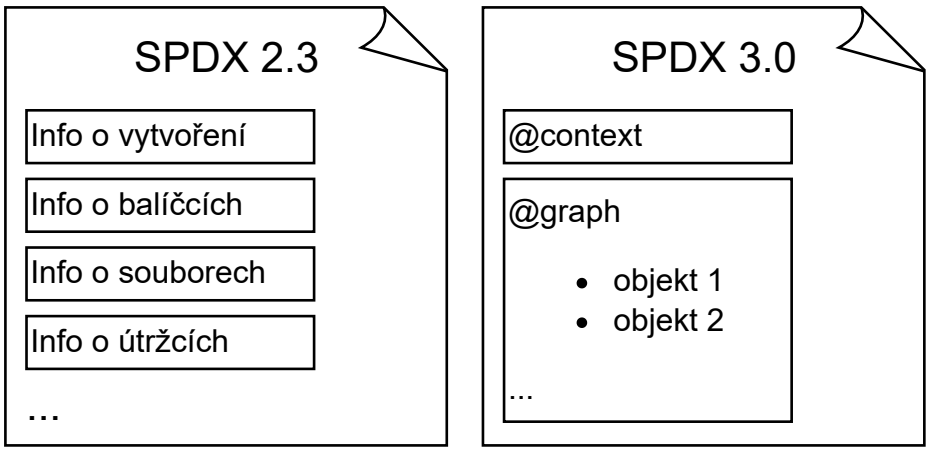
1.3.2 SPDX 3.0

Tento standard, vydaný v roce 2024 [4, 26], využívá k sestavení dokumentu specifikaci *JSON Linked Data* (JSON-LD) [25, 27] namísto jednoduchého souboru JSON, jako tomu bylo u verze SPDX 2.3. JSON-LD specifikuje formát JSON dat, která jsou lépe čitelná pro stroje díky jednoznačnosti klíčů dokumentu. Namísto jediného slova užívají unikátní identifikátory IRI[2], které jsou delší než názvy polí v předchozích verzích. Pro zjednodušení čitelnosti člověkem jsou pak tyto identifikátory zastoupeny předem definovanými zkrácenými verzemi. Tyto zkrácené identifikátory jsou uloženy v části dokumentu označené klíčem `@context`.

[1] Tato část je zastaralá a neměla by se již využívat.

[2] *Internationalized Resource Identifier* (IRI) je modernizovaným standardem *Uniform Resource Identifier* (URI).

Dalším klíčem v dokumentu je pak `@graph`, který obsahuje informace o všech ostatních prvcích dokumentu [25]. K tomuto klíči je přiřazen seznam obsahující prvky, které jsou součástí dokumentu. Rozdíly oproti SPDX ve verzi 2.3 jsou ilustrovány na obrázku 1.1.



Obr. 1.1: Porovnání verzí specifikace SPDX

Specifikace v této verzi je velice komplexní, a proto byla vydána i specifikace s názvem *SPDX Lite*, která má za úkol zjednodušit proces vytváření SBOM dokumentů v dostatečné kvalitě bez nutnosti užití všech specifikovaných polí či atributů. Tato odlehčená specifikace vyjadřuje, které objekty či atributy musí být v dokumentu přítomny a které jsou doporučené. Zbytek objektů a atributů je pouze dobrovolný. Základní pravidla pro tvorbu dokumentů SPDX 3.0 Lite jsou uvedena v příloze A.2.

1.3.3 CycloneDX

Standard CycloneDX vznikl v roce 2017 pod organizací OWASP [5]. Kromě SBOM umožňuje vytváření více typů *Bill of Material* (BOM), např. *Hardware Bill of Material* (HBOM). Primární účel této specifikace je identifikace kybernetického nebezpečí, lze ho ale použít i pro účely dodržování licenčních podmínek [28]. Je plně kompatibilní s prezidentským nařízením č. 14028 (viz část 1.1) a umožňuje popisovat mnohem komplexnější informace, než jsou tímto nařízením vyžadovány. Nejnovější verze této specifikace je CycloneDX 1.6, který byl vydán v roce 2024 [5].

Struktura standardu je podobná specifikaci SPDX 2.3, jelikož rozlišuje určité sekce dokumentu, přičemž každá z nich má jisté náležitosti [29]. Na rozdíl od specifikace SPDX 2.3 je však mnohem flexibilnější, tedy i složitější. Umožňuje například

i opakované vnoření daných typů objektů, což může být překážkou pro snadné vyhledávání v těchto dokumentech. Celkový přehled polí dokumentu CycloneDX 1.6 je uveden v tabulce A.13. Z důvodu složitosti je však tento přehled pouze zevrubný.

1.3.4 SWID

Formát SWID je standardem také známým pod označením ISO/IEC 19770-2:2015 [30]. SWID vznikl v roce 2012 a v roce 2015 byl tento standard upraven [31]. Slouží pro vytváření metadat zvané *tagy*, které obsahují informace o závislostech software. Ačkoliv je přístup k této ISO normě zpoplatněn [30], lze zdarma využít specifikaci vydanou úřadem *National Institute of Standards and Technology* (NIST) [32].

Specifikace pro metadata SWID existuje pouze ve formátu XML [32]. Ve společnosti Red Hat se tento standard nevyužívá vůbec [33], z tohoto důvodu se této specifikaci nebude diplomová práce dále věnovat.

1.4 Identifikátory užívané v dokumentech

Dokumenty SBOM obsahují informace o velkém množství komponentů. V lidské komunikaci by pro identifikaci byl nejspíše pro každý komponent užit název komponentu, avšak tento přístup by vytvářel nejistoty. Pro upřesnění je vhodné doplnit název také o využitou verzi tohoto software, či architekturu procesorů, pro které byl program vytvořen. Identifikátory využívané v dokumentech SBOM mají za úkol přesně identifikovat každý komponent v jednotném formátu. V následujících částech práce budou představeny některé z těchto identifikátorů.

1.4.1 PURL

Package URL (PURL) slouží jako jednoznačný identifikátor balíčku ve všeobecně známém formátu URL. Stejná zkratka je využívána i ve významu *Persistent URL*, avšak užití těchto dvou identifikátorů se liší. PURL užívané v dokumentech SBOM jsou definovány v rámci specifikace SPDX, v příloze E [34]. Formát těchto identifikátorů je uveden ve výpisu 1.1, pole zde byla pro přehlednost barevně odlišena.

Výpis 1.1: Formát identifikátoru PURL podle SPDX [34]

```
schéma:typ/jmennýProstor/název@verze?kvalifikátory#cesta
```

Pole `schéma` obsahuje vždy statický řetězec „`pkg`“. V poli `typ` je vyjádřen typ balíčku, například hodnota `oci` značí obraz kontejneru [33]. Část `jmennýProstor` je závislá na typu komponentu, může se například jednat o identifikátor vydavatele

balíčku. Kvalifikátory další informace ve formátu klíčů a hodnot. Příkladem kvalifikátoru je řetězec „`arch=x86_64`“, který vyjadřuje architekturu procesorů, pro které je balíček vytvořen. Při využití více kvalifikátorů se oddělují znakem „`&`“. Pole `cesta` může obsahovat cestu k souboru v balíčku [34].

Striktně povinná pole jsou pouze `schéma`, `typ` a `jmenný prostor`. Ostatní pole umožňují identifikovat přesný balíček, jeho jedinou verzi či soubor. Identifikátor je konformní se specifikací URL, avšak nesmí obsahovat uživatelská jména či hesla, stanice ani síťové porty [34].

1.4.2 CPE

Identifikátor *Common Platform Enumeration* (CPE) slouží k identifikaci tříd software i hardware a je udržován organizací NIST [35]. Cílem tohoto formátu není identifikovat jednotlivé instalace. Hlavním účelem tohoto identifikátoru je správa zranitelností, jelikož má za cíl usnadňovat identifikaci virtuálních tříd software (třídu si lze představit jako množinu instancí se společnými znaky, do třídy patří například všechny instalace software se specifickou konfigurací) [35].

Jedná se o poměrně rozsáhlou specifikaci, která již má mnoho verzí, z nichž nejnovější je (v čase psaní práce) verze CPE 2.3. Specifikace CPE obsahuje čtyři části; jmennou část, sekci párování jmen, slovník a jazyk aplikovatelnosti [35].

Jmenná část specifikace CPE definuje tzv. *Well-Formed Name* (WFN) jako abstraktní mapu klíčů a hodnot, která unikátně identifikují software či hardware[36]. Popisuje také, jak tuto abstraktní mapu přeložit do strojově čitelného identifikátoru CPE. WFN obsahuje deset atributů, mezi něž patří například identifikátor popisované součásti, vydavatel, produkt, verze a další [36]. Identifikátor CPE verze 2.3 je uveden ve výpisu 1.2. V tomto výpisu je pevně dané pouze pole „`cpe`“, ostatní atributy drží hodnoty využité k identifikaci komponentu.

Výpis 1.2: Formát identifikátoru CPE podle NIST [35]

```
cpe:cpeVerze:část:poskytovatel:produkt:verze:update:edice:jazyk:
softwareEdice:cílovýSoftware:cílovýHardware:další
```

Sekce párování jmen popisuje porovnávání dvou identifikátorů CPE a popis množinových operací, které vzniknou kombinováním více těchto identifikátorů. Díky tomu je možné ku příkladu zjistit, zdali je instalovaný software v množině software zranitelného nově objevenou slabinou [37].

Slovníková část umožňuje identifikátorům CPE přiřazovat metadata [38]. Každý identifikátor slovníku specifikuje pouze jedinou třídu software. Oficiální slovník je udržován organizací NIST a je spojen s *National Vulnerablity Database* (NVD) [38].

Poslední část specifikace popisuje jazyk aplikovatelnosti, což je funkcionalita umožňující popisovat komplexní platformy informačních technologií. Takto lze identifikátory sjednocovat do skupin, pro které je poté možno vybírat aplikovatelná doporučení či pravidla [39].

Společně tyto specifikace tvoří rámec využívaný převážně pro správu zranitelností, právě díky využití s databází slabin NVD. K tomuto účelu jsou identifikátory CPE využity i v dokumentech SBOM.

1.4.3 Identifikátory specifické pro formáty SBOM

Dříve popsané identifikátory jsou všeobecně uznávané a nezávislé na konkrétním formátu dokumentu SBOM. Existují však také identifikátory využívané pouze pro popis vztahů mezi jednotlivými komponentami zmíněnými v kusovníku. Identifikátory využívané v dříve zmíněných standardech zde budou přiblíženy.

SPDXID

Tento identifikátor je využíván v kusovnících podle standardu SPDX. Mezi verzemi SPDX 2.3 a 3.0 došlo u tohoto identifikátoru ke změnám. Ve starší verzi bylo nutné, aby každý identifikátor začínal řetězcem „`SPDXRef-`“ [11]. Každý z těchto identifikátorů byl využitelný pouze v dokumentu, ve kterém byl vytvořen, a tedy musel být unikátní pouze v rámci tohoto dokumentu.

Tyto identifikátory se využívají pro zapsání vztahů mezi jednotlivými prvky dokumentu. Speciální roli má identifikátor SPDXID ve tvaru „`SPDXRef-DOCUMENT`“, který musí být obsažen v každém SBOM formátu SPDX 2.3. Tento identifikátor popisuje dokument SBOM jako takový a musí být obsažen ve vztahu typu `DESCRIBES` či `DESCRIBED_BY` tak, aby bylo jasné, který element je popisován dokumentem.

U nejnovější verze specifikace SPDX se již jedná o IRI, tudíž se může odkazovat i na objekty mimo aktuální dokument SBOM [25]. Lze samozřejmě využít ostatní identifikátory konformní s formátem URI, jako jsou již dříve zmíněné identifikátory PURL či CPE.

BOM-Ref

Identifikátor BOM-ref využívaný ve standardu CycloneDX je také využitelný pouze pro účely navigace v dokumentu, není tedy nutné zajišťovat unikátnost mimo dokument. Specifikace neříká, jaký přesný formát by měl tento identifikátor nabývat, je v ní pouze uvedeno, že by neměl začínat frází „`urn:cdx:`“ [29], aby nedocházelo ke kolizím s funkcionalitou *BOM-Link* zmíněnou v následující části práce. Lze tedy pro tento identifikátor využít hodnotu z pole pro PURL či CPE.

BOM-Link

BOM-Link se také využívá v dokumentech podle specifikace CycloneDX a slouží pro reference na jiné dokumenty. Každý dokument SBOM podle této specifikace má mít unikátní sériové číslo, na které lze odkázat právě pomocí funkcionality BOM-Link [40].

Formát tohoto identifikátoru je uveden ve výpisu 1.3, jednotlivá pole jsou barevně odlišena. První dvě pole tohoto identifikátoru jsou stálá. Atribut `sériovéČíslo` odkazuje na identifikátor dokumentu SBOM, pole `bom-ref` pak odkazuje na lokální identifikátor BOM-Ref ve zmíněném dokumentu. Také je možné specifikovat verzi dokumentu.

Výpis 1.3: Formát identifikátoru BOM-Link podle OWASP [40]

```
urn:cdx:sériovéČíslo/verze#bom-ref
```

1.4.4 Identifikátory licencí

Aby metadata SBOM mohla sloužit k účelům dodržování licenčních ustanovení (jak bylo zmíněno v části 1.1.1), musí také obsahovat informace o licencích jednotlivých komponent. Kdyby však kusovníky obsahovaly veškerá znění licenčních ustanovení, musely by obsahovat velké množství textu. Proto vznikl pod organizací SPDX seznam licencí, jejichž názvy jsou zkráceny do jednoduchých a krátkých výrazů [41].

Tento seznam je využíván nejen v dokumentech SBOM podle specifikace SPDX. Je využíván v dokumentech CycloneDX či jej lze referencovat při licencování projektů v programovacím jazyce Python [42]. Seznam obsahuje známé zkratky, mezi nimiž je například i licence MIT [43], jež byla využita při zveřejnění praktické části.

1.5 VEX

Vulnerability Exploitability eXchange (VEX) dokument slouží ke zhodnocení stavu zranitelností v dané aplikaci [44]. Jedná se o speciální případ bezpečnostního doporučení (anglicky *security advisory*), které je strojově čitelné a snadno spojitelné se správným dokumentem SBOM. Tyto vlastnosti mají zaručit snadnou automatizaci ve světě bezpečnostních zranitelností, což zahrnuje rychlé zjištění, sledování a odstranění slabin [44].

Hlavní účel těchto dokumentů je vyrozumění o stavu daného digitálního produktu [44, 45]. Existují například situace, kdy je díky dokumentům SBOM zjištěna zranitelnost v některé ze závislostí produktu, což znamená, že ovlivněn může být

i celý produkt. V některých případech však může být celkový produkt zranitelností neovlivněn, i když jeho závislosti slabiny obsahují [45] (např. vstupy některé knihovny mohou být špatně ověřovány, nejsou však v produktu vystaveny útočníkům). V takovém případě je tedy vhodné vyrozumět operátory produktu o tom, že produkt není zranitelností ovlivněn.

Pro úplné porozumění zranitelnosti je tedy nutné spojit informace o zranitelnosti *Common Vulnerabilities and Exposures* (CVE), informace o zranitelných komponentech (např. CPE) a informace o zneužitelnosti (tedy VEX) [1].

Obdobně jako u dokumentů SBOM, tak i pro dokumenty VEX bylo vydáno doporučení o jejich minimálních náležitostech. Tyto náležitosti, zveřejněné agenturou *Cybersecurity & Infrastructure Security Agency* (CISA), jsou pro přehlednost uvedeny v tabulce 1.3. Agentura CISA také v roce 2023 specifikovala přesnější strukturu těchto dokumentů, která zachycuje i kardinalitu a zapouzdření dat [46].

Tab. 1.3: Povinná pole dokumentu VEX podle CISA [44]

Datové pole	**Popis**
Metadata	Formát VEX dokumentu, identifikátor dokumentu, autor, role autora, časové razítko.
Detaily produktu	Identifikace produktu, obdobně jako u dokumentů SBOM.
Detaily o zranitelnosti	Identifikátor zranitelnosti, jako například CVE a popis zranitelnosti.
Detaly o stavu produktu	Musí se jednat o jednu z hodnot *NOT AFFECTED* (zranitelnost není v tomto produktu zneužitelná), *AFFECTED* (je potřeba vykonat akce k odstranění slabiny), *FIXED* (produkt obsahuje opravu zranitelnosti) nebo *UNDER INVESTIGATION* (zatím není známo, zda je produkt zranitelný).
Akce	Pole, které musí mít dokumenty označené stavem *AFFECTED*, jedná se o kroky k odstranění zneužitelnosti slabiny.
Prohlášení o dopadu	Pole, které musí mít dokumenty označené stavem *NOT AFFECTED*, dále vysvětluje detaily vztahu zranitelnosti a produktu.

Formáty VEX

Metadata VEX je možné buď začlenit do dokumentů SBOM (pokud je to umožněno danou specifikací), nebo je možné je distribuovat separátně. Z těchto dvou přístupů je vhodnější využít samostatné dokumentyVEX, jelikož na rozdíl od souborů SBOM jsou údaje v dokumentech VEX méně stálá [47].

Začlenit metadata VEX do souborů SBOM umožňuje specifikace CycloneDX [47]. Pro samostatné soubory VEX existuje specifikace OpenVEX [48], která je založená na požadavcích agentury CISA z roku 2023. VEX dokumenty jsou také specifikovány v rámci *Common Security Advisory Framework* (CSAF) verze 2.0 [49].

1.6 Potřeby společnosti Red Hat

Problematika SBOM je poměrně komplexní, jak ale vyplývá z teoretické části, je nutné začít jejich adopci co nejdříve, aby byly splněna všechna legislativní pravidla týkající se kyberbezpečnosti. Momentální stav umožňuje využívat mnoho specifikací, které jsou samy velice flexibilní, je tedy poměrně náročné tvorbu SBOM sjednotit.

Ve společnosti Red Hat navíc existuje více způsobů, kterými může být komponent zveřejněn konečným zákazníkům. Všechny tyto metody implementují nějakou formu tvorby dokumentů SBOM. Aby vytvářená metadata mohla mít pozitivní dopad, musí existovat jednotná pravidla, jež definují, které informace musí SBOM obsahovat. Nestačí však pouze minimální pole specifikovaná organizací (viz tabulku 1.1), pokud má SBOM nést veškeré důležité informace.

Z tohoto důvodu se Red Hat Product Security zabývá definicí kvalitního dokumentu SBOM [33]. V rámci této definice jsou specifikovány také pojmy *komponent* a *produkt*. Produkt je konečný výsledek poskytnutý zákazníkovi, kdežto komponenty jsou jeho dílčí části.

Red Hat komponenty bývají vydávány ve více podobách. Těmi jsou *downstream* (komponent distribuovaný zákazníkovi), *upstream* (otevřený projekt, který je základem distribuovaného komponentu) a *midstream* (otevřený projekt rozšířený z *upstream* projektu) [33]. Tyto typy komponentů na sebe mají určité vazby, které je nutné reflektovat v dokumentu SBOM, aby bylo možné sledovat dodavatelský řetězec.

1.6.1 Typy SBOM podle společnosti Red Hat

Red Hat využívá dělbu SBOM typů do více vrstev, než specifikuje agentura NTIA (viz tabulku 1.2). Nedělí typy pouze podle fáze komponentu, ale také podle šíře a hloubky obsažených dat. Typy SBOM podle společnosti Red Hat jsou uvedeny v tabulce 1.4.

Tab. 1.4: Typy SBOM vytvářené společností Red Hat [33]

Typ SBOM	**Popis**
Build-time	Vytvářený při sestavení software, koresponduje s typem *build* podle NTIA.
Release-time	Vytvářený při zveřejnění software, kdy je známá například i URL repozitářů nebo registrů, kde je sestavený komponent k sestavení. Také koresponduje s typem *build* podle NTIA.
Component-level	SBOM jediného komponentu.
Product-level	SBOM celého produktu.
Shallow	Obsahuje pouze komponenty užité k sestavení souborů.
Complete	Obsahuje veškeré dostupné informace o všech závislostech.

Component-level SBOM metadata lze dále dělit na *Container image* a *RPM*. Každý z těchto komponentů vyžaduje jiná kritéria, avšak každý z nich je v dokumentech SBOM uveden na stejném místě. Potřeby společnosti Red Hat tedy zahrnují následování vlastní vydané specifikace, která dokáže zachytit data ohledně velké škály komponentů, z nichž každý má jiné vlastnosti, ale mohou mít stejné umístění v rámci dokumentu. Je tedy nutné pro správnou validaci vytvořit nástroj schopný tyto komponenty od sebe odlišit.

Řešení je vytvoření nástroje, který dokáže komponenty od sebe rozlišit a aplikovat veškerá pravidla daná specifikací. Pro rozlišení komponentů je nutné využít způsob filtrování jednotlivých prvků dokumentu SBOM.

1.6.2 Formáty využívané ve společnosti Red Hat

V rámci předchozích částí této práce byly představeny tři formáty dokumentů SBOM. Ve společnosti Red Hat jsou využívány formáty SPDX a CycloneDX, avšak využívané verze se mohou lišit.

Verze SPDX 3.0 není momentálně v celé organizaci využívána vzhledem ke své výrazné odlišnosti od ostatních formátů. Naopak lze nalézt dokumenty ve formátech SPDX 2.2 či CycloneDX 1.5. Všechny dokumenty SBOM mají jednotně formát JSON.

1.7 Související projekty

Nástrojů pro správu metadat SBOM existuje velká škála [50]. Problém je, že existující nástroje jsou stále vyvíjeny a žádný z nich neumí vše do dostatečné míry na to, aby bylo možné používat pouze jediný z nich [51]. Nástroje pro tvorbu dokumentů

SBOM někdy nezachytí veškerá fakta, nebo data v souborech SBOM zastarají, problémy jsou také mezi kompatibilitou nástrojů pro práci s těmito dokumenty [51]. Tomuto faktu nenapomáhá ani velké množství podob, které může dokument SBOM nabývat.

Vývoj však stále pokračuje kupředu a nástroje jsou neustále zdokonalovány. Některé užitečné nástroje pro tvorbu či správu metadat SBOM jsou uvedeny v následujících částech práce.

1.7.1 Nástroje svázané se specifikacemi SBOM

Organizace SPDX vydává vlastní sadu nástrojů pro práci s metadaty SBOM. Tyto nástroje existují v programovacích jazycích Java, Python, Golang, TypeScript a Javascript a mohou být využity i jako knihovny pro další nástroje v těchto jazycích [52]. SPDX pomůcky dokáží vytvářet a validovat dokumenty SBOM podle specifikace SPDX ve více verzích, převádět metadata mezi podporovanými formáty souborů nebo vizualizovat data obsažená v dokumentech. Navzdory tomu, že jsou tyto nástroje vydávány stejnou organizací, která i vydává tuto specifikaci, některé tyto nástroje ještě v čase psaní práce nemají plnou podporu nejnovější specifikace (SPDX 3.0) [53].

Organizace OWASP také vydala sadu nástrojů pro správu SBOM ve formátu CycloneDX. Tyto nástroje dokáží vytvářet, modifikovat, validovat, podepisovat, verifikovat podpisy, porovnávat a analyzovat metadata ve formátu CycloneDX [54]. Nejvíce užívaný CycloneDX nástroj je napsán v jazyce C#, avšak existují i knihovny pro použití v dalších programovacích jazycích, jako například Python nebo JavaScript.

1.7.2 Anchore Syft

Tento nástroj je velice využívaný pro vytváření metadat SBOM z kontejnerů a souborového systému. Nástroj je napsán v jazyce Go a v tomto jazyce může být využit také jako knihovna. Dokáže vytvářet dokumenty ve formátu SPDX 2.2 nebo 2.3 a CycloneDX 1.5 nebo 1.6 ve více souborových formátech. Podporuje velkou škálu ekosystémů, které dokáže v metadatech popsat a je stavěn pro využití se skenerem zranitelností Grype, také od organizace Anchore [51, 55].

Ačkoli to není primární úkol této aplikace, nástroj také umožňuje experimentálně metadata překládat mezi podporovanými formáty [55]. Z tohoto hlediska se stává zajímavým pro účely porovnání s nástrojem implementovaným v praktické části práce.

1.7.3 Interlynk sbomqs

Tento nástroj slouží k hodnocení kvality dokumentů SBOM, což je cíl i této práce. Aplikace umí hodnotit metadata na základě několika kritérií:

- Obsah minimálních informací podle NTIA (viz tabulku 1.1)
- Kvalita dokumentu
- Sémantika
- Sdílitelnost dokumentů
- Struktura dokumentu

V každé této kategorii může být obsaženo jedno nebo více pravidel, která jsou pojmenovaná zkrácenými identifikátory a delšími popisy. Kvalita SBOM pak závisí na identifikaci všech závislostí, komponent, licencí a na faktu, jestli je SBOM podepsán. Tento nástroj je velice silný, avšak neobsahuje možnosti od sebe komponenty rozlišovat a aplikovat na ně různá pravidla (tento požadavek byl zmíněn v části 1.6.1).

2 Implementace nástroje

V rámci diplomové práce byl vytvořen nástroj k hodnocení kvality a překládání metadat SBOM podle specifikace vydané společností Red Hat. Pro vývoj tohoto nástroje byl zvolen programovací jazyk Python. Výstupem musí být nástroj umožňující využití v automatizovaných procesech, tedy nejvhodnější přístup je implementace knihovny a nástroje s příkazovým rozhraním.

Aplikace byla pojmenována *SBOM Grader*, po její vyžadované funkcionalitě, jíž je hodnocení kvality dokumentů SBOM. Vzhledem k určení využití ve společnosti Red Hat, která poskytuje řešení s otevřeným kódem, je i tato implementovaná aplikace zveřejněna na platformě GitHub[1].

2.1 Vlastnosti aplikace

Vytvořená aplikace má za úkol pomoci společnosti Red Hat se správou softwarových kusovníků. Vyžaduje se, aby nástroj byl schopen následujících úkonů:

- ověření základních požadavků standardů SBOM
- ověření požadavků podle specifikace společnosti Red Hat
- překlad mezi standardy využívanými společností Red Hat

Tyto úkony musí být aplikovatelné pro metadata ve formátech, které jsou ve společnosti Red Hat využívány. Jak již bylo zmíněno v teoretické části (viz sekci 1.6.2), společnost Red Hat využívá SBOM ve standardech SPDX verze 2.2, 2.3 a CycloneDX verze 1.5 a 1.6, všechny tyto standardy jsou využívány pouze ve formátu JSON.

Z požadavků vyplývá, že je vhodné vytvořit nástrojů více, či vytvořit jeden nástroj s více schopnostmi. Byl vybrán přístup jediného nástroje s více funkcemi, jelikož implementace dílčích částí nástroje umožňuje vícenásobné užití vytvořeného kódu. Program byl rozdělen na tři separátní sekce, jimiž jsou jádro aplikace, nástroj hodnocení kvality a nástroj pro konverzi dokumentů.

2.2 Instalace nástroje

Aplikace SBOM Grader obsahuje závislosti, díky nimž není nutné veškerou funkcionalitu implementovat od základů. Tyto závislosti jsou uvedeny v elektronické příloze v souboru `pyproject.toml`, podle doporučení *Python Enhancement Proposal* (PEP) 621 [56] a podle návodu *Python Packaging Authority* (PyPA) [57] k distribuci softwarových balíčků.

[1] Projekt je dostupný na odkaze `https://github.com/BorekZnovustvoritel/SBOM-Grader`.

Aplikace je instalována nástrojem `pdm` [58]. Tento nástroj nejen využívá závislosti specifikované v souboru `pyproject.toml`, vytváří i vlastní soubor, `pdm.lock`. Tento soubor představuje zámek závislostí, který zajistí přenositelnost aplikace. Při instalaci na různých zařízeních jsou využity právě ty soubory závislostí, které jsou staženy z konkrétní adresy URL. Zvolený přístup zajišťuje, že – pokud je to možné – instalované závislosti jsou vždy neměnné a je vybrána pouze jejich korektní verze.

Instalace implementovaného nástroje je velice snadná z důvodu jeho zveřejnění na platformě *Python Packaging Index* (PyPI)[2]. Stačí pouze zadat příkaz `pip install sbomgrader` a nástroj je nainstalován. Tato vlastnost velice usnadňuje využitelnost nástroje v automatizovaných procesech, jelikož může být velice snadno nainstalován do spouštěných kontejnerů.

Druhou možností instalace je naklonováním repozitáře, kde je aplikace sdílena, poté lze využít instalaci nástrojem `pdm`. Tento přístup je vhodnější pro programátory, kteří chtějí do aplikace přispívat. Celý tento postup je uveden ve výpisu 2.1.

Výpis 2.1: Instalace aplikace na systému Linux

```
# Získání kódu
git clone https://github.com/BorekZnovustvoritel/SBOM-Grader.git
cd SBOM-Grader
# Vytvoření virtuálního prostředí Python
python3 -m venv .venv
# Aktivace prostředí
source .venv/bin/activate
# Získání nástroje PDM
python3 -m pip install pdm
# Instalace aplikace
pdm install
```

2.3 Jádro aplikace

Jádro projektu je balíčkem sloužícím jako knihovna pro jeho zbývající části. Celá aplikace byla cíleně psána jako balíček dále využitelný v jiných projektech. S tímto záměrem byl také zdrojový kód nástroje zveřejněn s permisivní licencí MIT [43].

Jádro poskytuje základní definice struktur souborového systému a datových struktur, také však obsahuje vlastní definici filtrování mapových struktur nazvanou *FieldPath Query Language*.

[2]Stránka projektu je dostupná na odkazu `https://pypi.org/project/sbomgrader`.

2.4 *FieldPath Query Language*

Tento filtrovací jazyk byl vytvořen jako nástroj umožňující aplikací pravidel či transformací pouze nad zvolenými položkami v metadatech. Obdobnými projekty jsou například `jmespath` nebo `jq`, které jsou však poměrně komplikované. `FieldPath Query Language` byl vytvořen pro jednoduché následování polí v metadatech SBOM, pro které není nutné zachovávat ani transformovat původní datové struktury a vždy je navrácen pouze seznam výsledků. Tímto omezením se zjednoduší celý jazyk a jeho využití by mělo být snadnější pro širší skupinu uživatelů.

2.4.1 Vyhledávání v mapách

Jazyk *FieldPath* podporuje vyhledávání v datových strukturách typu mapa či seznam. Pro dotazování na hodnoty mapy stačí pouze uvést název klíče, který na tuto hodnotu ukazuje. Pro nalezení hodnoty ve vnořených mapách stačí jednotlivé klíče oddělit tečkami.

2.4.2 Vyhledávání v seznamech

Vyhledávání v seznamech hodnot je již složitější, avšak neméně důležitou součástí tohoto jazyka. Vyhledávání nad seznamy je ohraničeno hranatými závorkami. V těchto hranatých závorkách může být obsažen jeden nebo více filtrovacích výrazů. Při využití více výrazů jsou odděleny čárkami, navráceny jsou pak jen položky splňující podmínky všech těchto výrazů.

Filtrovací výrazy jsou uvedeny v tabulce 2.1. V této tabulce se hovoří o volání funkce nad prvky. Touto funkcí se rozumí test odpovídající hodnoty či funkce přidávání do pole pro získání hodnoty.

2.4.3 Proměnné

Důležitou součástí tohoto jazyka jsou také proměnné. Jejich hodnota se odvíjí od právě zpracovávaného dokumentu a vždy obsahuje seznam hodnot. Proměnné lze poté používat ve filtrech seznamů, vždy na pravé straně, tedy na straně hodnot. Využití proměnné namísto pevně stanovené hodnoty je reprezentováno symboly `${` a `}` ohraničující její název.

Aby mohly být proměnné využity, musí být v příslušné sekci také definovány. Pro jejich definici postačuje uvést jejich název a *FieldPath* výraz, jež se má využít pro naplnění jejich hodnot.

Tab. 2.1: Implementované způsoby filtrování

Zápis filtru	**Chování filtru**
[|]	Funkce je zavolána nad všemi elementy seznamu. Může být porušeno, ne však všemi prvky.
[&]	Funkce je zavolána nad všemi elementy, žádný z nich pravidlo nesmí porušit.
[*atribut*=*hodnota*]	Funkce je zavolána pouze nad prvky, které obsahují atribut jména *atribut* a hodnoty *hodnota*.
[*atribut*!=*hodnota*]	Funkce je zavolána pouze nad prvky, které neobsahují atribut jména *atribut* a hodnoty *hodnota*.
[*atribut*%=*text*]	Funkce je zavolána pouze nad prvky, jejichž atribut jména *atribut* začíná textem *text*.
[*atribut*=%*text*]	Funkce je zavolána pouze nad prvky, jejichž atribut jména *atribut* končí textem *text*.
[*atribut*%*text*]	Funkce je zavolána pouze nad prvky, jejichž atribut jména *atribut* obsahuje text *text*.
[*číslo*]	Funkce je zavolána na prvku s indexem *číslo* v seznamu.
[@]	Funkce je zavolána pouze nad indexem odpovídajícím poskytnuté výchozí cestě. Bez poskytnutí této relativní cesty nelze tento filtr využít (viz část 2.4.4).

Při evaluaci filtrů obsahujících proměnné je nutné, aby výraz platil pro alespoň jeden prvek z tohoto seznamu. Pokud neplatí výraz pro žádný prvek ze seznamu hodnot, je právě zpracovávaný prvek vyloučen z výsledku vyhledávání.

Z hlediska časové složitosti je vhodné ukládat nalezené hodnoty pro každou proměnnou do množin, které umožňují dosažení i konstantní složitosti [59]. Tento přístup však nelze využít pro všechny existující hodnoty.

Programovací jazyk Python umožňuje do množin umisťovat pouze prvky mající implementovanou metodu `__hash__()`, což neplatí pro měnitelné (anglicky *mutable*) objekty. Mezi ně patří i seznamy a mapy. Pokud by tedy měla proměnná obsahovat i tyto datové struktury a ne pouze řetězce a čísla, došlo by k problémům. Aplikace k tomuto problému přistupuje tak, že všechny hodnoty proměnných jsou zpočátku získány v podobě seznamů, program se ale pokusí tyto seznamy převést na množiny, pokud mají být využity pro vyhledávání. Kvůli absenci metody `__hash__()` tento přístup nebude u některých hodnot možný a nastane výjimka, kterou je nutné ošetřit. Při této výjimce je nutné využít původní nezměněná data v podobě seznamů, jež ale umožňují dosáhnout pouze lineární složitosti. V případě neošetření této výjimky by došlo i k pádu aplikace.

2.4.4 Relativní cesty

Filtrovací jazyk také umožňuje vytváření relativních cest. Zástupný symbol `@` na začátku filtrovacího výrazu je nahrazen cestou k aktuálně zpracovávanému prvku, což umožňuje odkazy na položky umístěné v rámci tohoto prvku.

Je také umožněno využívat relativní cesty namísto filtrovacích výrazů seznamů. Celý tento výraz je nahrazen symbolem `[@]` a interpretovaná hodnota v hranatých závorkách se opět odvíjí od poskytnuté cesty k aktuálně zpracovávanému prvku. Nahrazení filtru seznamů lze využít pro nalezení cestu k objektu, jež sdílí pouze část cesty se zpracovávaným prvkem. Ukázky využití těchto vlastností jsou uvedeny v části 2.4.7.

2.4.5 Zápis cest

Cesty jsou ve formátu řetězců. Jednotlivé části cesty jsou odděleny tečkami či hranatými závorkami (v případě vyhledávání v seznamech). Hranaté závorky hrají významnou roli při načítání cesty, jelikož jejich počet určuje hloubku zanoření. Pro tečky toto však neplatí. Libovolné množství teček má stejný význam jako jediná tečka. Tečky lze rovněž uvádět před i za výrazy v hranatých závorkách a výsledek nebude dotčen. Stejně tak nebude výraz ovlivněn tečkami na začátku a konci výrazu.

Aby bylo možné cesty zapisovat do souborů formátu YAML bez vytváření příliš dlouhých řádků, je umožněno zapisovat cesty s mezerami ve výrazu. Je však nutné tuto mezeru umístit na pozici, která neovlivní výsledky vyhledávání. Při načítání jednotlivých částí specifikované cesty jsou vždy od jednotlivých názvů klíčů odebrány mezery před i za názvem, stejně tak jsou mezery odstraněny od výrazů na pravé straně výrazu (za symbolem operace).

Ukázky výrazů, které jsou všechny interpretovány totožně, jsou uvedeny ve výpisu 2.2. Interpretace zápisu cest je testována pomocí jednotkových testů, kterým se věnuje část 5.4.

Výpis 2.2: Ukázka ekvivalentních zápisů jediného výrazu

```
hello.foo[bar=1,spam!=2]ham
hello . foo [bar =1, spam != 2 ] ham
.hello.foo.[.bar.=1,.spam.!= 2 ].ham
 . hello . foo . [ . bar . =1, . spam . != 2] . ham
..hello..foo..[..bar..=1,..spam..!=2 ]..ham
```

2.4.6 Další vlastnosti filtrovacího jazyka

Jazyk *FieldPath* byl vytvořen pro rozhodování, nad kterými prvky z map či seznamů má být vykonána funkce. Toto umožňuje velice snadné vytváření testů hodnocení kvality (což je první součást zadání této práce). S tímto úmyslem byl také jazyk vytvořen.

Pokud je však místo testovací funkce nad každým nalezeným objektem zavolána procedura, která tento objekt přidá do předem daného seznamu, lze s jeho využitím extrahovat část dat z větších datových struktur. Z tohoto implementačního detailu pak plyne již dříve zmiňovaná vlastnost, že extrahovaná data mají vždy podobu seznamů obsahující extrahované prvky. A to i v případě, kdy filtrovací výraz logicky nemůže navrátit více prvků (navrátí tedy seznam s jediným prvkem). Na druhou stranu je tímto vynucen jednoznačný formát navrácených dat pro další práci s nimi.

Další vlastností je navrácení speciálního prvku nazvaného `FIELD_NOT_PRESENT`, pokud právě zpracovávaný prvek neobsahuje klíč mapy, na který je dotazován. Při dotazu na jakýkoli klíč tohoto prvku je vytvořena výjimka.

Pro zamezení výskytu těchto chyb musí uživatel specifikovat, že mu nevadí nepřítomnost daného pole. Toho docílí tím, že před dotazovaný klíč přidá virtuální klíč `?` oddělený tečkou, stejně jako při zápisu vnořených klíčů. Přítomnost tohoto virtuálního klíče způsobí tiché ukončení aktuální větve prohledávání, pokud není příští klíč obsažen v datech.

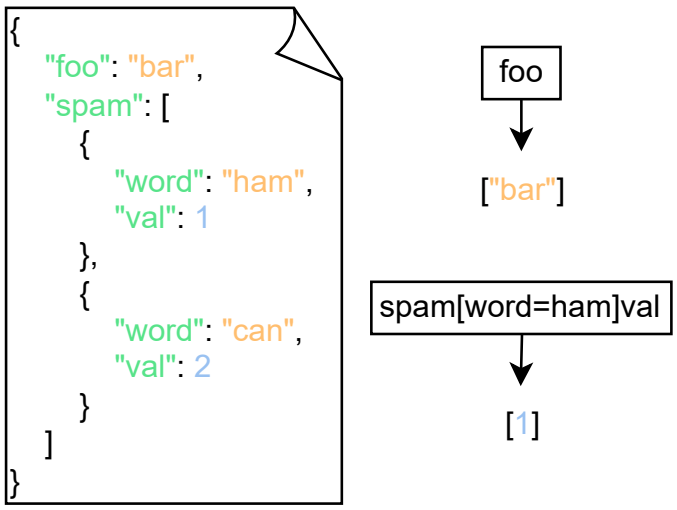
2.4.7 Příklady využití

Základní příklady využití tohoto vyhledávání cesty v dokumentech SBOM jsou uvedeny na obrázku 2.1. Na tomto obrázku jsou vyhledávací cesty uvedeny v obdélnících po straně, šipka pak ukazuje výstup z tohoto filtrování.

Výpis 2.3: Ukázka komplexnějšího filtrování s vnořením

```
fieldPath: >-
    packages[?.externalRefs[referenceType=purl]
    referenceLocator%=pkg:rpm/]SPDXID
```

Vnořené filtrování seznamů je znázorněno ve výpisu 2.3. V tomto výpisu je znázorněn také důvod využití vnořeného filtrování. Uvedený výraz navrací identifikátor `SPDXID` každého balíčku typu `RPM` ve specifikaci SPDX 2.3. Balíčky typu `RPM` jsou však identifikovány pouze identifikátorem `purl`, který začíná hodnotou `pkg:rpm/`. Identifikátorů `purl` však může mít každý balíček několik. Výsledek filtrování je tedy každý identifikátor `SPDXID`, který je uveden v balíčcích, jež obsahují alespoň jeden identifikátor `purl` začínající řetězcem `pkg:rpm/`.



Obr. 2.1: Ukázka filtrování

Pokud by se chtěl uživatel vnořeným filtrovacím výrazům vyhnout, nebylo by možné získat identifikátor `SPDXID`. Bylo by možné získat pouze informace o dané referenci balíčku, která obsahuje identifikátor `purl`. Příklad takového výrazu je uveden ve výpisu 2.4.

Výpis 2.4: Ukázka komplexnějšího filtrování bez vnoření

```
fieldPath: >-
    packages[|]?.externalRefs[referenceType=purl,
    referenceLocator%=pkg:rpm/]referenceType
```

Relativní cesty

Relativní cesty lze využít při definici samostatných částí kódu a jejich náležitostí. Toho je v aplikaci využito při definování map překladů (dále v části 4.2). Nad každým výskytem elementu lze v těchto mapách definovat proměnné obsahující zástupný znak `@`.

Například pokud je definován element cestou označenou `firstFieldPath` ve výpisu 2.5, budou nalezeny všechny balíčky, jejichž `SPDXID` začíná frází „`SPDXRef`“. Lze předpokládat, že tento element bude mít mnoho výskytů v dokumentu. Pokud je pro tento element definovaná proměnná `purls` z výpisu 2.5, bude tato proměnná obsahovat reference typu `purl`. Hodnota proměnné bude různá pro každý z výskytů tohoto elementu. Toho lze využít při přenášení dat mezi formáty SBOM.

Výpis 2.5: Ukázka relativních cest

```
firstFieldPath: 'packages[SPDXID\%=SPDXRef]'
...
firstVariables:
  - name: purls
    fieldPath: '@.externalRefs[referenceType=purl]referenceLocator'
```

Relativní filtrování seznamů

Cesty obsahující relativní filtrování seznamů jsou vhodné k nalezení atributu některého nadřazeného prvku. Ve výpisu 2.6 je uveden příklad využití této funkcionality. Tento výpis bude také nyní vysvětlen.

Výpis 2.6: Ukázka cest s relativním filtrováním

```
secondFieldPath: 'dependencies[ref=${foo}]?.provides[|]'
...
secondVariables:
  - name: bom_ref_of_parent
    fieldPath: 'dependencies[@].ref'
```

Ve formátu CycloneDX 1.6 jsou závislosti uvedeny v atributu `dependencies`, který obsahuje seznam objektů s klíči `ref` a dalšími poli, které jsou ale vždy seznamy řetězců. Tímto způsobem je zajištěno mapování *jedna ku n* mezi každým identifikátorem `ref` a jednotlivými identifikátory v seznamu pod klíčem `provides`.

Pro zachycení informace pro každý prvek je nutné iterovat nad všemi nalezenými identifikátory v seznamu `provides`, avšak klíč `ref` potřebný pro zachycení další části informace je uveden až o úroveň výše. Pro každý nalezený prvek ze seznamu `provides` (tyto prvky jsou například na cestách `dependencies[0]provides[0]`, `dependencies[0]provides[1]`, `dependencies[1]provides[0]`...) lze získat odpovídající nadřazený prvek dosazením správného indexu do pole `dependencies`. Přesně toto je relativním filtrováním umožněno.

Od relativních cest začínajících symbolem `@` se relativní filtrování odlišuje tím, že umožňuje opakovat pouze část cesty právě zpracovávaného prvku. Relativní cesty s prvním symbolem `@` mohou navrátit pouze objekty, které jsou přímo ve zpracovávaném prvku.

2.5 Podporované formáty SBOM

Pro jednoduchou rozšířitelnost jsou v jádru aplikace specifikovány podporované formáty SBOM pomocí konfiguračního souboru ve formátu YAML.

Tento soubor má pouze jediný klíč s názvem `formats`, který obsahuje seznam objektů. Každý z těchto objektů má atributy `name`, `value`, `expectedStructure` a `fallback`.

Hodnoty `name` a `value` jsou využity ke konstrukci třídy obsahující výčet těchto formátů. Název každého typu je načten z hodnoty prvního z těchto atributů, identifikátor použitý v ostatních souborech je načten z hodnoty `value`.

Atribut `expectedStructure` očekává objekt. Tento objekt má obsahovat klíče a očekávané hodnoty z metadat SBOM, které indikují, že se jedná právě o tento formát. Ku příkladu pro formát SPDX 2.3 je to mapa `{"spdxVersion": "SPDX-2.3"}`.

Pro opakovanou použitelnost stejných konfiguračních souborů (map konverzí či souborů testovacích pravidel, viz dále) lze také specifikovat zástupné formáty (v konfiguraci nazvány `fallback`). Tyto zástupné formáty značí, že pokud nebude nalezený konfigurační soubor s přesnou shodou formátu SBOM, může být využit formát jemu podobný. Například pokud má být hodnocen dokument formátu SPDX 2.2, ale existují pouze pravidla implementovaná pro verzi SPDX 2.3, může být i toto pravidlo využito, jelikož se tyto dva formáty liší pouze v detailech, které s velkou pravděpodobností nejsou v dokumentech využity.

3 Hodnocení kvality metadat SBOM

K ověření kvality metadat na základě specifikací vydaných tvůrci standardů i společností Red Hat byla vytvořena součást aplikace, která umožňuje spouštění funkcionálních testů nad poskytnutými daty. Nástroj využívá dříve vysvětlený filtrovací jazyk *FieldPath* (viz sekci 2.4).

3.1 Základní vlastnosti

Jak již bylo zmíněno v teoretické části práce, specifikací dokumentů SBOM je velké množství a nástroj nesmí být omezen pouze na jedinou specifikaci. Z tohoto důvodu byl vybrán přístup vytvoření validačních pravidel, která mohou být pro každou specifikaci implementována různým způsobem, jelikož stejná informace je v různých standardech obsažená na různých místech. Při porušení pravidla musí být uživatel upozorněn na tuto skutečnost a měl by mu být popsán způsob, jak dokument upravit, aby pravidlo porušeno nebylo.

Pravidla jsou rozdělena do tří kategorií. V kódu jsou tyto kategorie označeny anglickými slovy `MUST`, `SHOULD` a `MAY`. V implementovaném nástroji jsou tato klíčová slova užívána obdobně jako v RFC 2119 [60]. To znamená, že pokud je porušeno pravidlo úrovně `MUST`, SBOM je označen známkou F. Pokud je porušeno pravidlo `SHOULD`, dokument dostane známku sníženou o jedna. Pokud je porušeno pravidlo `MAY`, známka není dotčena, avšak uživatel bude upozorněn na potenciální zlepšení. Inspirací pro přípravu pravidel je repozitář s SBOM příklady poskytnutý společností Red Hat [61].

Pro různé typy SBOM dokumentů jsou navíc klíčové různé informace. Nelze tedy vytvořit pouze jedinou sadu validačních pravidel, která by dokázala oznámkovat správně veškeré typy možných dokumentů. Pokud ale nebude využita jediná sada pravidel, musí program sám umět rozpoznat, který soubor pravidel použít. Toho lze dosáhnout sledováním vztahů prvků dokumentu SBOM, které jsou specifické pro každý typ definovaný společností Red Hat.

Takto lze rozlišit dokumenty podle komponentu, který popisují. Není však snadné rozlišit SBOM na typy *Build-time* a *Release-time* (viz část 1.6.1). Tyto dokumenty jsou již z definice téměř totožné, do dokumentu typu *Release-time* jsou jen doplněny další informace. Proto nástroj bude od uživatele očekávat, že sám specifikuje, který z těchto typů by měl být daný SBOM. Pokud tak neučiní, nástroj musí využít validaci pro dokument typu *Release-time*, jejíž pravidla jsou přísnější.

3.2 Architektura řešení

Jak již bylo zmíněno, bylo vybráno řešení využívající pravidla o třech úrovních důležitosti. Před zahájením práce na programovém řešení byla pravidla sepsána a předložena ke kontrole konzultantům ve společnosti Red Hat. Z těchto sepsaných pravidel pak vyplynulo, že velké množství pravidel se opakuje pro všechny rozlišované verze SBOM dokumentů, některá pravidla se opakovala napříč alespoň některými typy a jen minimum pravidel bylo využito jen jednou. Tato pravidla se však v různých typech dokumentů mohla vyskytovat s různou důležitostí. Z těchto poznatků vyplynula nutnost užít definici pravidel oddělitelnou od kategorizace důležitosti pravidel.

3.2.1 Pravidla a soubory pravidel

Každé pravidlo (v programu pojmenované `Rule`) je definováno v souboru pravidel (`RuleSet`). Tento balík pravidel je definován jako soubor formátu YAML se strukturou definovanou pomocí funkcionality `jsonschema` [62]. Jedinou povinnou položkou je klíč `rules` ukazující na seznam pravidel. Také je však možné definovat proměnné využitelné v dalším filtrování (viz sekci 2.4.3).

Každé pravidlo musí mít název (`name`), hlášku k zobrazení v případě selhání (`failureMessage`) a seznam implementací. Implementacemi se rozumí specifikace formátu SBOM, tedy například SPDX 2.3. Pro každou implementaci může být pravidlo specifikováno různě. V aktuální verzi jsou pravidla implementována pro dokumenty SBOM ve specifikacích SPDX 2.3 a CycloneDX 1.6.

Cesta k polím

Tato součást definice pravidel definuje filtrovací výraz, který má být využit k dosažení polí, nad kterými má být provedeno funkční testování. Tento údaj očekává výraz z filtrovacího jazyka *FieldPath* (viz část 2.4).

Testovací funkce

Dalším důležitým polem v implementaci pravidel je `checker`. Toto pole udává způsob, jakým budou vyfiltrovaná pole validována. Podporované způsoby validace jsou uvedeny v tabulce 3.1.

Aby mohly být využity funkce jazyka Python (metoda `func_name`), je nutné umístit je do správného souboru. Ten musí být pojmenován po formátu SBOM s příponou `py`, ve složce nazvané `implementations` umístěné ve stejném adresáři jako soubor se souborem pravidel. Ukázka takové struktury souborového sytému je uvedena ve výpisu 3.1.

Tab. 3.1: Validační metody

Syntaxe metody	Popis metody
`eq:` *hodnota*	Je *hodnota* stejná jako hodnota pole v dokumentu?
`neq:` *hodnota*	Je *hodnota* různá od hodnoty pole v dokumentu?
`in:` *seznam*	Nachází se hodnota z dokumentu v *seznamu*?
`not_in:` *seznam*	Nenachází se hodnota z dokumentu v *seznamu*?
`str_startswith:` *text*	Začíná řetězec z dokumentu *textem*?
`str_endswith:` *text*	Končí řetězec z dokumentu *textem*?
`str_contains:` *text*	Obsahuje řetězec z dokumentu *text*?
`str_matches_regex:` *text*	Lze text z dokumentu zachytit regulárním výrazem *text*?
`length_eq:` *číslo*	Obsahuje seznam z dokumentu *číslo* elementů?
`length_gt:` *číslo*	Obsahuje seznam z dokumentu více než *číslo* elementů?
`length_lt:` *číslo*	Obsahuje seznam z dokumentu méně než *číslo* elementů?
`func_name:` *název*	Spustí funkci nazvanou *název* a výsledek validace rozhodne podle návratové hodnoty či chyb vrácených za běhu.

Proměnné

Pro jednodušší orientaci ve filtrovacích dotazech lze jejich dílčí části oddělovat do proměnných. Proměnné vyžadují pouze definici svého názvu a cesty v dokumentu.

Proměnné lze definovat v rámci souboru pravidel či jednotlivých pravidel. Při užití stejného názvu proměnné v souboru pravidel i v dílčím pravidle se užije definice proměnné napsaná v pravidle.

3.2.2 Kuchařky

Soubory pravidel definují, jak vypadají jednotlivá validační pravidla. Jak již ale bylo zmíněno dříve, je nutné mít schopnost pravidla využívat na více místech, s různými úrovněmi jejich síly. Z tohoto důvodu byly vytvořeny kuchařky. Kuchařky přesně specifikují, která pravidla a s jakou silou mají být použita pro který typ dokumentu SBOM.

Struktura kuchařek je jednoduchá. Jedná se o dokumenty formátu YAML obsahující klíče `MUST`, `SHOULD` a `MAY` a `rulesets`. Pole `rulesets` specifikuje, ze kterých sad pravidel mají být vyčítána pravidla, umožňuje specifikovat jak pravidla vestavěná, tak vlastní. Ostatní pole jsou pak jen názvy pravidel v seznamu.

Výpis 3.1: Ukázka umístění testovacích funkcí

```
/
├── ruleset1.yml ..................................... Soubor pravidel 1
├── ruleset2.yml ..................................... Soubor pravidel 2
└── transformers
    ├── ruleset1
    │   └── spdx23.py ..................... Funkce pro formát SPDX 2.3
    └── ruleset2
        ├── spdx23.py ..................... Funkce pro formát SPDX 2.3
        └── cdx16.py ...................... Funkce pro formát CycloneDX 1.6
```

Bylo vytvořeno celkem devět kuchařek pro různé typy dokumentů. Pokrývají metadata zachycující informace o obrazech kontejnerů (kuchařky existují zvlášť pro obrazy závislé na architektuře procesoru a pro jejich indexy), balíčcích RPM a produktech, byly však vytvořeny i obecné kuchařky. Většina kuchařek existuje pro časy *build* i *release*, výjimkou je kuchařka pro metadata o produktech. Ukázka kuchařky je uvedena ve výpisu 3.2.

Výpis 3.2: Ukázka obecné kuchařky

```
---
MUST:
  - Document passes schema validation
  - Main element is a package
SHOULD:
  - Document passes extra validation
  - All packages have a versionInfo
  - License list has correct form
  - All packages are referenced in relationships
  - All packages have a checksum
  - All packages have a supplier
  - All packages have at least one externalRef
  - RPM packages have packageFileName
rulesets:
  - general
  - specific
```

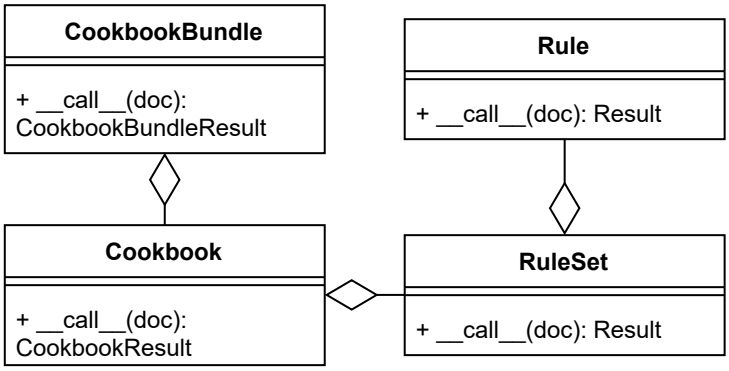
3.2.3 Balík kuchařek

Mohou se vyskytnout i situace, ve kterých je nutné validovat dokument pomocí více kuchařek zároveň. Aby se předešlo několikanásobnému spouštění stejných pravidel, byl vytvořen i objekt abstrahující několik kuchařek zároveň. Tento balík kuchařek zjistí, která pravidla jsou vyžadována alespoň jednou kuchařkou a spustí každé takto nalezené pravidlo právě jednou. Poté navrátí výsledek reprezentující spuštění kuchařek obsažených v souboru. Touto úsporou nedochází ke zbytečnému spouštění testů, které nejsou vyžadovány žádnou kuchařkou, a zároveň předchází opakovanému spouštění stejných testů na stejném dokumentu různými kuchařkami.

Balíky kuchařek nejsou reprezentovatelné žádným jediným souborem formátu YAML jako ostatní části struktury, ale lze je vytvořit za pomocí více souborů s kuchařkami.

V balíku kuchařek lze také zvolit *hlavní* kuchařku, jejíž výsledná známka bude reprezentovat známku získanou spuštěním celého balíku kuchařek. To je vhodné, pokud chce uživatel spustit více příbuzných kuchařek pro zjištění návrhů na vylepšení, ale směrodatný je výsledek pouze jediné kuchařky.

Pro zlepšení představy o struktuře nástroje je uveden zjednodušený diagram tříd na obrázku 3.1. Z tohoto diagramu byly odebrány atributy, vztahy s dalšími třídami i většina metod, z důvodu velké komplexity takového diagramu.



Obr. 3.1: Zjednodušený diagram tříd nástroje známkování

3.3 Popis využití nástroje

Pro práci s vytvořenou aplikací bylo vytvořeno terminálové uživatelské rozhraní. Aplikace pro hodnocení kvality se spustí příkazem `sbomgrader grade`. Rozhraní nástroje vyžaduje vždy právě jeden poziční argument, tím je hodnocený dokument SBOM. Další argumenty jsou uvedeny v tabulce 3.2.

Tab. 3.2: Parametry nástroje pro známkování SBOM

Argument	**Popis funkce argumentu**
`-h`	Zobrazí nápovědu a ukončí aplikaci.
`-c`	Umožňuje uživateli specifikovat, které kuchařky chce využít.
`-ct`	Umožňuje volit typ komponenty popisované SBOM dokumentem pro výběr z existujících kuchařek. Možnosti jsou `image`, `image_index`, `rpm`, `product` nebo `generic`.
`-st`	Umožňuje volit typ SBOM dokumentu pro výběr z existujících kuchařek. Možnosti jsou `build` nebo `release`.
`-g`	Specifikuje minimální známku, kterou musí SBOM obdržet, aby nástroj neoznačil výstup jako chybný. Možnosti jsou písmena `A` až `F`, výchozí je hodnota `B`.
`-o`	Specifikuje formát výstupu. Možnosti jsou `visual`, `markdown`, `json` nebo `yaml`. Výchozí je hodnota `visual`.

Výstup z nástroje může nabývat několik podob. Těmi jsou formáty Markdown, YAML, JSON nebo vizuální výstup. Markdown je datový formát zaměřený na čitelnost lidmi, další dva formáty je vhodné využít ke zpracování výstupu v další automatizaci. Příklad výstupu ve formátu Markdown je uveden ve výpisu 3.3. Formát tohoto výstupu je využit i při vykreslování vizuálního výstupu, zde je však tento formát kompilován za užití balíčku `rich` [63].

Výstup je vždy nasměrován na standardní výstup, pro zápis výpisu do souboru jej lze přesměrovat z příkazové řádky, jak je u Unixových systémů zvykem. Pokud při běhu nastanou chyby, či je nutné uživatele na něco upozornit, aplikace tyto informace vrátí do standardního chybového kanálu. Díky tomu lze od sebe jednoduše oddělit očekávaný výstup a ostatní hlášky.

Nástroj také umožňuje vypsat již implementované kuchařky. Toho lze dosáhnout příkazem `sbomgrader list --cookbooks`. Výstup tohoto příkazu je opět ve formátu Markdown vykresleném pomocí knihovny `rich`. Navrácen je seznam názvů jednotlivých vestavěných kuchařek, které lze vybrat při specifikaci pomocí parametru `-c` příkazu `sbomgrader grade`. Ukázka tohoto výstupu je uvedena ve výpisu 3.4.

Výpis 3.3: Výstup z nástroje, zkráceno

```
# Cookbook bundle result
**Grade: A**

## Used cookbooks
- image_index_release

---
# Cookbook: image_index_release

## Summary
Achieved grade: A

### MUST:
- Document passes schema validation ✓
- Main element is a package ✓

### SHOULD:
- All packages are referenced in relationships ✓
- All packages have a supplier ✓
- All packages have a checksum ✓
- All Image PURLs contain qualifier repository_url and tag ✓
- All packages have at least one externalRef ✓
```

Výpis 3.4: Seznam existujících kuchařek

```
$ sbomgrader list -c

 - generic_release
 - generic_build
 - image_release
 - image_build
 - rpm_release
 - image_index_build
 - image_index_release
 - product
 - rpm_build
```

4 Překlad mezi formáty SBOM

Druhou funkcionalitou nástroje je možnost překladu mezi formáty metadat SBOM využívaných ve společnosti Red Hat. Nástroje pro tyto konverze již v dnešní době existují (příkladem může být dříve zmíněný nástroj *Anchore Syft*, viz sekci 1.7.2), avšak společnost Red Hat vyžaduje, aby tato konverze byla bezeztrátová pro veškerá jimi definovaná data.

Formáty SBOM jsou vcelku flexibilní, existuje tedy více způsobů, jak do nich lze informaci zapsat. Ve společnosti Red Hat byla vydána specifikace, která by měla těchto více možností sjednotit na jedinou. Tato možnost však není univerzálně uznávána jako nutná součást a ostatní nástroje tyto informace mohou během překladu vypustit.

4.1 Způsob konverze

Pro zajištění bezeztrátovosti informace byl zvolen přístup, který mapuje informace do dvou standardů zároveň. Tyto informace jsou zachyceny v *mapě překladů* (v kódu `TranslationMap`).

Tyto mapy obsahují odkazy na dva formáty SBOM a seznam částí informace k zachycení (v kódu je pro tyto části informace využito pojmenování *chunk*). Každá tato část má pojmenování (pro lepší orientaci v konfiguračních souborech), cesty k polím v obou formátech a také formát, do kterého mají být informace v obou formátech zachyceny.

Pro vysvětlení konverze definujme D_0 jako vstupní dokument a D_1 jako dokument výstupní. Při konverzi nástroj postupuje po dílech informace. Pro každý díl jsou nalezeny všechny jeho výskyty v dokumentu D_0.

Pro každý výskyt informace jsou načteny všechny definované proměnné. Proměnné jsou pak vyplněny do formátu dat dokumentu D_1. Formát dat je definován v rámci dílu informace pomocí šablony `jinja2` [64]. Po vyplnění jsou data umístěna do D_1 na definované místo.

Pro každý výskyt informace je také možné využít proměnné s relativní cestou, jak bylo zmíněno v části 2.4.4.

4.2 Formát map konverzí

Aby bylo dosaženo co nejlepší rozšiřitelnosti nástroje, jsou mapy konverzí definovány pomocí souborů formátu YAML. Všechna pole týkající se jednoho ze dvou formátů SBOM, mezi kterými mapa překládá, jsou uvozena předponou *first* nebo *second*. Tyto předpony **nevyjadřují** směr překladu, mapy umožňují obousměrnou konverzi.

Soubor s mapou definuje dva formáty SBOM, pro jejichž konverzi lze tuto mapu využít. Oba formáty metadat SBOM jsou uvedeny pod klíči `first` a `second`. Mapa také nepovinně obsahuje proměnné, v polích `firstVariables` a `secondVariables`. Tyto atributy obsahují seznam objektů, každý z nich obsahuje název (klíč `name`) a výraz `fieldPath`.

Dalšími atributy jsou pak `chunks`, `firstPreprocessing`, `secondPreprocessing`, `firstPostprocessing` a `secondPostprocessing`. Tato pole mají v mapách překladu významné a netriviální role, proto se jim budou věnovat následující části práce. Minimalistický příklad mapy konverzí je uveden ve výpisu 4.2.

4.2.1 Díly informace

Jednotlivé části informace o mapování jsou uvedeny pod klíčem `chunks`. Formát tohoto pole je seznam objektů. Každý z těchto záznamů obsahuje atributy `name`, `firstData`, `firstFieldPath`, `secondData` a `secondFieldPath`. Nepovinně také mohou obsahovat seznamy `firstVariables` a `secondVariables`. V příkladovém výpisu 4.2 náleží částem informací řádky 5–25.

Proměnné (v polích `firstVariables` a `secondVariables`) umožňují načítat hodnoty, jež budou využity v datových částech. Jejich definice je obdobná jako v již dříve zmiňovaných souborech s pravidly (v části 3.2.1), avšak je pro ně umožněno i využití relativních cest[1] s počátkem v cestě k aktuálnímu výskytu informace. Filtrovací výraz definující možné cesty je uveden v poli `firstFieldPath` či `secondFieldPath` každého dílu informace.

Data

V atributech dat (v polích `firstData` a `secondData`) se nacházejí řetězce obsahující šablony `jinja2`. V těchto šablonách je možné využívat proměnné **z opačného formátu**. Tato zvláštnost je způsobena tím, že načítání dat je smysluplné pouze z formátu, který byl využit pro vstupní data. Tyto načtené hodnoty se však předávají do výstupního formátu.

Jak lze vidět ve výpisu 4.2, například na řádku 22, šablony `jinja2` využívají pro substituci zdvojené složené závorky.

Šablony `jinja2` poskytují také možnosti filtrování hodnot z proměnných. Kromě již existujících možností lze také využít vlastní definice filtrů. Pro ulehčení práce byly implementovány funkce `unwrap`, `slice`, `fallback`, `unify` a `func`.

Funkce `unwrap` vrací první prvek z poskytnutého seznamu. Díky již dříve zmiňované vlastnosti jazyka `FieldPath`, že jsou vždy navráceny seznamy nalezených hodnot, ulehčuje tato funkce definici dat k zápisu.

[1] Relativní cesty byly vysvětleny v části 2.4.4 a příklady jejich užití jsou uvedeny v části 2.4.7.

Filtr `slice` umožňuje rozdělit seznam či řetězec na jeho části. Funkce má dva argumenty, `start` a `end`, které společně určují indexy začátku i konce iterování. Při nevyplnění některé z těchto hodnot je navrácen seznam začínající v počátku originální hodnoty, respektive končící na poslední položce původního seznamu.

Využití funkce `fallback` spočívá ve vybrání náhradních hodnot, pokud jsou jiné hodnoty nedostupné. V rámci specifikace lze stejnou informaci uchovávat na různých místech, přičemž ne vždy je v dokumentech využité stejné pole. Tato funkce tedy přijme libovolný počet argumentů a navrátí hodnotu prvního neprázdného z nich, čímž nalezne první neprázdnou hodnotu mezi proměnnými.

Datový procesor `unify` se užívá stejně jako filtr `fallback`, avšak spojuje všechny hodnoty všech dostupných proměnných do jediného pole. Díky tomu lze spojit hodnoty získané v různých částech dokumentu, pokud je to potřeba.

Filtr `func` vyžaduje argument `name`, kterému je přiřazen řetězec obsahující název funkce. Tato funkce programovacího jazyka Python je poté zavolána nad daty vstupujícími do filtru. Aby byla funkce správně načtena, musí být obsažená v souboru nazvaném `first.py`, `second.py` nebo názvem SBOM formátu s příponou `py`. Tento soubor musí být obsažen ve složce nazvané stejně jako je nazván soubor s mapou (bez přípony `yml`), která musí ležet ve složce `transformers` ve stejném adresáři jako soubor s mapou. Přístupné jsou pouze ty transformační funkce, které jsou obsaženy v souboru příslušícímu vstupnímu formátu. Ukázka této souborové struktury je uvedena ve výpisu 4.1.

Výpis 4.1: Ukázka umístění transformačních funkcí

```
/
├── map1.yml ..................................... Mapa koverzí 1
├── map2.yml ..................................... Mapa koverzí 2
└── transformers
    ├── map1
    │   ├── first.py ..... Transformace pro hodnoty z prvního formátu
    │   └── second.py ..... Transformace pro hodnoty z druhého formátu
    └── map2
        ├── spdx23.py ..... Transformace pro hodnoty z formátu SPDX 2.3
        └── cdx16.py ..... Transformace pro hodnoty z formátu CycloneDX 1.6
```

Využití funkcí `func` a `unwrap` bylo také nastíněno ve výpisu 4.2, například na řádku 25. Z této ukázky je také patrné, že data jsou uložena ve formátu řetězce, který je uvozen symbolem „`|-`“.

4.2.2 Preprocesory

Preprocesory v kontextu map překladů jsou funkce volané pro přípravu vstupních dat ke zpracování. Tyto funkce jsou definovány v souborech s adresářovou strukturou totožnou s transformačními funkcemi (viz výpis 4.1), pouze s tím rozdílem, že adresář `transformers` je přejmenován na `preprocessing`.

Funkce musí být psány v jazyce Python a přijímat argument, do něhož je vložena datová struktura obsahující celý dokument SBOM. Pokud funkce navrátí hodnotu, bude tato hodnota užita namísto původního dokumentu. Pokud hodnotu nenavrátí, předpokládá se, že změny byly provedeny přímo v datech.

Všechny funkce ze souborů, které mají být užity, musí být uvedeny v mapě překladů pod klíči `firstPreprocessing` či `secondPreprocessing`. Tato pole obsahují seznamy řetězců obsahující název každé funkce. Příklad tohoto formátu zápisu je uveden ve výpisu 4.2, na řádcích 27–30.

Preprocesory je vhodné užít k transformaci dat na formát očekávaný v dalších částech map překladu. Změny zde definované se provádějí na vstupních datech před jejich zpracováním dalšími částmi mapy.

4.2.3 Postprocesory

Postprocesory jsou na rozdíl od preprocesorů funkce volané po dokončení přenosu všech částí informace do nového dokumentu.

Tyto procedury musejí přijímat dva argumenty. Prvním argumentem je původní dokument, druhým argumentem je pak nově dokončený dokument. Lze tedy nahlížet do původních dat, pokud je to pro některý postprocesor nutné. Těmito původními daty se myslí dokumenty, které již byly zpracovány všemi preprocesory.

Postprocesory lze využívat pro dokončení detailů (např. změna pořadí prvků v dokončeném dokumentu), či jimi lze provést funkcionální transformace dokumentů.

Názvy vyžadovaných postprocesorů je nutné uvést do polí `firstPostprocessing` a `secondPostprocessing`. Příklad tohoto zápisu je uveden ve výpisu 4.2, na řádcích 31–34. Funkce jsou vyhledávány v souborech se strukturou obdobnou transformačním funkcím (viz výpis 4.1), avšak je potřeba přejmenovat adresář `transformers` na `postprocessing`, obdobně jako tomu bylo již u preprocesorů.

4.3 Vytvořená mapa konverzí

Vzhledem k požadavkům společnosti Red Hat bylo nutné vytvořit jednu mapu konverzí. Tato mapa má za úkol převádět metadata mezi formáty SPDX 2.3 a CycloneDX 1.6. Díky mechanismu zástupných formátů (uvedeno v části 2.5) lze však převádět i na formáty SPDX 2.2 či CycloneDX 1.5.

Výpis 4.2: Příklad mapy překladů

```yaml
---
first: spdx23                # Zkratka pro SPDX 2.3
second: cdx16                # Zkratka pro CycloneDX 1.6

chunks:                      # Díly informace
  - name: Skeleton           # Statické informace
    firstFieldPath: .
    firstData: |-
      spdxVersion: "SPDX-2.3"
    secondFieldPath: .
    secondData: |-
      bomFormat: "CycloneDX"
  - name: Components         # Komponenty
    firstVariables:
      - name: spdxid
        fieldPath: '@.SPDXID'
    secondVariables:
      - name: cdx_id
        fieldPath: '@.bom-ref'
    firstFieldPath: packages[|]
    firstData: |-
      SPDXID: SPDXRef-{{ cdx_id | func(name="bid_to_spdxid") }}
    secondFieldPath: components[|]
    secondData: |-
      bom-ref: {{ spdxid | unwrap | func(name="spdxid_to_bid") }}

firstPreprocessing:          # Úprava SPDX na vstupu
  - normalize_spdx
secondPreprocessing:         # Úprava CycloneDX na vstupu
  - normalize_cdx
firstpostprocessing:         # Úprava SPDX na výstupu
  - make_it_more_spdx_like
secondPostprocessing:        # Úprava CycloneDX na výstupu
  - summon_a_cyclone
```

Mapa obsahuje část popisující statické informace, které náleží každému dokumentu. Mezi tyto informace patří například verze formátu. Mapy konverzí umí automaticky tyto informace upravit, pokud je vybrán jiný výstupní formát, než pro který je mapa psána. V tomto případě jsou informace upraveny podle popisu polí zástupných verzí.

V dalších částech mapy jsou uvedeny informace o jednotlivých komponentech a jejich vztazích. Vztahy mezi komponenty mohou být vyjádřeny stejnými klíčovými slovy, ale znamenat jiný vztah. Je tedy nutné sledovat i typy jednotlivých komponent, aby byl překlad co nejpřesnější.

Příkladem různého vztahu při stejném výrazu je SPDX vztah `CONTAINS`, který je využit jak pro závislosti obrazů kontejnerů, tak i pro nadřazené repozitáře k balíčkům RPM [61]. Ve formátu CycloneDX jsou tyto vztahy vyjádřeny naprosto odlišně, avšak při zkoumání typů jednotlivých komponent lze jednoduše určit, o který vztah se jedná.

Vytvořená mapa tedy obsahuje jedenáct kusů informace, které obsahují informace mapované do zmíněných dvou formátů. Celkem má mapa 856 řádků, avšak toto číslo se může v budoucnu změnit při úpravách. Úpravy jsou však žádoucí pro technologii, jejíž samotná standardizace ještě prochází vývojem. Dříve citovaná standardizace SBOM v rámci společnosti Red Hat se v současné době stále vyvíjí, což lze odvodit i z aktivity v repozitáři specifikace [61].

4.4 Popis využití nástroje

Obdobně jako nástroj pro hodnocení kvality metadat, i tento nástroj disponuje uživatelským rozhraním v příkazové řádce. Pro spuštění je nutné nástroj mít nainstalovaný (popsáno v části 3.3), poté stačí zadat příkaz `sbomgrader convert` s příslušnými parametry. Příkaz vždy vyžaduje poziční argument obsahující cestu k dokumentu ke konverzi. Další parametry jsou uvedeny v tabulce 4.1

Tab. 4.1: Parametry nástroje pro konverzi SBOM

Argument	**Popis funkce argumentu**
`-h`	Zobrazí nápovědu a ukončí aplikaci.
`-f`	Vybere výstupní formát SBOM. Povinný argument.
`-m`	Umožní specifikovat vlastní mapu konverzí. Udává cestu k souboru.

Vlastních map konverzí lze specifikovat i více, vždy je však použita pouze jediná. Mapy specifikované uživatelem mají přednost před vestavěnými mapami. Také přesné shody formátu mají přednost před zástupnými verzemi (více v části 2.5).

Pro vypsání již vestavěných map lze využít příkaz `sbomgrader list --maps`. Tento příkaz vypíše seznam dostupných směrů konverze podle dostupných map, výstup je formátován opět pomocí knihovny `rich`. Nástroj také zobrazuje náhradní možnosti podle zástupných formátů (viz část 2.5). Ukázka výstupu z tohoto příkazu je uvedena ve výpisu 4.3.

Výpis 4.3: Seznam existujících map

```
$ sbomgrader list -m

 - cdx16 <-> spdx23
 - cdx15 <-> spdx22 (fallback)
 - cdx16 <-> spdx22 (fallback)
 - cdx15 <-> spdx23 (fallback)
```

5 Popis nasazení a testování nástroje

Do budoucna se počítá s nasazením tohoto nástroje do procesu publikování software ve společnosti Red Hat. Měl by zajišťovat následování pravidel pro dokumenty SBOM tak, aby byla tato metadata vhodná k zajištění sledování slabin v software a odstranění rizik s těmito slabinami spojenými. Nástroj má být nasazen do existujících CI/CD řešení, mezi které patří OpenShift Pipelines [65], GitLab CI [66] nebo GitHub Workflows [67].

Nástroj lze také využít ke kalibraci nově vznikajících nástrojů, které mají za úkol generovat dokumenty SBOM podle specifikace Red Hat. Tento způsob využití je vhodný při nastavení nástroje v kroku automatických testů v rámci CI/CD automatizace daného nástroje.

5.1 Nasazení aplikace s využitím nástrojů CI/CD

Pro využití v technologii OpenShift je nutná kontejnerizace aplikace vytvořením obrazu jejího (anglicky *container image*). K tomuto účelu projekt obsahuje soubor `Dockerfile` obsahující jednotlivé kroky nutné k sestavení tohoto obrazu. K lokálnímu sestavení obrazu lze využít příkazy `docker build .` či `podman build .` podle instalovaného nástroje.

Po vytvoření obrazu je nutné jej nahrát do některého repozitáře, ke kterému má CI/CD dostatečný přístup. Možnostmi jsou například Quay [68] či DockerHub [69]. Po nahrání již lze aplikaci využít v CI/CD, stačí pouze specifikovat URL, na které se nachází nahraný obraz. Pro využití tohoto obrazu mimo OpenShift Pipelines je nutné také k vytvořenému kontejneru připojit virtuální úložiště (anglicky *volume*), které obsahuje dokumenty SBOM ke zpracování, mapy překladů či kuchařky hodnocení dokumentů.

K datu odevzdání práce nebyla aplikace prozatím nasazena, avšak o jejím využití proběhlo ve společnosti Red Hat již několik jednání. Aplikace je ve stavu, který nasazení umožňuje.

5.2 Metodologie testování aplikace

Aby byla ověřena funkčnost aplikace, je nutné navrhnout a implementovat testy. Tyto testy mají za úkol zjistit nejen, zda je aplikace funkční, ale také zda jsou zdrojové soubory správně naformátované, ať je v budoucnu snadnější zdrojový kód udržovat.

Pro automatizaci testů byl zvolen nástroj `tox` [70]. Tento nástroj umožňuje vytvoření konfiguračního souboru s názvem `tox.ini`, ve kterém jsou definována prostředí pro běh testů. Pro spuštění všech testů sekvenčně lze užít jediný příkaz, a to právě `tox`. Pro výběr jednoho z prostředí lze také využít přepínač `-e` s názvem zvoleného prostředí (například `tox -e yamllint`).

Jelikož je zdrojový kód aplikace zveřejněn na platformě GitHub, lze k automatickému testování využít mechanismus *GitHub Workflows*. Tento mechanismus zajistí, že jsou testy vykonány nad každým novým příspěvkem do kódu, čímž se zajistí ověření kvality s každou změnou. Pro ukázku je ve výpisu 5.1 uveden závěr kroku `tox` z GitHub Workflow pro validaci kódu. Jak lze vidět, všechny kroky byly v tomto případě úspěšné.

Výpis 5.1: Závěr výpisu z GitHub Workflows

```
test: OK (14.35=setup[10.48]+cmd[3.88] seconds)
perf-test: OK (14.17=setup[8.59]+cmd[5.58] seconds)
yamllint: OK (8.91=setup[8.58]+cmd[0.32] seconds)
black: OK (9.60=setup[8.77]+cmd[0.84] seconds)
mypy: OK (13.47=setup[8.48]+cmd[4.99] seconds)
congratulations :) (60.58 seconds)
```

5.3 Testy formátování

Formátování zdrojového kódu zajišťuje program `black` [71]. Tento nástroj dokáže nejen zajistit formátování zdrojového kódu, ale umí i vytvořit jednoduchý test, který zjistí, zda jsou všechny zdrojové soubory naformátované správně. Test selže, pokud tomu tak není.

Jak bylo popisováno v dřívějších kapitolách, aplikace obsahuje také velké množství souborů formátu YAML. Obdobně jako nástroj `black` ověřuje formátování zdrojového kódu v jazyce Python, lze pro tyto soubory využít aplikaci `yamllint` [72]. Test opět selže, pokud formát souborů neodpovídá kritériím definovaným v nástroji.

Byl také využit nástroj `mypy` [73] k ověření užití správných typů. Využití takového nástroje zmenšuje šanci na chyby v kódu. Test selže při nalezení nedostatků ve statickém typování aplikace. Napravit výsledek testu lze opravou chyby či explicitním označením nálezu jako falešně pozitivní.

5.4 Jednotkové testy

Nejspíše nejdůležitějším prvkem testovací sady jsou však testy funkcionality. Byly vytvořeny jednotkové testy s užitím balíčku `pytest` [74], který umožňuje také parametrizaci testových funkcí.

Testy jsou zaměřeny jak na dílčí funkcionalitu některých kritických funkcí či metod, tak i na komplexnější procesy. Pro ověření kvality nástroje jsou operace známkování a překlad metadat SBOM testovány za užití dat poskytnutých společností Red Hat. Celkem je nad aplikací spouštěno 41 jednotkových testů.

5.5 Testování výkonnosti aplikace

Při implementaci aplikace probíhalo testování na testovacích datech zveřejněných společností Red Hat. Tato data jsou poměrně malá (seznamy komponentů v kusovnících obsahují řádově jednotky či desítky položek), při jejich zpracování nenastal žádný zásadní problém.

Opravdová data však mohou být komplexnější (lze očekávat stovky až nízké tisíce komponentů), což může mít dopad na výkonnost aplikace. Při testování první verze překladače metadat SBOM[1] trvala operace překladu přibližně deset minut. V závislosti na tomto zjištěném faktu bylo nutné vytvořit plán optimalizace, jakož i plán testování výkonnosti aplikace.

5.5.1 Profilování původní verze překladače dat

Jako testovací data jsou využity dokumenty obsahující sto komponentů ve formátu SPDX 2.3. Pro profilování aplikace byl využit nástroj `cProfile`, který je vestavěný přímo v základní knihovně jazyka Python [75]. Užití tohoto nástroje je velice snadné pro spustitelné balíčky jazyka Python (vyznačují se přítomností souboru `__main__.py`), jímž je i implementovaný nástroj. Ke spuštění profilování byl využit příkaz `python3 -m cProfile -m sbomgrader convert <nazev_souboru>.json -f <format>` (je nutné aplikaci instalovat jako vývojář, což bylo nastíněno ve výpisu 2.1).

Profilování bylo uskutečněno na počítači Asus x571gd vybaveným procesorem Intel i5 8300H (architektury `AMD86_64`) o základní frekvenci 2,5 GHz a maximální frekvenci 4 GHz. Procesor má čtyři jádra, osm vláken. Počítač má 8 GB RAM

[1] Tato verze je dostupná pod Git značkou (*tagem*) `first_translation_implementation` v repozitáři projektu: `https://github.com/BorekZnovustvoritel/SBOM-Grader/releases/tag/first_translation_implementation`.

architektury DDR4 a je na něm nainstalován operační systém GNU/Linux Manjaro s verzí kernelu 6.6.85-2 s verzí jazyka Python 3.13.2. Další parametry jsou pro výsledky profilování irelevantní.

Tab. 5.1: Výsledky profilování před optimalizací

Volání vše/přímé	**Celk. čas** [s]	**Volaný kód** soubor:řádek(funkce)
1710/1	5,663	`{built-in method builtins.exec}`
1	5,663	`<string>:1(<module>)`
1	5,663	`<frozen runpy>:201(run_module)`
815/1	5,586	`<frozen runpy>:65(_run_code)`
1	5,586	`__main__.py:1(<module>)`
1	5,205	`__main__.py:216(main)`
1	5,201	`__main__.py:164(convert)`
1	4,869	`translation_map.py:302(convert)`
11	4,821	`translation_map.py:138(convert_and_add)`
202	2,937	`translation_map.py:44(render)`
564062/5158	2,111	`field_resolve.py:425(_run_on_path)`
631	2,096	`field_resolve.py:320(resolve_variables)`
202	1,507	`environment.py:1092(from_string)`
202	1,504	`environment.py:731(compile)`
417	1,457	`field_resolve.py:625(__populate_variables)`
410	1,451	`field_resolve.py:705(get_objects)`
410	1,450	`field_resolve.py:636(run_func)`
202	1,444	`field_resolve.py:755(insert_at_path)`
...	...	...

Výsledky tohoto profilování jsou zaneseny do tabulky 5.1. Jedná se samozřejmě o zjednodušený výpis, z něhož byly vypuštěny některé údaje. Pro přehlednost byla volání na vestavěné funkce ponechána v černé barvě, volání na vlastní funkce přebarvena na modro a šedě jsou pak označena volání instalovaných závislostí, v tomto případě `jinja2`. Lze vidět, že největší část času je využita na výpočty v implementovaném kódu, nejvíce jsou však využity moduly `translation_map.py` (překlady dokumentů) a `field_resolve.py` (vyhledávání v dokumentech). Soubor `__main__.py` je hlavním modulem, z nějž jsou veškeré ostatní moduly volány.

5.5.2 Optimalizace

V návaznosti na tyto výsledky byly implementovány optimalizace do modulů, které byly profilováním odhaleny jako pomalé. Tyto optimalizace spočívají v redukci počtu volání funkcí se stejnými parametry díky uložení a opětovnému užití výsledku prvního takového běhu.

Další implementovaný způsob optimalizace je úzce svázán s jazykem `FieldPath`. Jak již bylo zmíněno dříve, tento jazyk umožňuje využívat proměnné, jejichž hodnota je odvozena z dokumentů (viz část 2.4.3). V původní verzi aplikace jsou před využitím jazyka automaticky naplněny všechny proměnné hodnotami.

V optimalizované verzi jsou však hodnotami naplněny jen ty hodnoty, které jsou pro daný výraz nutně potřebné. Stejné optimalizace je užito i pro naplnění hodnot nutných k vytvoření nových dat pro překlad.

5.5.3 Profilování nové verze překladače dat

Po dokončení optimalizace bylo profilování znovu spuštěno s novou verzí aplikace na stejném stroji se stejnými vstupními daty. Výsledek tohoto profilování je uveden ve výpisu 5.2. Lze si všimnout kratšího celkového běhu aplikace i menšího počtu volání na implementované funkce. Výsledek překladu (při použití stejné verze mapy překladů) je totožný. Po této optimalizaci je již aplikace blízko limitům daným využitými závislostmi, což lze usoudit z přítomnosti funkcí knihoven `jinja2` a `py-yaml` vysoko ve výpisu profilování.

Vhodným způsobem, jak dále optimalizovat tuto aplikaci, je omezit počet volání těchto funkcí na minimum. Jak lze vidět z rozdílů dvou dříve zmíněných rozdílů profilování, i tento krok optimalizace byl vykonán. Lze si totiž všimnout, že ve výpisu 5.2 již chybí volání funkce `from_string`. V nejnovější verzi aplikace je výsledek této funkce totiž uložen do paměti po prvním volání s danými parametry.

Počet volání funkce mající za účel veškeré vyhledávání v datech (`_run_on_path`) byl omezen, právě díky výše zmíněnému vícenásobnému užití jednou načtených hodnot. Díky tomu vykonávání této kritické funkce zabralo menší část času výpočtu. Funkce také v novější verzi obdržela hodnoty pouze nutných proměnných, které byly ve většině případů převedeny na množiny hodnot namísto seznamů.

Profilování paměti

Aplikace byla také profilována nástroji `valgrind` a `massif` pro paměťovou náročnost [76]. Testování opět proběhlo na stejném stroji a využity byly tři dokumenty obsahující sto, pět set a tisíc komponentů respektive. Výsledky tohoto měření jsou uvedeny v tabulce 5.3.

Časová náročnost bez nástroje profilování

Nástroj profilování sám také zpomaluje běh aplikace, proto bylo provedeno také testování časové náročnosti pouze s využitím příkazu `time`. Testování proběhlo opět s užitím dokumentů obsahujících sto, pět set a tisíc komponentů na stejném počítači. Do výsledků je zahrnut pouze celkový čas, do něhož je započítán jak čas výpočtu, tak i čas strávený systémovými voláními. Výsledky měření jsou uvedeny v tabulce 5.4.

5.5.4 Profilování nástroje známkování

Tento nástroj byl napsán s myšlenkou optimalizace již v základu, i při využití více kuchařek (viz část 3.2.2), jejichž pravidla se mohou překrývat, je každé pravidlo vykonáno právě jednou. Pokud pravidlo v kuchařce není obsaženo, není vykonáno ani jednou.

Výsledný čas tedy lineárně závisí na počtu prvků vstupního dokumentu, jelikož nad každým prvkem musí být zavolána testovací funkce, avšak je nad každým z těchto prvků volána maximálně jednou. Také tedy závisí na použité kuchařce, čas běhu aplikace je lineárně ovlivněn počtem pravidel (při předpokladu, že pravidla obsahují přibližně stejně náročné vyhledávací výrazy).

Výsledek profilování nástroje pro známkování dat, jež bylo také uskutečněno na stejném počítači i se stejným vstupním dokumentem, je uvedeno v tabulce 5.5. Lze si zde všimnout, že velká část celkového času je užita pro volání funkcí z knihovny `importlib`. Tato knihovna obstarává načítání a spouštění pravidel definovaných jako funkce, mezi něž pak patří i validace nástroji vydanými autory standardů SBOM.

Profilování paměti

Dále bylo využito nástroje `valgrind` pro profilování využití paměti i pro nástroj známkování dokumentů SBOM. Při testování známkování dokumentů se stem, pěti sty i tisícem komponentů bylo vždy dosaženo stejného výsledku, a to využití 11,2 MB paměti.

Časová náročnost bez nástroje profilování

I nástroj pro známkování metadat byl podroben testu časové náročnosti s využitím příkazu `time`. Výsledky tohoto měření jsou uvedeny v tabulce 5.6. I v tomto případě se jedná o celkový čas běhu aplikace.

5.5.5 Implementované testy výkonnosti

Pro automatické testování výkonnosti aplikace, byly vytvořeny testy, které se automaticky spouštějí při každém příspěvku do repozitáře. Pro přípravu tyto testy vygenerují soubor o velikosti 100 komponentů ve formátech SPDX 2.3 a CycloneDX 1.6. Tyto soubory jsou pak oznámkovány a převedeny na opačný formát. Tato akce je poté ještě jednou opakována a výsledný čas je podroben zkoušce.

Test známkování selže, pokud byl průměrný čas na oznámkovaný dokument delší než půl sekundy. Test konverze selže, pokud trvá překlad průměrně více než jednu a půl sekundy.

Časy i počet opakování před vyhodnocením lze samozřejmě nastavit v kódu. Tyto hodnoty byly vybrány pro zajištění relativně spolehlivé míry, která však zbytečně nevyčerpává minuty běhu Workflows na platformě GitHub.

Tyto testy samozřejmě lze spustit i lokálně, výsledky se však budou lišit podle užitého počítače. Preferovaný způsob realizace těchto testů je právě na platformě GitHub. Pro představu, na počítači s parametry zmíněnými výše (v části 5.5.1) proběhnou tyto testy asi o pět procent rychleji než v kontejneru platformy GitHub.

5.6 Porovnání výsledků s aplikací SBOMQS

Nástroj SBOMQS umožňuje hodnocení dokumentů SBOM podle mnoha kritérií, podle více testovacích sad. Jeho vestavěné testovací sady byly vypsány v sekci 1.7.3. Tyto sady jsou víceméně ekvivalentní kuchařkám z vlastního nástroje (viz sekci 3.2.2).

Jak již bylo zmíněno v praktické části, nástroj SBOMQS neumožňuje rozlišovat pravidla pro různé typy komponentů. Tato vlastnost je v implementovaném nástroji samozřejmě dostupná, jelikož se jedná o požadovanou vlastnost pro správné hodnocení dle specifikace společnosti Red Hat.

Aplikace SBOMQS navrací desetinné číslo v rozmezí nula až deset, které odpovídá kvalitě dokumentu. Implementovaný nástroj zobrazuje známku v rozsahu A–F. Oba nástroje umožňují vykreslit výstup ve formě dobře čitelné lidmi či ve formátu JSON. Aplikace z této práce navíc podporuje formát YAML.

Pro srovnání rychlosti, dokumenty o délce sto a pět set komponentů nástroj SBOMQS ohodnotí za 0,1 s. Data obsahující tisíc komponent pak za 0,2 s. Výsledky lze porovnat s časy běhu implementované aplikace (viz tabulku 5.6). Rozdíl je pravděpodobně způsoben užitím kompilovaného jazyka pro nástroj SBOMQS, jakož i nutností složitějšího vyhledávání v metadatech implementovanou aplikací.

5.7 Porovnání výsledků překladu s aplikací Syft

Aplikace Syft podporuje překlad mezi formáty pouze částečně, jen jako experimentální vlastnost, jejíž implementace ještě není dokončena. Tento fakt je vypsán jako varování i při každém užití této funkcionality. Tento fakt však nebrání využití tohoto nástroje k účelům srovnání s implementovaným nástrojem.

Při konverzi stejného souboru, který je užitý pro jednotkové testy implementované aplikace, Syft nepředá všechna obsažená data. Hlavními problémy jsou redukce počtu identifikátorů `purl` (viz část 1.4.1) a úplné vypuštění identifikátorů `cpe` (nastíněny v části 1.4.2). Ve formátu CycloneDX Syft také vypouští pole popisující, který prvek je v SBOM míněn jako hlavní (pole `metadata.component` není přítomno). Stejně tak nejsou do výsledku překladu přeneseny všechny vztahy komponentů.

Výpis 5.2: Porovnání výsledků překladu vlastního nástroje a Syft: statická data

Vlastní aplikace:

```
...
"metadata": {
    ...
    "component": {
      "name": "Red Hat Enterprise Linux",
      "version": "9.2 EUS",
      "supplier": {
        "name": "Red Hat",
        "url": [
          "https://www.redhat.com"
        ]
      ...
      }
    },
    "tools": {
      "components": [
        ...
        {
          "name": "SBOMGrader",
          "version": "0.2.3",
          "type": "application"
        }
      ]
    }
  },
...
```

Aplikace Syft:

```
...
"metadata": {
    ...
    "tools": {
      "components": [
        {
          "type": "application",
          "author": "anchore",
          "name": ""
        }
      ]
    }
  },
...
```

Ukázky výsledků překladu jsou uvedeny ve výpisech 5.2 a 5.3. V těchto výpisech jsou některé informace vypuštěny, aby nedošlo k znepřehlednění dat. Všechna vypuštěná data jsou označena symbolem „...“. Lze si všimnout jak dříve zmíněných rozdílů v datech, tak i dodání dalších metadat aplikací Syft (začínají frází „`syft:metadata:`“). Tato data jsou specifická pouze pro tuto aplikaci a nepřinášejí do dat žádnou přidanou hodnotu (lze si všimnout, že aplikace označila uložení metadat délky „0“.

Aplikace jsou srovnatelně rychlé pro malé soubory SBOM (přibližně půl sekundy pro SBOM obsahující pouze dva komponenty), pro větší soubory je aplikace Syft rychlejší. Konverze trvala 0,5 pro metadata obsahující sto komponentů, 0,6 s pro metadata o délce pět set komponentů a 0,7 s pro data obsahující tisíc komponentů, tato data lze porovnat s daty naměřenými pro implementovanou aplikaci (viz tabulku 5.4). Kratší čas běhu nástroje Syft je způsoben mimo jiné tím, že tato aplikace je napsaná v kompilovaném programovacím jazyce [55]. Testování proběhlo opět na stejném stroji, který byl již popsán dříve.

Tab. 5.2: Výsledky profilování po optimalizaci

Volání vše/přímé	**Celk. čas** [s]	**Volaný kód** soubor:řádek(funkce)
1523/1	2,373	`{built-in method builtins.exec}`
1	2,373	`<string>:1(<module>)`
1	2,373	`<frozen runpy>:201(run_module)`
815/1	2,335	`<frozen runpy>:65(_run_code)`
1	2,335	`__main__.py:1(<module>)`
1	2,058	`__main__.py:209(main)`
1	2,055	`__main__.py:157(convert)`
1	1,633	`translation_map.py:373(convert)`
11	1,587	`translation_map.py:192(convert_and_add)`
202	1,402	`translation_map.py:58(render)`
821	0,759	`cached_python_loader.py:39(load_func)`
814	0,758	`cached_python_loader.py:24(_load_all_references)`
814	0,737	`<frozen runpy>:262(run_path)`
202	0,672	`environment.py:1275(render)`
11838	0,668	`method 'join' of 'str' objects`
811	0,659	`utils.py:187(func)`
814	0,534	`<frozen runpy>:250(_get_code_from_file)`
1213	0,525	`built-in method builtins.compile`
99	0,521	`__init__.py:1(<module>)`
2727	0,496	`<template>:4(root)`
439	0,492	`field_resolve.py:403(resolve_variables)`
206	0,486	`__init__.py:117(safe_load)`
206	0,486	`__init__.py:74(load)`
206	0,481	`constructor.py:47(get_single_data)`
...	...	...
103060/1707	0,430	`field_resolve.py:541(_run_on_path)`
...	...	...

Tab. 5.3: Paměťová náročnost konverze podle velikosti dokumentů

Počet komponentů	**Využitá paměť** [MB]
100	10,0
500	10,8
1000	12,2

Tab. 5.4: Časová náročnost konverze podle velikosti dokumentů

Počet komponentů	**Běh aplikace** [s]
100	1,5
500	6,9
1000	16,5

Tab. 5.5: Výsledky profilování nástroje pro známkování

Volání vše/přímé	**Celk.** **čas** [s]	**Volaný kód** soubor:řádek(funkce)
764/1	1,241	`built-in method builtins.exec`
1	1,241	`<string>:1(<module>)`
1	1,241	`<frozen runpy>:201(run_module)`
9/1	1,203	`<frozen runpy>:65(_run_code)`
1	1,203	`__main__.py:1(<module>)`
1	0,928	`__main__.py:209(main)`
1	0,925	`__main__.py:88(grade)`
1	0,737	`cookbook_bundles.py:104(__call__)`
1	0,629	`cookbook_bundles.py:97(ruleset)`
2	0,629	`cookbooks.py:139(ruleset)`
2	0,629	`cookbooks.py:157(initialize)`
4	0,629	`rules.py:142(from_file)`
81/17	0,525	`<frozen importlib._bootstrap>:13(_find_and_load)`
...	...	...

Tab. 5.6: Časová náročnost známkování podle velikosti dokumentů

Počet komponentů	**Běh aplikace** [s]
100	0,5
500	0,8
1000	1,4

Výpis 5.3: Porovnání výsledků překladu vlastního nástroje a Syft: komponent

Vlastní aplikace:

```
...
"name": "openssl",
"bom-ref": ...,
"purl": ...,
"type": "library",
"version": "3.0.7-18.el9_2",
"supplier": {
  "name": "Red Hat",
  "url": [
    "https://www.redhat.com"
  ]
},
"licenses": [
  {
    "license": {
      "name": "Apache-2.0",
      "acknowledgement": "declared"
    }
  }
],
"evidence": {
  "identity": [
    {
      "field": "purl",
      "concludedValue": ...
    },
    {
      "field": "purl",
      "concludedValue": ...
    },
  ...
  ]
}
...
```

Aplikace Syft:

```
...
"bom-ref": ...,
"type": "library",
"name": "openssl",
"version": "3.0.7-18.el9_2",
"licenses": [
{
  "license": {
    "id": "Apache-2.0"
  }
}
],
"purl": ...,
"properties": [
  {
    "name": "syft:package:type",
    "value": "rpm"
  },
  {
    "name": "syft:package:metadataType",
    "value": "rpm-db-entry"
  },
  {
    "name": "syft:metadata:size",
    "value": "0"
  }
]
...
```

Závěr

Tato práce byla zaměřena na praktickou implementaci nástroje pro účely společnosti Red Hat. Tento nástroj umožňuje známkování a překlad metadat SBOM podle kritérií vydaných touto společností.

Aplikaci bylo nutné vytvořit v reakci na novou legislativu týkající se ochrany softwarového dodavatelského řetězce. Tato legislativa se v aktuální době dotýká pouze software vydaného pro vládní instituce v USA, avšak v roce 2027 bude podobná legislativa účinná i v Evropské unii. Všechny tyto skutečnosti byly uvedeny v teoretické části práce.

Nástroj byl vytvořen v programovacím jazyce Python. Jedná se o terminálovou aplikaci, podporující dvě specifikace dokumentů SBOM, jimiž jsou SPDX 2.3 a CycloneDX 1.6, které jsou také jedinými standardy popsány ve specifikaci společnosti Red Hat. Aplikace obsahuje více než 3700 řádků kódu a 1600 řádků konfiguračních souborů. Je také přiložen soubor umožňující snadnou kontejnerizaci aplikace.

Nástroj známkování obsahuje vlastní specifikaci pravidel, jimiž jsou předložené dokumenty SBOM testovány. Pravidla jsou pak rozdělena podle síly, s níž mají být aplikována na jednotlivé typy dokumentů. Také je možné zvolit formát výstupu z běhu aplikace, díky čemuž je umožněno další strojové zpracování výstupu.

Nástroj pro konverzi dokumentů obsahuje vestavěnou mapu konverzí, která specifikuje, jak data mezi formáty převádět podle specifikace ze společnosti Red Hat. Tato mapa je plně nahraditelná uživatelským vstupem, lze tedy vytvářet i definice vlastní.

Přínos vytvořené aplikace spočívá v zaručení vydávání kvalitních dokumentů SBOM, což ulehčí proces reakce na bezpečnostní incidenty způsobené zranitelnostmi ve společnosti Red Hat. To je velice důležité jak z pohledu zákazníků společnosti, tak i pro vládní organizace.

Aplikace také umožňuje sjednotit nakládání s metadaty pod jediný standard, aniž by bylo nutné měnit velkou část již existující infrastruktury společnosti Red Hat. V případě změny specifikace stačí změnit konfigurační soubory vytvořené aplikace, jelikož je nástroj plně rozšířitelný.

Implementovaná aplikace byla zveřejněna na platformy GitHub a PyPI, odkud je možné ji instalovat jediným příkazem. Aplikace také byla silně optimalizována a testována, aby bylo její užití možné i v kontejnerech některých řešení CI/CD infrastruktury společnosti Red Hat.

Literatura

[1] Michelle Jump, Art Manion, and Framing Working group. Framing software component transparency: Establishing a common software bill of materials (sbom). *NTIA*, pages 1–34, 2021. URL: https://www.ntia.gov/sites/default/files/publications/ntia_sbom_framing_2nd_edition_20211021_0.pdf.

[2] Tony Turner. Sbom formats explained and compared. *Fossa*, 2024. URL: https://fossa.com/blog/sbom-formats-compared-explained/.

[3] Gede Artha Azriadi Prana, Abhishek Sharma, Lwin Khin Shar, Darius Foo, Andrew E. Santosa, Asankhaya Sharma, and David Lo. Out of sight, out of mind? how vulnerable dependencies affect open-source projects. *Empirical Software Engineering*, 26(4):59, Apr 2021. doi:10.1007/s10664-021-09959-3.

[4] The Linux Foundation. About spdx. *The Linux Foundation*, 2023, 2023. URL: https://spdx.dev/about/overview/.

[5] OWASP Foundation. Cyclonedx history. *CycloneDX*, 2024. URL: https://cyclonedx.org/about/history/.

[6] Adrian Bridgewater. Linux foundation eases open source licensing woes. *ComputerWeekly.com*, 2011. URL: https://web.archive.org/web/20210820144000/https://www.computerweekly.com/blog/Open-Source-Insider/Linux-Foundation-eases-open-source-licensing-woes.

[7] Zákon o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), 2000. URL: https://www.zakonyprolidi.cz/cs/2000-121.

[8] Philippe Ombredanne. Free and open source software license compliance: Tools for software composition analysis. *Computer*, 53(10):105–109, 2020. doi:10.1109/MC.2020.3011082.

[9] Směrnice evropského parlamentu a rady 96/9/es ze dne 11. března 1996 o právní ochraně databází, 1996. URL: https://eur-lex.europa.eu/legal-content/CS/TXT/?uri=CELEX:31996L0009.

[10] CC0 1.0 Univerzální. URL: https://creativecommons.org/publicdomain/zero/1.0/deed.cs.

[11] The Linux Foundation. *Composition of an SPDX document*, 2022. URL: https://spdx.github.io/spdx-spec/v2.3/composition-of-an-SPDX-document/.

[12] Cloudflare. What is a supply chain attack? *Cloudflare*, 2024. URL: https://www.cloudflare.com/learning/security/what-is-a-supply-chain-attack/.

[13] Rodrigo Freire. Understanding red hat's response to the xz security incident. *Red Hat Blog*, 2024. URL: https://www.redhat.com/en/blog/understanding-red-hats-response-xz-security-incident.

[14] Narges Yousefnezhad and Andrei Costin. Understanding sboms in real-world systems – a practical devops/secops perspective. In Boris Shishkov, editor, *Business Modeling and Software Design*, pages 293–304, Cham, 2024. Springer Nature Switzerland.

[15] Executive order on improving the nation's cybersecurity, 2021. URL: https://www.whitehouse.gov/briefing-room/presidential-actions/2021/05/12/executive-order-on-improving-the-nations-cybersecurity/.

[16] National Institute of Standards and Technology, USA. *Software Security in Supply Chains: Software Bill of Materials (SBOM)*, 2022. URL: https://www.nist.gov/itl/executive-order-14028-improving-nations-cybersecurity/software-security-supply-chains-software-1.

[17] National Telecommunications and Information Administration, USA. *SBOM Options and Decision Points*, 2021. URL: https://www.ntia.gov/sites/default/files/publications/sbom_options_and_decision_points_20210427-1_0.pdf.

[18] The United States Department of Commerce, USA. *The Minimum Elements For a Software Bill of Materials (SBOM)*, 2021. URL: https://www.ntia.gov/sites/default/files/publications/sbom_minimum_elements_report_0.pdf.

[19] Akt o kybernetické odolnosti zveřejněn, 2024. URL: https://nukib.gov.cz/cs/infoservis/aktuality/2192-akt-o-kyberneticke-odolnosti-zverejnen/.

[20] Nařízení evropského parlamentu a rady (eu) 2024/2847 ze dne 23. října 2024 o horizontálních požadavcích na kybernetickou bezpečnost produktů s digitálními prvky a o změně nařízení (eu) č. 168/2013 a (eu) 2019/1020 a

směrnice (eu) 2020/1828 (akt o kybernetické odolnosti) (text s významem pro ehp), 2024. URL: https://eur-lex.europa.eu/legal-content/CS/TXT/?uri=CELEX:32024R2847.

[21] Evropská komise. Eu cyber resilience act. *European Comission*, 2024. URL: https://digital-strategy.ec.europa.eu/en/policies/cyber-resilience-act.

[22] Bundesamt für Sicherheit in der Informationstechnik. *Cyber Resilience Requirements for Manufacturers and Products - Part 2: Software Bill of Materials (SBOM)*, 2.0.0 edition. URL: https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TR03183/BSI-TR-03183-2-2_0_0.pdf?__blob=publicationFile&v=3.

[23] NTIA. *Types of Software Bill of Material (SBOM) Documents*, 2023. URL: https://www.cisa.gov/resources-tools/resources/types-software-bill-materials-sbom.

[24] ISO. *Information technology — SPDX® Specification V2.2.1*. URL: https://www.iso.org/obp/ui/en/#iso:std:81870:en.

[25] The Linux Foundation. *The System Package Data Exchange® (SPDX®) Specification Version 3.0.1*, 2024. URL: https://spdx.github.io/spdx-spec/v3.0.1/.

[26] Noah Lehman. Spdx 3.0 revolutionizes software management in systems with enhanced functionality and streamlined use cases. *The Linux Foundation*, 2024. URL: https://www.linuxfoundation.org/press/spdx-3-revolutionizes-software-management-in-systems-with-enhanced-functionality-and-streamlined-use-cases.

[27] World Wide Web Consortium. *JSON-LD 1.1*, 2020. URL: https://www.w3.org/TR/json-ld11/.

[28] OWASP Foundation. *CycloneDX One Pager*. URL: https://cyclonedx.org/guides/CycloneDX%20One%20Pager.pdf.

[29] OWASP Foundation. *CycloneDX v1.6 JSON Reference*, 2024. URL: https://cyclonedx.org/docs/1.6/json/.

[30] ISO/IEC. *Software asset management*. URL: https://www.iso.org/obp/ui/en/#iso:std:iso-iec:19770:-2:ed-2:v2:en.

[31] Tom Alrich, Chris Clark, and Patrick Dwyer et al. Survey of existing sbom formats and standards. *NTIA Multistakeholder Process on Software Component Transparency Standards and Formats Working Group*, 2021. URL: https://www.ntia.gov/sites/default/files/publications/sbom_formats_survey-version-2021_0.pdf.

[32] NIST. *Guidelines for the Creation of Interoperable Software Identification (SWID) Tags*. doi:10.6028/NIST.IR.8060.

[33] Martin Prpič. Understanding sboms. *Red Hat Security Data Guidelines*, 2024. URL: https://redhatproductsecurity.github.io/security-data-guidelines/sbom/.

[34] SPDX. *Package URL specification v1 (Normative)*. URL: https://spdx.github.io/spdx-spec/v3.0.1/annexes/pkg-url-specification/.

[35] NIST. *Security Content Automation Protocol*. URL: https://csrc.nist.gov/projects/security-content-automation-protocol/specifications/cpe.

[36] NIST. *Common Platform Enumeration: Naming Specification Version 2.3*. doi:10.6028/NIST.IR.7695.

[37] NIST. *Common Platform Enumeration: Name Matching Specification Version 2.3*. doi:10.6028/NIST.IR.7696.

[38] NIST. *Common Platform Enumeration: Dictionary Specification Version 2.3*. doi:10.6028/NIST.IR.7697.

[39] NIST. *Common Platform Enumeration: Applicability Language Specification Version 2.3*. doi:10.6028/NIST.IR.7698.

[40] OWASP. *Authoritative Guide to SBOM*. URL: https://cyclonedx.org/guides/OWASP_CycloneDX-Authoritative-Guide-to-SBOM-en.pdf.

[41] The Linux Foundation. *SPDX License List*. URL: https://spdx.org/licenses/.

[42] Python.org. *Improving License Clarity with Better Package Metadata*. URL: https://peps.python.org/pep-0639/.

[43] The MIT License. URL: https://mit-license.org/.

[44] Cybersecurity and Infrastructure Security Agency. Vulnerability exploitability exchange (vex) use case. *CISA*, 2022. URL:

https://www.cisa.gov/resources-tools/resources/vulnerability-exploitability-exchange-vex-use-case-document-april-2022.

[45] NTIA. *Vulnerability-Exploitability eXchange (VEX) – An Overview*, 2021. URL: https://www.ntia.gov/files/ntia/publications/vex_one-page_summary.pdf.

[46] Cybersecurity and Infrastructure Security Agency. *Minimum Requirements for Vulnerability Exploitability eXchange (VEX)*, 2023. URL: https://www.cisa.gov/resources-tools/resources/minimum-requirements-vulnerability-exploitability-exchange-vex.

[47] OWASP Foundation. *Vulnerability Exploitability eXchange (VEX)*. URL: https://cyclonedx.org/capabilities/vex/.

[48] OpenVEX. *Welcome to OpenVEX!*, 2023. URL: https://github.com/openvex.

[49] OASIS OPEN. *Common Security Advisory Framework Version 2.0*, 2022. URL: https://docs.oasis-open.org/csaf/csaf/v2.0/os/csaf-v2.0-os.html.

[50] Arushi Arora. Sok: A framework for and analysis of software bill of materials tools technical report. Technical report, OSTI.GOV, 2022. URL: https://doi.org/10.2172/2204407.

[51] Gregorio Dalia, Corrado Aaron Visaggio, Andrea Di Sorbo, and Gerardo Canfora. Sbom ouverture: What we need and what we have. In *Proceedings of the 19th International Conference on Availability, Reliability and Security*, ARES '24, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3664476.3669975.

[52] Gary O'Neall, Henk Birkholz, Jilayne Lovejoy, Kyle E. Mitchell, Maximilian Huber Philippe Ombredanne, Sebastian Schuberth, Steve Winslow, Thomas Steenbergen, and Alexios Zavras. Spdx github organization. URL: https://github.com/spdx/.

[53] Meret B., Armin Tänzer, et al. Spdx – tools-python, 2024. URL: https://github.com/spdx/tools-python.

[54] Patrick Dwyer, Andreas Hilti, and Michael Tsfoni. Cyclonedx cli tool. URL: https://github.com/CycloneDX/cyclonedx-cli.

[55] Alex Goodman, Christopher Angelo Phillips, Keith Zantow, Dan Luhring, Weston Steimel, and William Murphy. Anchore syft. URL: https://github.com/anchore/syft/.

[56] Python.org. *Storing project metadata in pyproject.toml*, 2020. URL: https://peps.python.org/pep-0621/.

[57] Python.org. *pyproject.toml specification*, 2025. URL: https://packaging.python.org/en/latest/specifications/pyproject-toml/#pyproject-toml-spec.

[58] Pdm, 2025. URL: https://pdm-project.org/latest/.

[59] Python Time Complexity of Data Types. URL: https://wiki.python.org/moin/TimeComplexity.

[60] *Key words for use in RFCs to Indicate Requirement Levels*, 1997. URL: https://www.rfc-editor.org/rfc/rfc2119.

[61] Martin Prpič and Tim Waugh. Red hat product security, 2025. URL: https://github.com/RedHatProductSecurity/security-data-guidelines/tree/main.

[62] Build more. break less. empower others., 2025. URL: https://json-schema.org/.

[63] Will McGugan. Welcome to rich's documentation! URL: https://rich.readthedocs.io/en/stable/index.html#.

[64] Armin Ronacher, David Lord, et al. Jinja, 2007. URL: https://jinja.palletsprojects.com/en/stable/.

[65] About openshift pipelines, 2025. URL: https://docs.redhat.com/en/documentation/red_hat_openshift_pipelines/1.18/html/about_openshift_pipelines/index.

[66] Get started with gitlab ci/cd, 2025. URL: https://docs.gitlab.com/ci/.

[67] About workflows, 2025. URL: https://docs.github.com/en/actions/writing-workflows/about-workflows.

[68] Quay container registry, 2025. URL: https://quay.io/.

[69] Docker hub container image library, 2025. URL: https://hub.docker.com/.

[70] Bernát Gábor, Jürgen Gmach, et al. tox - automation project, 2025. URL: https://tox.wiki/en/latest/.

[71] Łukasz Langa, Jelle Zijlstra, et al. The uncompromising code formatter, 2025. URL: https://black.readthedocs.io/en/stable/index.html.

[72] Adrien Vergé. yamllint, 2025. URL: https://github.com/adrienverge/yamllint.

[73] Jukka Lehtosalo, Michael J. Sullivan, et al. Mypy, 2025. URL: https://www.mypy-lang.org/.

[74] Bruno Oliveira, Holger Krekel, et al. pytest, 2025. URL: https://docs.pytest.org/en/stable/.

[75] Python. *The Python Profilers*, 2025. URL: https://docs.python.org/3/library/profile.html.

[76] Massif, 2025. URL: https://valgrind.org/docs/manual/ms-manual.html.

Seznam symbolů a zkratek

BOM Bill of Material

CI/CD Continuous Integration and Continuous Delivery

CISA Cybersecurity & Infrastructure Security Agency

CLI Command Line Interface

CSAF Common Security Advisory Framework

CPE Common Platform Enumeration

CVE Common Vulnerabilities and Exposures

FOSS Free and Open Source Software

HBOM Hardware Bill of Material

IRI Internationalized Resource Identifier

JSF JSON Signature Format

JSON JavaScript Object Notation

JSON-LD JSON Linked Data

NIST National Institute of Standards and Technology

NTIA National Telecommunication and Information Administration

NÚKIB Národní úřad pro kybernetickou a informační bezpečnost

NVD National Vulnerablity Database

PEP Python Enhancement Proposal

PURL Package URL

PyPA Python Packaging Authority

PyPI Python Packaging Index

RDF Resource Description Framework

RPM RPM Package Manager

SBOM Software Bill of Material

SPDX System Package Data Exchange

SWID SoftWare IDentification

URI Uniform Resource Identifier

URL Uniform Resource Locator

VCS Version Control System

VEX Vulnerability Exploitability eXchange

WFN Well-Formed Name

Seznam příloh

A Popis specifikací SBOM dokumentů

Pro přehlednost byl popis standardů dokumentů SBOM vyčleněn z textu do příloh. Nejdetailnější specifikace je však vždy zveřejněná na stránkách dané specifikace a v této příloze jsou některé údaje zjednodušeny či vypuštěny. Povinná pole jsou zanesena **tučně** a pole, které mohou obsahovat více hodnot, jsou označena ***kurzívou***. Atributy objektů jsou pak označeny tak, že jim v tabulce předřazena tečka. Pokud je teček více, znamená to, že se jedná o atribut atributu (počet teček značí zanoření).

A.1 Specifikace SPDX 2.3

V této verzi jsou dokumenty SBOM poměrně jasně specifikované a standard není příliš obsáhlý, proto je možné pomocí tabulek uvést všechny důležité detaily pomocí tabulek. To však neplatí pro další standardy zde uvedené. Pro porovnání, délka JSON schématu SPDX 2.3[1] je okolo 750 řádků, schéma stejného formátu CycloneDX[2] má bezmála 5700 řádků a JSON-LD schéma pro SPDX 3.0.1[3] je dlouhé téměř 6000 řádků. Kvůli tomuto faktu bude uveden podrobný popis pouze pro specifikaci SPDX 2.3.

A.2 Specifikace SPDX 3.0 Lite

Pro přiblížení této minimální verze SPDX verze 3.0 jsou zde uvedena alespoň její základní pravidla ve formě tabulky tříd v klíči `@graph` dokumentu (viz tabulku A.9 a následující). Všechny třídy a atributy jsou uvedeny `strojopisem`. Pro více detailů o každém atributu je vhodné přímo navštívit online dokumentaci [25], jelikož z důvodu rozsahu není možné uvést všechny detaily do textu této práce.

V tabulkách se také vyskytují vynechávky (např. `/.../AnyLicenseInfo`). Ty byly vytvořeny v důsledku nedostatku místa na stránce tohoto dokumentu. Všechny objekty týkající se licencí spadají do skupiny `/SimpleLicensing`,

[1] Dostupné z `https://github.com/spdx/tools-java/blob/master/resources/spdx-schema-v2.3.json`

[2] Dostupné z `https://github.com/CycloneDX/specification/blob/master/schema/bom-1.6.schema.json`

[3] Dostupné z `https://github.com/spdx/tools-java/blob/master/resources/spdx-schema-v3.0.1.json`

Tab. A.1: Informace o vytvoření dokumentu SPDX 2.3

Název	**Formát**	**Další údaje a komentáře**
spdxVersion	SPDX-2.3	Verze dokumentu.
dataLicense	Název licence	Licence k datům v dokumentu, k nalezení v listu licencí SPDX
SPDXID	SPDXRef-DOCUMENT	Identifikace celého dokumentu pouze pro reference v rámci dokumentu.
name	Text	Identifikace celého dokumentu lidsky přívětivou formou i pro účely přesahující tento dokument.
documentNamespace	URI bez sekcí	URI nesmí obsahovat sekce (oddělené znakem #), hodnota musí být unikátní pro jakýkoliv SBOM dokument i jeho verzi.
externalDocumentRefs	Objekt	Reference na objekty v jiných dokumentech.
.**spdxDocument**	SPDXID	Identifikátor přiřazený externímu dokumentu.
.**externalDocumentId**	URI	Unikátní identifikátor k dohledání dokumentu.
.**checksum**	Objekt	Otisk referencovaného souboru.
..**algorithm**	Jedna z hodnot	Např. SHA1.
..**checksumValue**	HEXA	Hexadecimální hodnota.
licenseListVersion	např. 3.25	Verze listu známých licencí.
creators	[Typ]: [další info]	Typem může být například Tool či Person.
created	ISO 8601	Například 1970-01-01T00:00:00Z
comment	Text	Komentář tvůrce.

A.3 Specifikace CycloneDX 1.6

Dokumenty vydané pod touto specifikací mají pevnou formu, avšak všechny kořenové objekty obsahují velké množství objektů zanořených do několika vrstev, navíc se mohou vyskytovat i na více místech. Z důvodu přehlednosti je tedy v této příloze uveden pouze základní přehled specifikace. Sekce CycloneDX 1.6 dokumentu jsou popsány v tabulce A.13.

Tab. A.2: Informace o balíčcích v dokumentu SPDX 2.3

Název	**Formát**	**Další údaje a komentáře**
name	Text	Název balíčku.
SPDXID	SPDXRef-[id]	Unikátní identifikátor v rámci tohoto SPDX dokumentu.
versionInfo	Text	Verze balíčku.
packageFileName	Text	Název souboru s balíčkem, např. python3.12.zip
supplier	[typ]: [kontakt]	Typ může být Person nebo Organization. Také je možné použít hodnotu NOASSERTION.
originator	[typ]: [kontakt]	Obdobný formát jako pole PackageSupplier.
downloadLocation	URI nebo VCS URL	Místo stažení balíčku, může nabývat také NOASSERTION nebo NONE.
filesAnalyzed	true/false	Indikuje, zda byly soubory v balíčku analyzovány. Výchozí hodnota je true.
packageVerificationCode *.p...V...C...ExcludedFiles* .p...V...C...Value	Objekt Text Text	Unikátní identifikátor podle souborů v balíčku. Soubory neúčastnící se na výpočtu identifikátoru. Hexadecimální hodnota.
checksums .algorithm .checksumValue	Objekt Jedna z hodnot HEXA	Otisk (hash) balíčku. Např. SHA1. Hexadecimální hodnota otisku.
homepage	URL	Domovská stránka balíčku.

Tab. A.3: Informace o balíčcích v dokumentu SPDX 2.3 – pokračování

Název	**Formát**	**Další údaje a komentáře**
`sourceInfo`	Text	Další informace o zdroji balíčku.
`licenseConcluded`	Název licence	Také může obsahovat `NOASSERTION` nebo logický výraz s více licencemi.
`licenseInfoFromFiles`	Název licence	Informace o licencích všech souborů v balíčku.
`licenseDeclared`	Název licence	Licence specifikovaná autory.
`licenseComments`	Text	Komentáře o licencích.
`copyrightText`	Text	Nebo také `NOASSERTION`. Text o procesu k získání hodnoty `ConcludedLicenseField`.
`summary`	Text	Popis balíčku.
`description`	Text	Podrobnější popis balíčku.
`comment`	Text	Komentáře k balíčku.
`externalRefs`	Objekt	Reference na externí dokumenty relevantní k balíčku.
`.referenceCategory`	Jedna z hodnot	Kategorie odkazu, např. `SECURITY`.
`.referenceLocator`	Text	Unikátní identifikátor.
`.referenceType`	Text	Např. `cpe23Type`.
`.comment`	Text	Popis externích vztahů. Ke každé referenci maximálně jeden komentář.
`attributionTexts`	Text	Zásluhy týkající se balíčku.
`primaryPackagePurpose`	Jedna z hodnot	Hodnoty jsou například `FRAMEWORK`, `LIBRARY`...
`releaseDate`	ISO 8601	Čas vydání balíčku.
`buildDate`	ISO 8601	Čas vytvoření balíčku.
`validUntilDate`	ISO 8601	Čas, kdy balíčku končí podpora.

Tab. A.4: Informace o souborech v dokumentu SPDX 2.3

Název	**Formát**	**Další údaje a komentáře**
fileName	./[soubor]	Název souboru.
SPDXID	SPDXRef-[id]	Unikátní identifikátor v rámci SPDX dokumentu.
fileTypes	Jedna z hodnot	Hodnoty jsou např. BINARY, IMAGE nebo TEXT.
checksums .algorithm .checksumValue	Objekt Jedna z hodnot HEXA	Otisk souboru (hash). Např. SHA1. Hexadecimální hodnota otisku.
licenseConcluded	Název licence	Je také možné použít NOASSERTION nebo NONE.
licenseInfoInFiles	Název licence	Specifikuje, které licence byly opravdu v souboru nalezeny.
licenseComments	Text	Komentář k licencím.
copyrightText	Text	Text stanovující autorské právo nebo také NOASSERTION nebo NONE.
comment	Text	Komentáře k souoru.
noticeText	Text	Pole pro upozornění z licencí nalezené v dokumentu.
fileContributor	Text	Přispěvatelé k souboru.
attributionTexts	Text	Zásluhy týkající se souboru.

Tab. A.5: Informace o útržcích v dokumentu SPDX 2.3

Název	**Formát**	**Další údaje a komentáře**
SPDXID	SPDXRef-[id]	Unikátní identifikátor v rámci SPDX dokumentu.
snippetFromFile	SPDXID	Reference na SPDX objekt souboru obsahující tento útržek. Také je možné odkazovat na jiné soubory zmíněné v poli ExternalRef.
ranges	Objekt	Rozsah, na kterém se útržek vyskytuje v souboru.
.startPointer	Objekt	Ukazatel na začátek útržku.
..**reference**	SPDXID	Reference na soubor.
..offset	Číslo	Počet bajtů od začátku.
..lineNumber	Číslo	Číslo řádku.
.endPointer	Objekt	Ukazatel na začátek útržku.
..**reference**	SPDXID	Reference na soubor.
..offset	Číslo	Počet bajtů od začátku.
..lineNumber	Číslo	Číslo řádku.
licenseConcluded	Název licence	Licence k útržku zjištěná tvůrcem dokumentu. Také je možné použít NOASSERTION nebo NONE.
licenseInfoInSnippets	Název licence	Licence obsažená v útržku. Také je možné použít NOASSERTION neo NONE.
licenseComments	Text	Komentáře k licencování útržku.
copyrightText	Text	Pole pro upozornění z licencí nalezené v útržku.
comment	Text	Komentáře k útržku.
name	Text	Název útržku.
attributionTexts	Text	Zásluhy týkající se útržku.

Tab. A.6: Informace o licencích v dokumentu SPDX 2.3

Název	**Formát**	**Další údaje a komentáře**
licenseID	LicenseRef-[id]	Povinné pole, pokud licence není vedena v listu licencí SPDX.
extractedText	Text	Znění licence. Povinné, pokud bylo uvedeno LicenseID.
name	Text	Název licence, pokud není na SPDX listu licencí.
crossRefs .**url** .Další parametry	Objekt URL	Odkazy na zdroje licence. Lze specifikovat více informací o vlastnostech URL odkazu.
comment	Text	Komentáře k licenci.

Tab. A.7: Informace o vztazích v dokumentu SPDX 2.3

Název	**Formát**	**Další údaje a komentáře**
relatedSpdxElement	SPDXID	Související element.
spdxElementId	SPDXID	Element, vůči kterému je souvislost definována.
relationshipType	Jedna z hodnot	Popis vztahu prvků. Např. VARIANT_OF nebo COPY_OF.
comment	Text	Komentář.

Tab. A.8: Informace o anotacích v dokumentu SPDX 2.3

Název	**Formát**	**Další údaje a komentáře**
annotator	[Typ]: [další info]	Typem může být například Tool či Person.
annotationDate	ISO 8601	Čas anotace.
annotationType	Jedna z hodnot	Může nabývat hodnoty buď OTHER nebo REVIEW.

Tab. A.9: Popis specifikace SPDX 3.0 Lite

Název	**Formát**	**Další údaje a komentáře**
`/Core/SpdxDocument`	Objekt	Popisuje objekt, jehož jsou ostatní elementy součástí.
`.creationInfo`	`/Core/CreationInfo`	Informace o vytvoření objektu.
`.element`	`/Core/Element`	Musí obsahovat alespoň jeden `/Core/Bom`.
`.rootElement`	`/Core/Element`	Měl by obsahovat alespoň jeden `/Core/Bom`.
`.spdxId`	URI	Identifikátor.
`.comment`	Text	Komentář.
`.dataLicense`	`/.../AnyLicenseInfo`	Licence k datům.
`.name`	Text	Název dokumentu.
`.namespaceMap`	`/Core/NamespaceMap`	Mapa předpon identifikátorů.
`.verifiedUsing`	`/Core/Hash`	Ověření integrity.
`/Software/Sbom`	Objekt	Objekt SBOM.
`.creationInfo`	`/Core/CreationInfo`	Informace o vytvoření objektu.
`.element`	`/Core/Element`	Musí obsahovat alespoň jeden objekt `/Software/Package`
`.rootElement`	`/Core/Element`	Měl by obsahovat alespoň jeden objekt `/Software/Package`
`.spdxId`	URI	Identifikátor.
`.sbomType`	Jedna z hodnot	Hodnoty podle tabulky 1.2, malým písmem.

Tab. A.10: Popis specifikace SPDX 3.0 Lite – část 2

`/Software/Package`	Objekt	Balíček. Každý objekt tohoto typu musí být zmíněn jako atribut `from` objektu `/Core/Relationship` typu `hasConcludedLicense` a `hasDeclaredLicense`.
`.copyrightText`	Text	Znění autorských nároků.
`.creationInfo`	`/Core/CreationInfo`	Informace o vytvoření objektu.
`.name`	Text	Název balíčku.
`.packageVersion`	Text	Verze balíčku.
`.spdxId`	URI	Identifikátor.
`.suppliedBy`	`/Core/Agent`	Která entita balíček vydala.
`.attributionText`	Text	Uznání k balíčku.
`.builtTime`	`/Core/DateTime`	Čas sestavení.
`.comment`	Text	Komentář.
`.downloadLocation`	URI	Objekt musí mít tento atribut nebo atribut `.packageUrl`.
`.homepage`	URI	Domovská stránka balíčku.
`.originatedBy`	`/Core/Agent`	Tvůrce balíčku.
`.packageUrl`	URI	Objekt musí mít tento atribut nebo atribut `.downloadLocation`.
`.releaseTime`	`/Core/DateTime`	Čas vydání balíčku.
`.supportLevel`	Jedna z hodnot	Informace o podpoře.
`.validUntilTime`	`/Core/DateTime`	Čas vypršení platnosti balíčku.
`.verifiedUsing`	`/Core/Hash`	Ověření autenticity balíčku.
`/Core/Hash`	Objekt	Otisk.
`.algorithm`	Jedna z hodnot	Algoritmus otisku.
`.hashValue`	Text	Hodnota otisku.
`.comment`	Text	Komentář.

Tab. A.11: Popis specifikace SPDX 3.0 Lite – část 3

`/.../LicenseExpression`	Objekt	Licence.
`.creationInfo`	`/Core/CreationInfo`	Informace o vytvoření objektu.
`.licenseExpression`	Text	Obsahuje licenční výraz SPDX.
`.spdxId`	URI	Identifikátor.
`.licenseListVersion`	`/Core/SemVer`	Verze listu licencí.
`/.../SimpleLicensingText`	Objekt	Licenční informace mimo list SPDX.
`.creationInfo`	`/Core/CreationInfo`	Informace o vytvoření objektu.
`.licenseText`	Text	Text k licenci.
`.spdxId`	URI	Identifikátor.
`.comment`	Text	Komentář.
`/Core/Agent`	Objekt	Entita (člověk, služba, organizace...).
`.creationInfo`	`BlankNode`)	Informace o vytvoření objektu, nemělo by se vyplňovat.
`.name`	Text	Název entity.
`.spdxId`	URI	Identifikátor.
`.externalIdentifier`	`/Core/ExternalIdentifier`	Externí identifikátor entity.
`/Core/CreationInfo`	Objekt	Informace o vytvoření.
`.created`	`/Core/DateTime`	Čas vytvoření.
`.createdBy`	`/Core/Agent`	Entita, která objekt vytvořila.
`.specVersion`	`/Core/SemVer`	Verze objektu.
`.comment`	Text	Komentář.

Tab. A.12: Popis specifikace SPDX 3.0 Lite – část 4

`/Core/ExternalIdentifier`	Objekt	Ukazatel mimo rozsah dokumentu.
`.externalIdentifierType`	Jedna z hodnot	Např. `cpe22`, `cve`...
`.identifier`	Text	Vlastní identifikátor.
`/Core/NameSpaceMap`	Objekt	Mapa předpon identifikátorů.
`.namespace`	URI	Část URI, která má být zastoupena.
`.prefix`	Text	Náhrada za URI.
`/Core/Relationship`	Objekt	Vztahy mezi objekty.
`.creationInfo`	`/Core/CreationInfo`	Informace o vytvoření objektu.
`.from`	`/Core/Element`	Element, první strana vztahu.
`.relationshipType`	Jedna z hodnot	Např. `hasDeclaredLicense`.
`.spdxId`	URI	Identifikátor.
`.to`	`/Core/Element`	Druhá strana vztahu.

Tab. A.13: Struktura dokumentu CycloneDX 1.6 [29]

Název	**Formát**	**Další údaje a komentáře**
bomFormat	CycloneDX	Označení specifikace.
specVersion	1.6	Verze specifikace.
serialNumber	urn:uuid:<uuid>	Identifikátor dokumentu.
version	Číslo	Verze dokumentu.
metadata	Objekt	Informace o dokumentu.
components	Objekt	List softwarových a hardwarových součástí popisované aplikace.
services	Objekt	List služeb využitých popisovanou aplikací.
externalReferences	Objekt	Zdroje relevantní k BOM.
dependencies	Objekt	Závislosti komponentů a služeb. Musí být uvedeny i prázdná pole, pokud závislosti neexistují (nevyplnění značí neznámé závislosti).
compositions	Objekt	Popisují dílčí části (komponenty, služby, závislosti) a jejich kompletnost.
vulnerabilities	Objekt	Slabiny v komponentech či službách.
annotations	Objekt	Komentáře k jakékoli části dokumentu.
formulation	Objekt	Popisuje, jak byl vytvořen či nasazen komponent či služba.
declarations	Objekt	Popisují dodržování standardizací či tvrzení s důkazy.
definitions	Objekt	Vlastní definice, které mohou být zmíněné v ostatních částech dokumentu.
properties	Objekt	Definice vlastních hodnot, které nejsou ve specifikaci podporovány.
signature	JSF	Digitální podpis dokumentu. JSF je standard vytvořený k tomuto účelu.

B Obsah elektronické přílohy

Elektronická příloha obsahuje implementovaný nástroj pro hodnocení kvality SBOM. Tento nástroj je dostupný také na platformě GitHub (viz část 2). Výpis přiložených souborů je zjednodušen, ve skutečnosti je odevzdáno více než sto souborů.

- `/`
  - `README.md` — Dokumentace
  - `pyproject.toml` — Metadata o projektu
  - `tox.ini` — Nastavení nástroje `tox`
  - `LICENSE` — Licence MIT
  - `pdm.lock` — Soubor se zámkem závislostí
  - `Dockerfile` — Soubor pro tvorbu obrazu kontejneru
  - `sbomgrader`
    - `__main__.py` — Definice spustitelného balíčku, obsahuje příkazové rozhraní
    - `core` — Balíček jádra aplikace
      - `cached_python_loader.py` — Modul načítání testovacích funkcí
      - `field_resolve.py` — Modul pro vyhledávání v datech
      - `logging.py` — Modul pro nastavení výpisů z aplikace
      - `utils.py` — Modul pro práci se soubory
      - `documents.py` — Modul reprezentace metadat
      - `definitions.py` — Modul statických definic
      - `formats.py` — Modul definic formátů SBOM
      - `enums.py` — Modul statických definic výčtů
    - `grade` — Balíček známkování metadat SBOM
      - `choose_cookbooks.py` — Modul načítání kuchařek
      - `rule_loader.py` — Modul načítání pravidel
      - `rules.py` — Modul s obsluhou logiky pravidel
      - `cookbooks.py` — Modul s obsluhou logiky kuchařek
      - `cookbook_bundles.py` — Modul s obslohou logiky balíků kuchařek
    - `translate` — Balíček konverze metadat SBOM
      - `choose_map.py` — Modul načítání map překladů
      - `translation_map.py` — Modul s obsluhou logiky map překladů
      - `prune.py` — Modul čištění nevalidních dat
    - `cookbooks` — Adresář s definicemi kuchařek
    - `rulesets` — Adresář s definicemi souborů pravidel
    - `translation_maps` — Adresář s definicemi map
    - `formats` — Adresář s definicemi podporovaných formátů SBOM
  - `tests` — Adresář testů
    - `conftest.py` — Modul s prerekvizitami testování
    - `performance` — Balíček pro výkonnostní testování
    - `unit` — Balíček pro jednotkové testy
    - `testdata` — Adresář s testovacími daty
  - `.github` — Adresář s nastavením CI/CD pro platformu GitHub