特別研究報告

題目

定期的な脆弱性診断とチェックサムの検証を行う SPDX ドキュメント管理ツールの作成

指導教員
肥後 芳樹 教授

報告者
岸本 理央

令和 5 年 2 月 7 日

大阪大学 基礎工学部 情報科学科

令和 4 年度 特別研究報告

定期的な脆弱性診断とチェックサムの検証を行う SPDX ドキュメント管理ツールの作成

岸本 理央

内容梗概

近年のソフトウェア開発では，ソフトウェアの一部の機能を実現するためにライブラリが広く利用されている．ライブラリの利用には開発効率の向上などのメリットがある一方，ライブラリに含まれる不具合が取り込まれるというデメリットがある．使用しているライブラリに脆弱性と呼ばれるサイバーセキュリティ上の欠陥が発見された場合は迅速な対応が必要であるが，使用しているソフトウェアの管理が十分に行われておらず，適切な対応が行われないことが問題となっている．このような問題を解決するために，ソフトウェア部品表（SBOM）の利用が推奨されている．SBOM は，ソフトウェアに含まれるソフトウェアコンポーネント，それらのライセンス，およびソフトウェアコンポーネント間の依存関係を記述したドキュメントである．SPDX は SBOM の記述形式の 1 つであり，その形式に従って記述された SBOM は SPDX ドキュメントと呼ばれる．SPDX ドキュメントの情報はソフトウェアの適切な管理に役立てることができるが，その情報を用いたソフトウェアの管理を支援するツールは不足している．特に，ソフトウェアの脆弱性診断およびチェックサムの検証によるソフトウェアの破損・改ざん検知は定期的に実行することが重要である．それらの処理を既存のツールを組み合わせて実行することも可能であるが，手動作業を必要とする部分も多く残り，手間がかかるものである．

本研究では，SPDX ドキュメントを用いたソフトウェアの適切な管理を，既存のツールの組み合わせによって行う場合に比べてより省力化することを目的に，SPDX ドキュメントの管理を支援するツールを作成した．作成したツールによって，ソフトウェアの脆弱性診断およびチェックサムの検証によるソフトウェアの破損・改ざん検知を自動的かつ定期的に行うことが出来る．

作成したツールの有用性を確認するために，既存のツールの組み合わせで定期的な脆弱性診断とチェックサムの検証を行う場合と作成したツールを用いてそれらを行う場合を比較し，定期実行にかかる労力が削減できることを確認した．また，脆弱性診断処理とチェックサムの検証処理が定期実行にあたって十分な速度で動作することを確認した．

主な用語

SBOM
SPDX
脆弱性

目 次

1　まえがき

近年のソフトウェア開発では，ソフトウェアの一部の機能を実現するためにライブラリが利用されることがある．ライブラリの利用には開発期間の短縮や開発費用の削減が可能になるというメリットがある一方，ライブラリに含まれる不具合も取り込まれるというデメリットがある．ソフトウェアに含まれる不具合の中でサイバーセキュリティ上の欠陥は脆弱性と呼ばれ，発見された場合は脆弱性が含まれないバージョンにバージョンアップするなどの対応を迅速に行うことが必要である．しかし，使用しているソフトウェアの管理は十分に行われておらず，脆弱性への対応が遅れたり，行われなかったりすることが問題となっている．2021 年に深刻な脆弱性が発見された Java のログライブラリである Log4j では，2023 年 1 月時点においても日本から行われたダウンロードの 20%以上が脆弱性を含むバージョンであることが確認されている [22].

このようなソフトウェアの管理に関する問題を解決するために，ソフトウェア部品表（SBOM）の利用が推奨されている [20]．SBOM は，特定のソフトウェアに含まれるすべてのソフトウェアコンポーネント，それらのライセンス，およびソフトウェアコンポーネント間の依存関係が記述されているドキュメントである．SPDX は SBOM の記述形式の 1 つであり，その形式に従って記述された SBOM は SPDX ドキュメントと呼ばれる．SPDX ドキュメントに記載された情報に基づいて，使用しているソフトウェアやその依存ライブラリに脆弱性が含まれていないかを確認したり，ソフトウェアの改ざんや破損が発生していないかを確認したりすることが出来る．このように，SPDX ドキュメントはソフトウェアの適切な管理に役立つものであるが，SPDX ドキュメントを用いたソフトウェアの管理を支援するツールは不足している [30]．既存のツールを組み合わせて SPDX ドキュメントを用いたソフトウェアの管理を行うことは可能であるが，手動作業を必要とする部分も多く残り，手間がかかるものである．

本研究では，SPDX ドキュメントを用いたソフトウェアの適切な管理を，既存のツールの組み合わせによって行う場合に比べてより省力化して行うことを可能にすることを目的に，SPDX ドキュメントの管理を支援するツールを作成した．作成したツールに対して，SPDX ドキュメントを管理対象として追加することで，脆弱性診断とチェックサムの検証によるソフトウェアの破損・改ざん検知を自動的かつ定期的に行うことが出来る．これらの処理を定期的に実行することは既存のツールを組み合わせることでも可能であったが，手動作業を必要とする部分も多く残り，手間がかかるものであった．

作成したツールの有用性を確認するために，既存のツールの組み合わせで定期的な脆弱性診断とチェックサムの検証を行う場合と作成したツールを用いてそれらを行う場合を比較し，定期実行にかかる労力が削減できることを確認した．また，脆弱性診断処理とチェックサム

の検証処理が定期実行にあたって十分な速度で動作することを確認した.

以降，2 章では SBOM の利用推奨に至る背景と SPDX に関する説明を行い，先行研究と課題について論じる．3 章では脆弱性診断およびチェックサムの検証の手法について記す．4 章では作成したツールの具体的な実装について記す．5 章では作成したツールの評価を行う．最後に，6 章でまとめと今後の課題を記している．

2　背景

この章では本研究の背景として，ソフトウェアが利用するライブラリに起因する脆弱性に関する問題と，その問題への対応策として利用が勧められているソフトウェア部品表（SBOM）およびSBOMの形式の1つであるSPDXを説明する．それらを受けて，現在の課題と本研究の目的について述べる．

2.1　ライブラリの利用

ソフトウェアの開発では，ソフトウェアの一部の機能を実現するために既存のライブラリを利用することがある．ライブラリの利用によって，機能の実装に要する時間を削減できるため，開発期間の短縮や開発費用の削減が可能になる．また，広く利用されているライブラリは様々な環境で検証されているため，同様の機能を独自に実装する場合に比べて，信頼性が高く安定しているとされている [18].

ライブラリとして提供されるソフトウェアは，オープンソースソフトウェア（以降単にOSSという）として公開されることがある．ソフトウェア企業のSynopsysが2022年に発表したOSSに関する調査報告によると，調査した商用ソフトウェアのコードベースの97%にOSSが使用されていた [25].

2.2　脆弱性

ソフトウェアの不具合や設計上のミスが原因となって発生したサイバーセキュリティ上の欠陥を脆弱性と呼ぶ [31]．脆弱性には，遠隔地から送信した任意のコードを実行できる「リモートコード実行」や，コンピューターやネットワーク，システムを利用する権限を持たないユーザーが，一時的に利用権限を取得する「特権の昇格」などがある．

使用するソフトウェアに脆弱性が発見された場合，脆弱性が修正されたバージョンへのバージョンアップなどの対応を迅速に行う必要があるが，深刻な脆弱性であっても十分な対応が行われないことが問題となっている．2021年12月には，JavaのOSSライブラリであるLog4jにおいてリモートコード実行の脆弱性が発見された．この脆弱性は，脆弱性が存在するバージョンのLog4jが動作しているサーバー等に対して，脆弱性を悪用する細工されたデータを送信することで，攻撃者が任意のコードを実行できる可能性があるというものである．Log4jは汎用のログライブラリであり，Javaを用いて開発された様々なソフトウェアで利用されていたことから影響を受けるソフトウェアの数は多く，Maven Centralレポジトリに登録されているパッケージの約4%を占める，1万7000以上のパッケージが影響を受けると報告されている [14]．この脆弱性を悪用した攻撃は日本国内でも観測されている．脆弱性

の発見から 1 年以上が経過した 2023 年 1 月時点においても，日本における Log4j のダウンロードの 20%以上が脆弱性を含むバージョンであることが確認されている [22].

情報が公開された脆弱性は攻撃の対象となるため，脆弱性情報の公表後も脆弱性を含むバージョンのソフトウェアが利用され続けることは大きな問題である．また，近年は，攻撃者が既存の OSS に正当なコードを装って不正なコードを混入させたり，不正なコードを含むソフトウェアを既存の OSS と類似した名前で公開して誤用させたりすることで，ソフトウェアに意図的に混入させた脆弱性を利用して攻撃を行う，ソフトウェアサプライチェーン攻撃と呼ばれる攻撃が増加しており，脆弱性への迅速な対応がより重要になっている [21].

2.3 ソフトウェア部品表（SBOM）

ソフトウェア部品表（Software Bill of Materials，SBOM）は，特定のソフトウェアに含まれるすべてのソフトウェアコンポーネント，各ソフトウェアコンポーネントのライセンス，依存関係を一覧化したドキュメントである [32]．SBOM によって，そのソフトウェアの開発にどのソフトウェアコンポーネントが利用されているか把握でき，それらの情報を脆弱性への対応やソフトウェアのライセンスの順守に役立てることができる．アメリカ合衆国国家電気通信情報管理庁（NTIA）によって SBOM に最低限必要な要素が示されており，SBOM はデータフィールドとして以下の項目を含む必要があるとされている [19].

- ソフトウェアコンポーネントの提供者
- ソフトウェアコンポーネントの名前
- ソフトウェアコンポーネントのバージョン
- その他の一意な識別子
- ソフトウェアコンポーネント間の依存関係
- SBOM の著者
- SBOM の作成日時

2.4 SPDX

SPDX は SBOM を記述するための公開標準の一つであり，Linux Foundation のプロジェクトとして仕様が策定されている．現在も仕様の改訂が継続して行われており，最新バージョンは 2.3 である．バージョン 2.2.1 の仕様は，セキュリティ，ライセンス順守，その他

のソフトウェアサプライチェーン成果物の国際標準として ISO/IEC 5962:2021[15] で認定されている [26].

SPDX の仕様に従って作成された SBOM は SPDX ドキュメントと呼ばれる．図 1 に SPDX ドキュメントの構成を示す．SPDX ドキュメント作成情報には，ドキュメントの作成者と作成日時の情報が含まれている．パッケージ情報にはソフトウェア自身や依存ライブラリの名前，バージョン，ライセンスなどの情報が記述されている．また，図 1 に示す通り，SPDX ドキュメントはソフトウェアに関する様々な情報から構成されており，各情報は SPDX 要素と呼ばれ，他の要素から参照するための ID が振られている．SPDX 要素間の関係情報には，それらの ID を用いて SPDX 要素間の関係が記述されている．上記の項目は 2.3 節で述べた SBOM に最低限必要な要素に対応しており，SPDX はそれらの要素をすべて含んでいる．加えて，ファイル情報にはソフトウェアに関連するファイルのファイル名やチェックサムなどの情報が記述されており，それらの情報をファイルの同一性検証に利用することができる．

2.5 OSV データベース

OSV データベースは OSS の脆弱性を格納したデータベースである．様々な脆弱性データベースに登録された OSS の脆弱性に関する情報が集約されており，各脆弱性は Open Source Security Foundation[3] によって策定された Open Source Vulnerability 形式（OSV 形式）[4] で記述されている．OSV データベースがデータソースとして利用している脆弱性データベースには，GitHub Security Advisories，PyPA，RustSec，Global Security Database などが含まれる [5]．OSS の名前，バージョン，エコシステム等の情報を用いて，対象のソフトウェアに含まれる脆弱性の情報を取得可能な API が提供されている．

2.6 先行研究と課題

2.6.1 先行研究

SPDX ドキュメントを用いたソフトウェアの脆弱性診断ツールとして，spdx-to-osv[28] と OSV-Scanner[13] が存在する．

spdx-to-osv は，SPDX Workgroup によって開発されているコマンドラインツールである．入力として SPDX ドキュメントのパスと結果出力先ファイルのパスを指定すると，SPDX ドキュメントに記載されたパッケージに関連する脆弱性の情報を OSV データベースから取得し，OSV 形式で結果出力先ファイルに出力する．

OSV-Scanner は，Google によって開発されているコマンドラインツールである．入力として SPDX ドキュメントのパスを指定すると，ドキュメントに記載されたパッケージに関

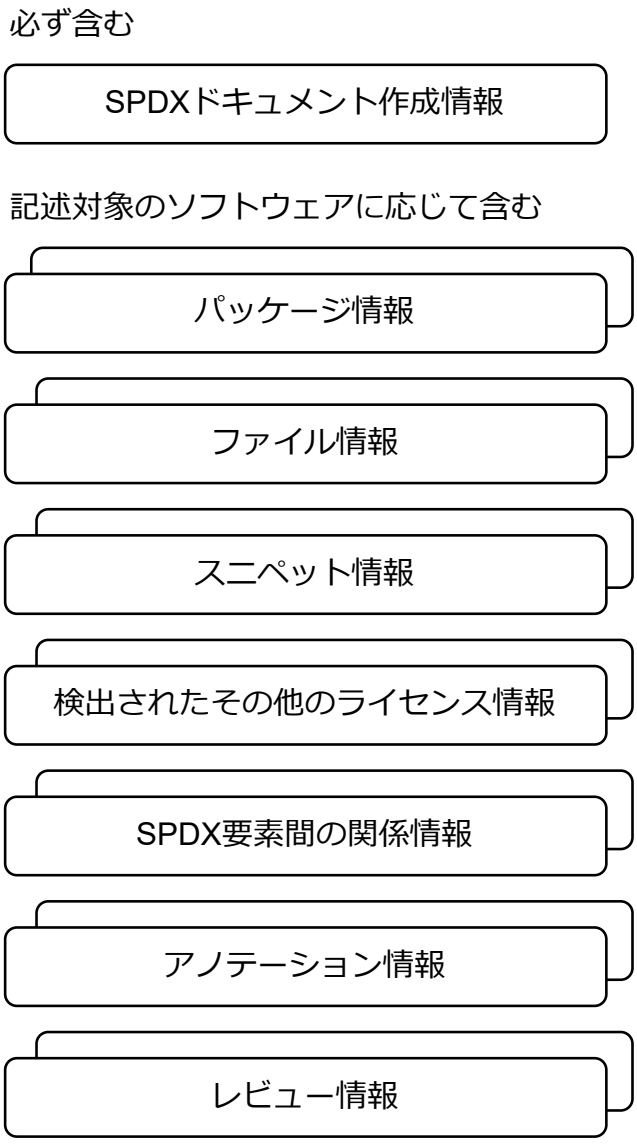

図 1: SPDX ドキュメントの構成（[27] を基に作成）

連する脆弱性の情報を OSV データベースから取得し，Markdown のテーブル形式，または OSV 形式で出力する．OSV-Scanner の入力として利用できる SPDX ドキュメントには，パッケージの情報に package URL[7] が記載されている必要がある．SPDX では package URL の記載が任意であるため，一部の SPDX ドキュメントでは正しく脆弱性診断が行えない可能性がある．

spdx-to-osv と OSV-Scanner のように，SPDX ドキュメントを用いたソフトウェアの脆弱性診断を支援する既存のツールは，指定した 1 つの SPDX ドキュメントに記述された情報に基づいて脆弱性診断を一度実行するものであり，複数の SPDX ドキュメントに対する脆弱性診断をまとめて行う機能や，定期的に脆弱性診断を行う機能は持たない．

2.6.2 課題と本研究の目的

ソフトウェアの脆弱性は日々発見されるため，ソフトウェアの脆弱性診断は定期的に行うことが望ましい．しかし，2.6.1 項で述べたように，SPDX ドキュメントに記載された情報に基づいて脆弱性診断を行う既存のツールには，脆弱性診断の定期的な実行を支援する機能が不足している．

また，脆弱性診断の正当性の保証には，SPDX ドキュメントの内容と実際のソフトウェアの情報が一致する必要があり，ソフトウェアを構成するファイルの改ざんや破損が発生していないことをファイルの同一性検証によって定期的に確認するのが望ましい．ファイルの同一性検証は，SPDX ドキュメントのファイル情報にチェックサムとして記載された正当なファイルのハッシュ値と，ローカルファイルから計算したハッシュ値の一致を確認することで行えるが，この作業を自動化するツールは存在しないため定期的に行うのは難しい．

本研究では SPDX を用いたソフトウェアの管理に関するこれらの問題を解決することを目的として，SPDX ドキュメントの管理手段を提供し，SPDX ドキュメントを用いたソフトウェアの脆弱性診断およびチェックサムの検証を定期的に自動実行するツールを作成する．

3　SPDX ドキュメントの管理手法

本章では，SPDX ドキュメントを対象とした脆弱性診断とチェックサムの検証の手法について説明を行う．

3.1　脆弱性診断

ソフトウェアの脆弱性に影響を与えるのは，ソフトウェアが実行時に依存するパッケージの脆弱性のみであり，実行時に依存関係にないパッケージの脆弱性はソフトウェアの脆弱性に影響しない．SPDX ドキュメントには，ソフトウェアのテスト処理のみが依存するパッケージのように，ドキュメントで記述しているソフトウェアと関係しているが，実行時には依存しないパッケージの情報も記述されている可能性がある．そのため，ドキュメントに記載されたパッケージ間の関係情報を利用して依存関係グラフを作成し，ソフトウェアが実行時に依存するパッケージを特定する．特定された依存パッケージについて，ドキュメントに記載された情報を利用して OSV データベースから関連する脆弱性情報を取得することで脆弱性診断を行う．

3.1.1　SPDX ドキュメントにおけるパッケージ情報と要素間の関係の記述

SPDX ドキュメントには，ソフトウェアに関係するパッケージの情報を記述する要素が存在する．コード 1 に JSON 形式の SPDX ドキュメントにおいてパッケージの情報が記述された要素を示した．この例では，名前が log4net（2 行目），バージョンが 2.0.8（12 行目）であるライブラリについて記述されており，この要素には ID として SPDXRef-Package-log4net（3 行目）が与えられている．そのほか，ダウンロード元やライセンス，著作権の情報は NOASSERTION となっており，記載されていない．

コード 1: パッケージの情報

```
1  {
2    "name": "log4net",
3    "SPDXID": "SPDXRef-Package-log4net",
4    "downloadLocation": "NOASSERTION",
5    "filesAnalyzed": false,
6    "licenseConcluded": "NOASSERTION",
7    "licenseInfoFromFiles": [
8      "NOASSERTION"
9    ],
10   "licenseDeclared": "NOASSERTION",
11   "copyrightText": "NOASSERTION",
12   "versionInfo": "2.0.8",
13   "externalRefs": [
```

```
14      {
15        "referenceCategory": "PACKAGE-MANAGER",
16        "referenceType": "purl",
17        "referenceLocator": "pkg:nuget/log4net%402.0.8"
18      }
19    ],
20    "supplier": "NOASSERTION"
21  }
```

また，SPDX ドキュメントには SPDX 要素間の関係を記述する要素が存在する．要素間の関係の情報は，関係の種類と，関係する 2 つの要素の ID を用いて記述される．JSON 形式の SPDX ドキュメントにおいて，要素間の関係を記述した部分をコード 2 に示した．この例では，図 2 に示した関係が記述されている．コード 2 の 1 行目から 5 行目に記載された関係は，この SPDX ドキュメント（SPDXRef-DOCUMENT）がソフトウェア（SPDXRef-RootPackage）を記述（DESCRIBES）することを示している．また，6 行目から 10 行目ではソフトウェア（SPDXRef-RootPackage）がライブラリ（SPDXRef-Package-log4net）に依存（DEPENDS_ON）することが記述されている．

コード 2: 要素間の関係の情報

```
1  {
2    "relationshipType": "DESCRIBES",
3    "relatedSpdxElement": "SPDXRef-RootPackage",
4    "spdxElementId": "SPDXRef-DOCUMENT"
5  },
6  {
7    "relationshipType": "DEPENDS_ON",
8    "relatedSpdxElement": "SPDXRef-Package-log4net",
9    "spdxElementId": "SPDXRef-RootPackage"
10 }
```

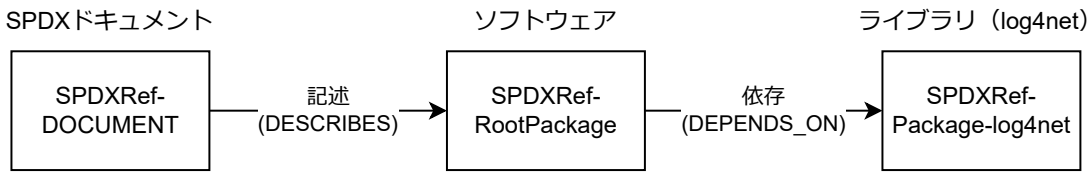

図 2: コード 2 で記述された関係

3.1.2 依存関係グラフ

依存関係グラフは，パッケージ間の依存関係を表す有向グラフであり，頂点はパッケージを表し，有向辺はパッケージ間の依存関係を表す．依存関係グラフの例を図 3 に示した．あるパッケージが別のパッケージに直接的または間接的に依存する場合，依存関係グラフにおいて 2 つのパッケージ間に経路が存在する．例えば，図 3 では，パッケージ A からパッケージ B に有向辺 AB が存在しており，パッケージ A がパッケージ B に依存することを示している．また，パッケージ A からパッケージ G への経路は存在しないため，パッケージ A はパッケージ G に依存しないことが分かる．

SPDX ドキュメントに記載されたパッケージ間の関係情報に基づいて作成した依存関係グラフにおいて，ドキュメントが記述しているソフトウェアのパッケージから経路が存在するパッケージを列挙することで，ソフトウェアが実行時に依存するパッケージを特定する．

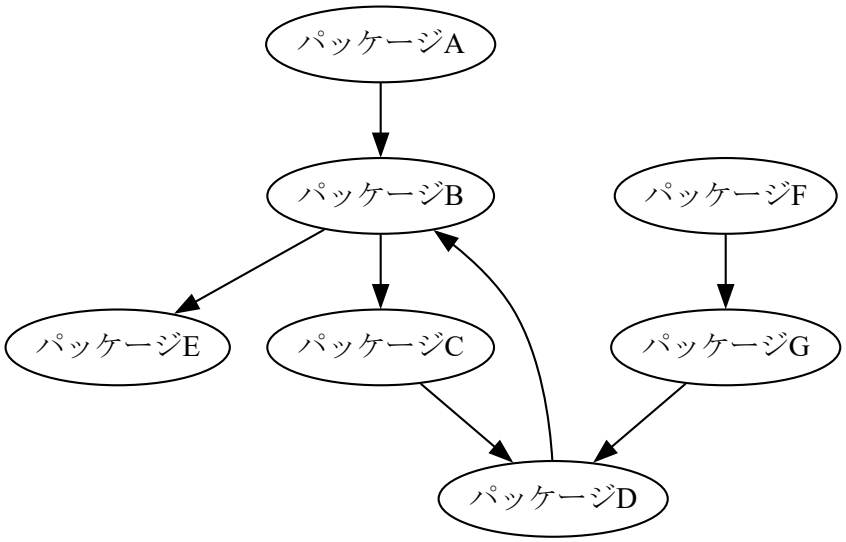

図 3: 依存関係グラフの例

3.2 チェックサムの検証

SPDX ドキュメントには，ソフトウェアを構成するファイルを一覧して記述するセクションが存在しており，ファイルの情報としてファイル名とチェックサムが記載されている．JSON 形式の SPDX ドキュメントにおいてファイルの情報を記述した部分をコード 3 に示した．この例はファイル名が log4net.dll であるファイルの情報であり，SHA-256 と SHA-1 で計算されたチェックサムが記述されている．SPDX ではファイルのチェックサムとして SHA-1 ハッシュ値の記載が必須となっているため，チェックサムの検証ではローカルファイルの SHA-1

ハッシュ値を計算し，その値がSPDXドキュメントに記載されたチェックサムの値と一致することを確認する．値が一致しない場合はローカルファイルの改ざんや破損が発生していると考えられる．

コード 3: ファイルの情報

```
{
  "fileName": "./log4net.dll",
  "SPDXID": "SPDXRef-File--log4net.dll-40
      FDBA136F864C8A2F3E3F9C9E3949F7582A6077",
  "checksums": [
    {
      "algorithm": "SHA256",
      "checksumValue": "
          dde0278246298776f9d7fa731cef6463a4dbd5a53b7034e88aca30852e272b73"
    },
    {
      "algorithm": "SHA1",
      "checksumValue": "40fdba136f864c8a2f3e3f9c9e3949f7582a6077"
    }
  ],
  "licenseConcluded": "NOASSERTION",
  "licenseInfoInFiles": [
    "NOASSERTION"
  ],
  "copyrightText": "NOASSERTION"
}
```

4　ツールの実装

4.1　開発言語と想定環境

ツールの作成は，C#[16] を用いて行った．また，UI フレームワークとして，Windows 上で動作する WPF（Windows Presentation Foundation）[12] を用いた．使用した.NET のバージョンは.NET 6.0 であり，ツールの実行には.NET 6.0 がインストールされた Windows 環境が必要である．また，SPDX ドキュメントの形式の変換に Java で作成された tools-java[29] を使用しているため，Java 11 以上が実行できる環境が必要である．ツールの動作確認は，Windows10 上で行った．

4.2　対応する SPDX のバージョン

本ツールは SPDX のバージョン 2.2.2[10] の仕様に従って作成された SPDX ドキュメントを対象としている．

4.3　ツールの構成

図 4 にツールの構成を示す．ツールは管理対象のソフトウェアのリストを保持している．ソフトウェアのリストの各項目は，ソフトウェアの表示名，SPDX ドキュメントの内容，およびソフトウェアが配置されているディレクトリのパスを含んでいる．このリストの情報に基づいて定期的に脆弱性診断とチェックサムの検証を行う．ツールが用いるデータの管理は SQLite[23] を用いて行っている．ソフトウェアのリスト，チェックサムの検証の結果，脆弱性診断の結果は，ツール内部の SQLite データベースに保存される．定期的な脆弱性診断とチェックサムの検証において問題が見つかった場合は，Windows のアプリ通知機能を用いて利用者に通知を行う．

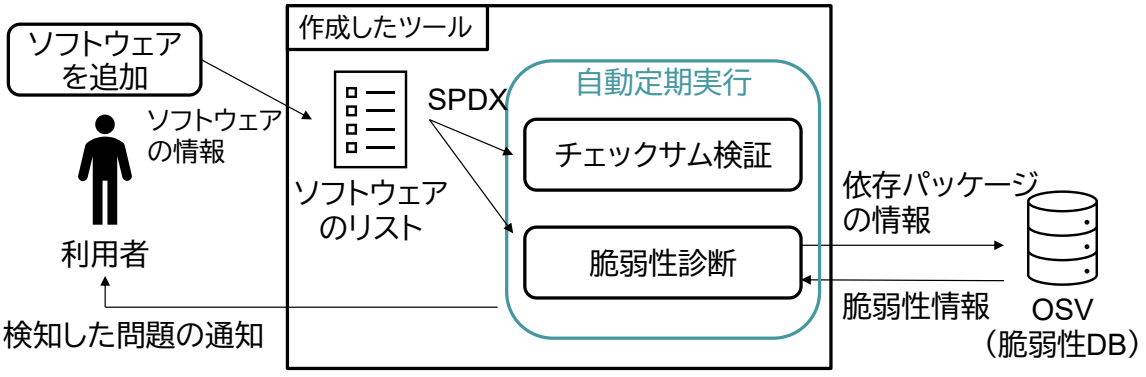

図 4: ツールの構成

4.4　管理するソフトウェアの追加

管理するソフトウェアの追加は，図 5 に示す管理対象への追加画面で行う．この画面で，SPDX ドキュメントファイルのパス，ソフトウェアが配置されているディレクトリのパス，ツールでの表示名を指定し，追加ボタンを押すことで管理対象に追加される．SPDX ドキュメントファイルのパスとツールでの表示名は必須入力項目であり，ソフトウェアが配置されているディレクトリのパスは任意入力項目である．指定されたパスに存在する SPDX ドキュメントの内容，ツールでの表示名，およびソフトウェアが配置されているディレクトリのパスはツール内部の SQLite データベースに保存される．

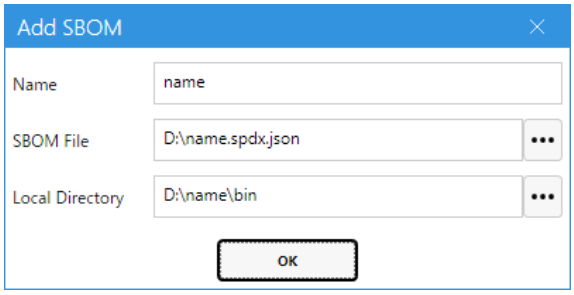

図 5: 管理するソフトウェアの追加画面

4.4.1　tools-java を用いたファイル形式の変換

SPDX ドキュメントは，tag-value 形式，JSON 形式，XML 形式など様々なファイル形式を用いて記述することができる．形式の変換は情報の損失なしに行えるため，SPDX ドキュメントの内容はツールで扱いやすい JSON 形式に変換してツール内部のデータベースに保存する．SPDX ドキュメントのファイル形式の変換は，SPDX Workgroup によって公式に提供されている tools-java[29] を用いて行う．このツールは Java で作成されているため，.NET の標準ライブラリで提供される System.Diagnostics.Process クラスの Run メソッドを用いて java コマンドを実行することで使用する．tools-java の引数として，ファイル形式の変換を表す Convert を指定し，I オプションで変換元の SPDX ドキュメントファイルのパス，O オプションで変換先の SPDX ドキュメントファイルのパスを指定する．変換先のファイルパスには一時ファイルを指定する．変換後のファイルデータをツール内部のデータベースに保存した後に一時ファイルは削除する．

4.5　管理するソフトウェアの一覧表示

管理対象のソフトウェアの表示名，最新の脆弱性診断結果，およびチェックサムの検証結果を一覧表示する．管理対象のソフトウェアを一覧表示する画面を図 6 に示す．リストから

選択されたソフトウェアについて，脆弱性診断結果の詳細と，チェックサムの検証結果の詳細を表示する．脆弱性診断結果の表示では，脆弱性を含むパッケージを強調表示し（図 7），クリックによって脆弱性を含むパッケージが選択されれば脆弱性の詳細を表示する（図 8）．

図 7 の例では，5 つのソフトウェアが管理されており，ILSpy，jellyfin，ScreenToGif の 3 つのソフトウェアに脆弱性が見つかったことが表示されている．また，ILSpy がソフトウェアのリストで選択されており，画面右側のパッケージごとの脆弱性診断結果のリストにおいて，Newtonsoft.Json のバージョン 8.0.3 が脆弱性を含むパッケージとして強調表示されている．図 8 は，このパッケージを選択して脆弱性の詳細情報を表示した場合の画面画像である．

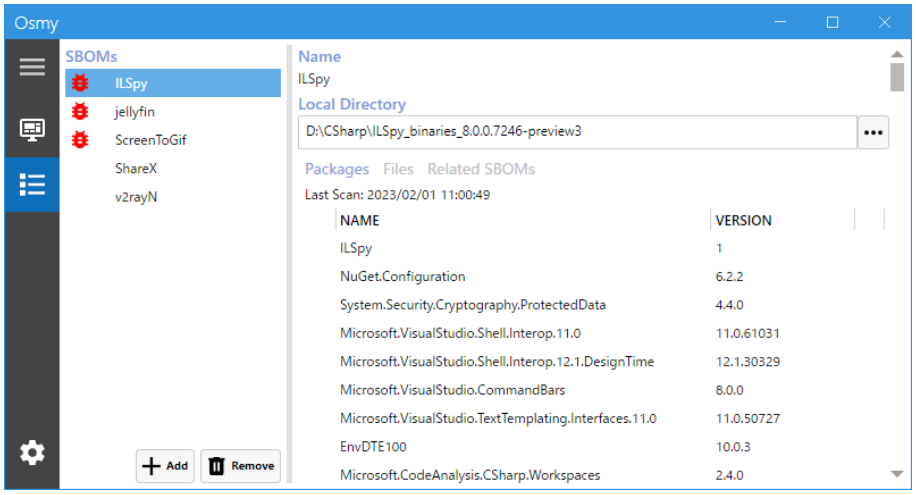

図 6: 管理対象のソフトウェアの一覧表示

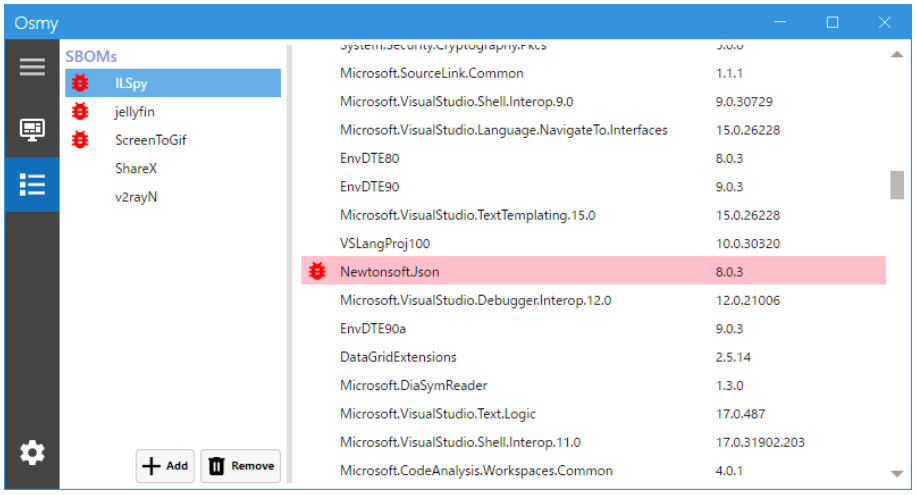

図 7: 脆弱性を含むパッケージの強調表示

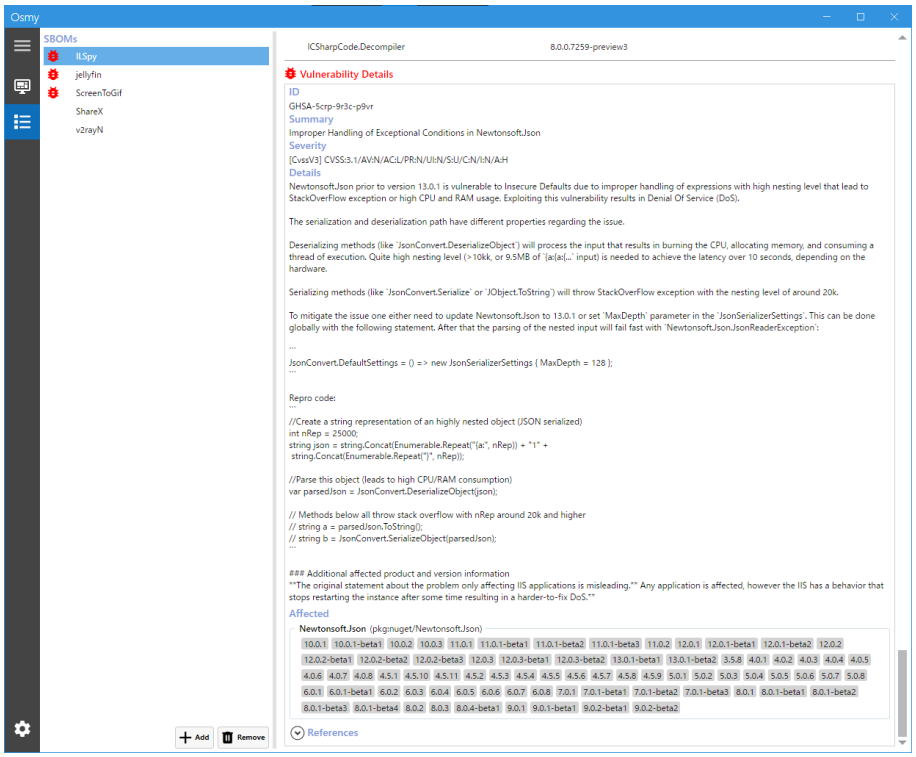

図 8: 脆弱性の詳細情報

4.6 管理するソフトウェアの削除

管理対象のソフトウェアの一覧から，削除したいソフトウェアを選択して削除ボタンを押すことで，ソフトウェアが管理対象から削除される．削除時には，JSON 形式に変換して保持されていた SPDX ドキュメントのデータと，そのドキュメントの情報に基づいて行われた脆弱性診断とチェックサムの検証の結果がツール内部のデータベースから削除される．

4.7 脆弱性診断

SPDX ドキュメントの情報から，SPDX ドキュメント内に情報が記載されたパッケージの依存関係グラフを作成し，SPDX ドキュメントが記述しているソフトウェアが実行時に依存するパッケージを特定する．特定されたパッケージについて，OSV データベースから脆弱性情報を取得する．

4.7.1 依存関係グラフの作成

始めに，SPDX ドキュメントに情報が記述されているすべてのパッケージをグラフの頂点として追加する．次に，SPDX ドキュメントに記述された SPDX 要素間の関係情報の内，関

係が定義される 2 つの SPDX 要素がともにパッケージ情報であり，SPDX ドキュメントが記述するソフトウェアの実行時の脆弱性に影響するものを選択し，それらに基づいてグラフに辺を追加する．

SPDX 要素間に定義される関係の内，ソフトウェアの実行時の脆弱性に影響する関係を表 1，表 2，表 3 に示す．表 1 に記載された関係は順方向の依存関係である．この関係は，パッケージ A がパッケージ B に依存することを示し，B が脆弱性を含む場合に A が脆弱性を含む可能性がある．したがって，依存関係グラフにパッケージ A からパッケージ B への有向辺 AB を追加する．表 2 に記載された関係は逆方向の依存関係である．この関係は，パッケージ B がパッケージ A に依存することを示し，A が脆弱性を含む場合に B が脆弱性を含む可能性がある．したがって，依存関係グラフにパッケージ B からパッケージ A への有向辺 BA を追加する．表 3 に記載された関係は，一方が脆弱性を含む場合に他方も脆弱性を含む可能性がある関係である．これらは，双方向の依存関係であると考えて，依存関係グラフに双方向の辺を追加する．

表 1: B が脆弱性を含む場合に A が脆弱性を含む可能性がある関係

関係	説明
CONTAINS	A が B を含む
DYNAMIC_LINK	A が B を動的リンクする
EXPANDED_FROM_ARCHIVE	A はアーカイブ B から展開された
FILE_ADDED	A はファイル B を追加した
GENERATED_FROM	A は B から作成された
PATCH_FOR	A は B のパッチである
STATIC_LINK	A は B を静的リンクする
HAS_PREREQUISITE	A は B を前提条件として持つ
DEPENDS_ON	A は B に依存する

4.7.2 依存パッケージの特定

4.7.1 項で述べた方法で作成した依存関係グラフにおいて，SPDX ドキュメントが記述しているソフトウェア自身のパッケージ（以降単にルートパッケージという）から深さ優先探索を行う．探索の過程で訪問されたパッケージはルートパッケージから経路が存在するため，

表 2: A が脆弱性を含む場合に B が脆弱性を含む可能性がある関係

関係	説明
CONTAINED_BY	A は B に含まれる
DISTRIBUTION_ARTIFACT	A の配布には B の配布も必要である
GENERATES	A は B を生成する
OPTIONAL_COMPONENT_OF	A は B のオプションコンポーネントである
PATCH_APPLIED	A は B に適用されたパッチファイルである
PREREQUISITE_FOR	A は B の前提条件である
DEPENDENCY_OF	A は B の依存要素である
OPTIONAL_DEPENDENCY_OF	A は B のオプションの依存要素である
RUNTIME_DEPENDENCY_OF	A は B の実行時依存要素である
COPY_OF	A は B の複製である
PACKAGE_OF	A は B のパッケージである
VARIANT_OF	A は B の派生である

表 3: 一方が脆弱性を含む場合に他方も脆弱性を含む可能性がある関係

関係	説明
COPY_OF	A は B の複製である
PACKAGE_OF	A は B のパッケージである
VARIANT_OF	A は B の派生である

SPDX ドキュメントが記述しているソフトウェアの実行時の依存パッケージである．訪問されたパッケージを記録し，依存パッケージのリストを作成する．

4.7.3　OSV データベースから脆弱性情報を取得

4.7.2 項で述べた方法で特定された依存パッケージについて，パッケージの名前とバージョンを用いて OSV データベースに問い合わせを行い，パッケージに含まれる脆弱性の情報を取得する．OSV データベースへの問い合わせは，公式に提供されている OSV API[6] との HTTP 通信によって行う．OSV API では，脆弱性情報の取得クエリを最大 1000 件までまとめて行うバッチ実行 API が提供されているため，これを用いて脆弱性情報の取得処理を効率化する．また，様々なソフトウェアで広く利用されているライブラリのように，同一パッケージの情報が複数の SPDX ドキュメントに含まれることがあるため，名前とバージョンが一致するパッケージについてのクエリは 1 つにまとめて行い，クエリの数を削減する．取得した脆弱性の情報は，ツール内部のデータベースに保存する．

4.8　チェックサムの検証

チェックサムの検証は，検証対象ファイル情報の取得，ローカルファイルのハッシュ値の計算，チェックサムの一致確認の 3 つの処理からなる．

4.8.1　検証対象ファイル情報の取得

SPDX ドキュメントのファイル情報から検証対象ファイルのファイル名とチェックサム（SHA-1 ハッシュ値）を取得する．ファイル情報には，ファイルのチェックサムとして，異なるアルゴリズムで計算した複数の値を記載することが出来るが，SHA-1 ハッシュ値のみ記載が必須とされているため，本ツールのチェックサムの検証処理では SHA-1 ハッシュ値のみを用いる．

4.8.2　ローカルファイルのハッシュ値の計算

4.8.1 項で述べた方法で取得したファイル名が，基準ディレクトリからの相対パスとして記載されていることを想定してローカルファイルの絶対パスを作成する．相対パスの基準ディレクトリしては，管理するソフトウェアの追加時に指定された，ソフトウェアが配置されているディレクトリのパスを用いる．SHA-1 ハッシュ値の計算は，.NET の標準ライブラリで提供される System.Security.Cryptography.SHA1 クラスを用いて行う．

4.8.3　チェックサムの一致確認・結果の保存

4.8.2 項で述べた方法で計算したハッシュ値と，4.8.1 項で述べた方法で SPDX ドキュメントから取得したチェックサムの値が一致することを文字列の比較によって確認する．確認の結果はツール内部のデータベースに保存する．

4.9　問題発見時の利用者への通知

脆弱性診断とチェックサムの検証において問題が発見された場合，利用者に通知を行う．通知は Windows のアプリ通知機能を用いて行い，脆弱性が検出されたソフトウェアの数と，チェックサムの不一致が検出されたソフトウェアの数をそれぞれ通知する（図 9，図 10）．

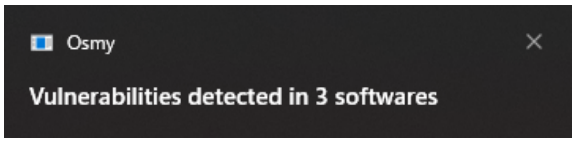

図 9: 脆弱性検出時の通知

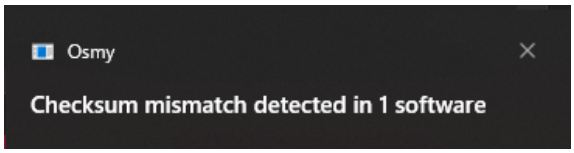

図 10: チェックサムの不一致検出時の通知

5　評価

作成したツールが実現する脆弱性診断の定期実行手法の評価と，作成したツールの性能の評価を行い，ツールの有用性を確認する．

5.1　定期的な脆弱性診断手法の評価

定期的な脆弱性診断を既存のツールである spdx-to-osv を用いて行う場合と，作成したツールを用いて行う場合を比較する．

5.1.1　既存ツール（spdx-to-osv）を用いる場合

spdx-to-osv は脆弱性診断対象の SPDX ドキュメントファイルのパスを引数として与えることで OSV データベースへの問い合わせが行われ，検出された脆弱性情報がファイルに出力される．脆弱性診断を行いたい SPDX ドキュメントの数だけ，引数を変えながら spdx-to-osv を実行することで脆弱性診断対象のすべての SPDX ドキュメントについて脆弱性診断を行うことが出来る．spdx-to-osv の終了コードは脆弱性の検出有無では変化しないため，出力されたファイルの内容から脆弱性が検出されているかを確認する必要がある．定期的な脆弱性診断は，これらの作業を繰り返し行うことで実現できる．

5.1.2　作成したツールを用いる場合

SPDX ドキュメントを作成したツールに管理対象として追加することで，あらかじめ設定した間隔で脆弱性診断が定期実行される．脆弱性が検出された場合は通知が行われるため，脆弱性の検出有無の確認を定期実行毎に行う必要はない．

5.1.3　2 つの場合の比較

既存ツールを用いる場合に必要であった結果の確認作業が作成したツールの通知機能によって不要になり，脆弱性診断の実行毎に手動で作業を行う必要が無くなる．作成したツールでは SPDX ドキュメントを管理対象に追加する作業が必要になるが，この作業は SPDX ドキュメント毎に一度だけ行うものであるため，大きな負担の増加にはならない．実行毎に毎回必要であった作業が不要になることから定期的な脆弱性診断の実行は省力化されていると言え，結果の確認作業を忘れることで脆弱性への対応が遅れる可能性も低下することから，作成したツールは有用であると考えられる．

5.2 定期的なチェックサムの検証手法の評価

チェックサムの検証をチェックサムを計算するコマンドを用いて行う場合と，作成したツールを用いて行う場合を比較する．

5.2.1 チェックサムを計算するコマンドを用いる場合

Windows ではファイルのハッシュ値を計算するコマンドとして，Get-FileHash コマンドが利用できる．このコマンドは引数でファイルパスとハッシュアルゴリズムを指定すると，指定したハッシュアルゴリズムで計算されたファイルのハッシュ値が出力されるものである．SPDX ドキュメントに記載された各ファイルについて，このコマンドを実行してハッシュ値を計算し，SPDX ドキュメントにチェックサムとして記述されているハッシュ値と照合することでチェックサムの検証が行える．定期的なチェックサムの検証は，これらの作業を繰り返し行うことで実現できる．

5.2.2 作成したツールを用いる場合

作成したツールを用いる場合は，SPDX ドキュメントを管理対象に追加し，ソフトウェアが配置されているディレクトリを指定する．管理対象として追加された SPDX ドキュメントについて，ソフトウェアが配置されているディレクトリが指定されていれば自動的にチェックサムの検証が定期実行されるため，実行毎の利用者による操作は不要である．また，自動実行でチェックサムの不一致が検出された場合は通知が行われるため，チェックサムの検証結果を定期実行毎に確認する必要は無い．

5.2.3 2 つの場合の比較

チェックサムを計算するコマンドを用いてチェックサムの検証を行う場合，SPDX ドキュメントに情報が記載された各ファイルについてコマンドを実行してハッシュ値を計算し，ドキュメントに記載されたチェックサムと照合する作業が必要であり，大きな手間がかかる．作成したツールでは，SPDX ドキュメントを管理対象に追加し，ソフトウェアが配置されているディレクトリを指定することで，チェックサムの検証は自動的に行われる．このように，チェックサムの検証に必要であった手動作業が大幅に削減されることから，作成したツールは有用であると考えられる．

5.3 ツールの性能評価

5.3.1 評価方法

作成したツールによって行われる脆弱性診断およびチェックサムの検証処理の実行時間を計測し，定期的な実行が可能な速度で処理が行えることを確認する．脆弱性診断およびチェックサムの検証の対象となる SPDX ドキュメントは，表 4 に示した 5 個の OSS から sbom-tool[17] を用いて作成した．脆弱性診断とチェックサムの検証それぞれについて，実行時間を 5 回計測し平均を取った．実行時間の計測は表 5 に示した環境で行った．

表 4 中のパッケージとは，SPDX ドキュメントの作成時に sbom-tool によって，ソフトウェアが実行時に依存すると判断されたパッケージを指す．それらには実際には実行時に依存しないパッケージが含まれている場合や，ソフトウェアのコンパイル時に複数のパッケージのファイルが 1 つにまとめられる場合があるため，パッケージ数よりファイル数が少ないことがある．対象の OSS に含まれるパッケージの内，互いに名前またはバージョンが異なるユニークなパッケージの数は 525 個であった．

表 4: SPDX ドキュメントの作成に用いた OSS

	パッケージの数	ファイル数
v2rayN[11]	36	60
ShareX[9]	11	557
jellyfin[2]	162	177
ScreenToGif[8]	75	1
ILSpy[1]	267	38
合計（ユニーク）	551（525）	833（833）

表 5: 計測環境

OS	Windows 10
CPU	Intel Core i7-11700K
RAM	32GB

5.3.2 計測結果

処理時間の計測結果を表 6 に示した．脆弱性診断の実行時間の平均値は約 8.6 秒であり，チェックサムの検証の実行時間の平均値は約 5.3 秒であった．

表 6: 作成したツールの処理時間

	脆弱性診断（秒）	チェックサムの検証（秒）
1	9.9886185	5.7431088
2	7.6718912	5.1176218
3	8.0809527	5.4572970
4	8.6877487	5.0429236
5	8.5597892	5.1126877
平均	8.59780006	5.29472778

5.3.3 考察

作成したツールは名前とバージョンが一致するパッケージについての問い合わせはまとめて行うため，この値はユニークな 525 個のパッケージに対して行った処理時間である．ソフトウェア企業の Synopsys による調査 [24] によると，商用ソフトウェアに含まれる OSS パッケージの数は平均 528 個である．脆弱性診断に要する時間は OSV データベースに問い合わせを行う回数におおむね比例し，今回の計測では，525 個のパッケージの脆弱性診断に要した時間は約 8.6 秒であったため，528 個のパッケージの脆弱性診断は約 9 秒で行える．したがって，1 時間に約 400 個のソフトウェアについて脆弱性診断が行えると考えられる．

ソフトウェアに含まれるファイルの数はソフトウェアによって大きく異なる．そのため，一般的なソフトウェアに含まれるファイルの数を推測することは難しいが，計測を行ったコンピューターにインストールされていたソフトウェアから計算した，1 ソフトウェアあたりのファイル数は 7000 個であった．今回の計測では，833 個のファイルのチェックサムの検証に要した時間は約 5.3 秒であったため，7000 個のファイルに対するチェックサムの検証は約 45 秒で行える．したがって，1 時間に約 80 個のソフトウェアについてチェックサムの検証が行えると考えられる．

肥後研究室で利用されているサーバーの 1 つにインストールされているソフトウェアの数は，629 個であった．それらのソフトウェアについて，作成したツールを用いて脆弱性診断とチェックサムの検証を行う場合の処理時間を，上記の結果に基づいて計算すると，脆弱性

診断の処理時間は約 1.6 時間，チェックサムの検証の処理時間は約 7.9 時間である．したがって，作成したツールは実際に運用されているサーバーについて，1 日に 1 回以上脆弱性診断とチェックサムの検証が行える速度で処理が可能であり，十分な性能を持つと考えられる．

6　まとめ

本研究では，SPDX ドキュメントを用いたソフトウェアの脆弱性診断およびチェックサムの検証を定期的に自動実行するツールを作成した．作成したツールによって，既存ツールの組み合わせで同様の処理を行う場合に必要であった手動作業が自動化され，SPDX ドキュメントの適切な管理がより容易に行えるようになった．

作成したツールは，使用した UI フレームワークが Windows 上でしか動作しないことにより動作環境に強い制約が生じているため，UI フレームワークの変更等により様々なプラットフォーム上で利用できるようにしたい．また，脆弱性診断で利用している OSV データベースには OSS の脆弱性のみが登録されており，OSS 以外の脆弱性を検出することが出来ないため，脆弱性情報を取得するデータベースを追加することにより，より多くのソフトウェアの脆弱性を検出できるようにする必要があると考えられる．

今回行ったツールの評価では，管理対象として用いる SPDX ドキュメントを大量に用意することが難しかったため，ツールを用いて管理した SPDX ドキュメントの数は少なかった．大量の SPDX ドキュメントを管理した場合の性能と使いやすさの評価を追加で行い，問題が見つかった場合はツールの改良を行いたい．

謝辞

大阪大学大学院情報科学研究科コンピュータサイエンス専攻 肥後 芳樹 教授には，ご多忙の中研究活動に対して多くの貴重なご助言やご指導を賜りました．心より深く感謝申し上げます．大阪大学大学院情報科学研究科コンピュータサイエンス専攻 松下 誠 准教授には，研究室の発表機会において，ご意見，ご助言を賜りました．心より深く感謝申し上げます．

南山大学大学院理工学研究科ソフトウェア工学専攻 井上 克郎 教授，福知山公立大学情報学部 眞鍋雄貴 講師，ならびに大阪大学大学院情報科学研究科コンピュータサイエンス専攻 神田 哲也 助教には研究活動の直接のご指導，論文の執筆に至るまで，あらゆる場面で多くのご指導を賜りました．心より深く感謝申し上げます．

大阪大学大学院情報科学研究科コンピュータサイエンス専攻 田邉 傑士 氏には，研究室での生活や，研究活動において，日々様々なご支援，ご助言を賜りました．心より深く感謝申し上げます．

最後に，その他様々なご指導，ご助言等を賜りました，大阪大学大学院情報科学研究科コンピュータサイエンス専攻肥後研究室の皆様に，心より深く感謝申し上げます．

参考文献

[1] Ilspy. https://github.com/icsharpcode/ILSpy.

[2] jellyfin. https://github.com/jellyfin/jellyfin.

[3] Open Source Security Foundation. https://openssf.org/.

[4] Open source vulnerability format. https://ossf.github.io/osv-schema/.

[5] OSV. https://osv.dev/.

[6] OSV API. https://osv.dev/docs/.

[7] package URL. https://github.com/package-url/purl-spec.

[8] Screentogif. https://github.com/NickeManarin/ScreenToGif.

[9] Sharex. https://github.com/ShareX/ShareX.

[10] Spdx v2.2.2. https://spdx.github.io/spdx-spec/v2.2.2/.

[11] v2rayn. https://github.com/2dust/v2rayN.

[12] .NET Foundation. WPF. https://github.com/dotnet/wpf.

[13] Google. OSV-Scanner. https://github.com/google/osv-scanner.

[14] Google. Understanding the Impact of Apache Log4j Vulnerability. https://security.googleblog.com/2021/12/understanding-impact-of-apache-log4j.html, 2021.

[15] ISO. ISO/IEC 5962:2021. https://www.iso.org/standard/81870.html.

[16] Microsoft. C#. https://learn.microsoft.com/ja-jp/dotnet/csharp/.

[17] Microsoft. sbom-tool. https://github.com/microsoft/sbom-tool.

[18] P. Mohagheghi, R. Conradi, O.M. Killi, and H. Schwarz. An empirical study of software reuse vs. defect-density and stability. In *Proceedings. 26th International Conference on Software Engineering*, pp. 282–291, 2004.

[19] NTIA. The Minimum Elements For a Software Bill of Materials (SBOM). https://www.ntia.gov/report/2021/minimum-elements-software-bill-materials-sbom, 2021.

[20] NTIA Multistakeholder Process on Software Component Transparency Framing Working Group. Framing Software Component Transparency: Establishing a Common Software Bill of Material (SBOM). https://www.ntia.gov/files/ntia/publications/framingsbom_20191112.pdf, 2019.

[21] sonatype. 2021 state of the software supply chain. https://www.sonatype.com/resources/state-of-the-software-supply-chain-2021.

[22] sonatype. Log4j Exploit Updates. https://www.sonatype.com/resources/log4j-vulnerability-resource-center.

[23] SQLite. SQLite Home Page. https://sqlite.org/index.html.

[24] Synopsys. 2021 年の OSSRA レポートから読み解く、商用ソフトウェアにおけるオープンソースの状況. https://www.synopsys.com/blogs/software-security/ja-jp/open-source-trends-ossra-report/, 2021.

[25] Synopsys. 2022 Open Source Security and Risk Analysis Report. https://www.synopsys.com/software-integrity/resources/analyst-reports/open-source-security-risk-analysis.html, 2022.

[26] SPDX Workgroup. About - Software Package Data Exchange (SPDX). https://spdx.dev/about.

[27] SPDX Workgroup. Composition of an SPDX document. https://spdx.github.io/spdx-spec/v2.2.2/img/spdx-2.2.2-document.png.

[28] SPDX Workgroup. spdx-to-osv. https://github.com/spdx/spdx-to-osv.

[29] SPDX Workgroup. tools-java. https://github.com/spdx/tools-java.

[30] Boming Xia, Tingting Bi, Zhenchang Xing, Qinghua Lu, and Liming Zhu. An empirical study on software bill of materials: Where we stand and the road ahead, 2023.

[31] 総務省. 国民のためのサイバーセキュリティサイト — 用語辞典（さ行）. https://www.soumu.go.jp/main_sosiki/cybersecurity/kokumin/glossary/glossary_03.html#%E8%84%86%E5%BC%B1%E6%80%A7.

[32] 日立ソリューションズ. SBOM：Software Bill Of Materials とは?（フォーマット、ユースケース、ツールについて）. https://www.hitachi-solutions.co.jp/oms/sp/blog/2021120105/, 2021.