社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

# SPDXドキュメントを用いた脆弱性診断とチェックサム検証を行うツール

岸本　理央† 　神田　哲也† 　眞鍋　雄貴†† 　井上　克郎††† 肥後　芳樹†

† 大阪大学大学院情報科学研究科 〒565–0871 大阪府吹田市山田丘 1–5
††† 南山大学理工学部ソフトウェア工学科 〒466–8673 愛知県名古屋市昭和区山里町 18
†† 福知山公立大学情報学部情報学科 〒620–0886 京都府福知山市堀 3370

**あらまし**　近年のソフトウェア開発ではライブラリが広く利用されている．しかし，その管理は十分ではなく，ライブラリに脆弱性が発見された場合の対応の遅れなどが問題となっている．これらの問題を解決するためにソフトウェア部品表（Software Bill of Materials，SBOM）の利用が推奨されているが，その活用を支援するツールは不足している．そこで，本研究では，SBOM の主要な記述形式の 1 つである SPDX 形式で作成された SBOM を用いたソフトウェアの適切な管理を，既存のツールを組み合わせて行う場合に比べてより省力化することを目的として，SPDX ドキュメントの管理を支援するツール「Osmy」を作成した．Osmy 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できる．また，Osmy によって SPDX ドキュメントの管理が省力化されることを確認し，Osmy が定期実行にあたって十分な速度で動作することも確認した．

**キーワード**　SBOM，SPDX，脆弱性

# A tool for vulnerability assessment and checksum verification using SPDX documents

Rio KISHIMOTO†, Tetsuya KANDA†, Yuki MANABE††, Katsuro INOUE†††, and Yoshiki HIGO†

† Graduate School of Information Science and Technology, Osaka University
1–5 Yamada, Suita, Osaka, 565–0871 Japan
††† Faculty of Science and Technology, Nanzan University
18 Yamazato, Showa, Nagoya, 466–8673 Japan
†† Faculty of Informatics, The University of Fukuchiyama 3370 Hori, Fukuchiyama, Kyoto, 620–0886 Japan

**Abstract**　Libraries are widely used in recent software development, but their management is insufficient, and there are problems such as delays in responding when vulnerabilities are discovered in the libraries. To solve these problems, the use of software bill of materials (SBOM) is recommended. However, there is a lack of tools to support its use. Therefore, in this study, we developed a tool called “Osmy” to support the management of SBOM created in the SPDX format, one of the major formats of SBOM, with the aim of making the appropriate management of software using SPDX documents more labor-saving than using existing tools. Osmy can automatically and periodically perform software vulnerability assessment and software corruption/falsification detection through checksum verification. We have confirmed that Osmy is fast enough to run on a regular basis, and that it reduces the amount of work required to manage SPDX documents.

**Key words**　SBOM，SPDX，Vulnerability

## 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広く利用されている．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 [1], [2]．ソフトウェアに含まれるセキュリティ上の欠陥は脆弱性と呼ばれ，脆弱

性の発見時には脆弱性が含まれないバージョンへの更新対応などを迅速に行う必要がある．しかし，ソフトウェアの管理は十分に行われておらず，ソフトウェア自身やソフトウェアが依存するライブラリの脆弱性への対応の遅れや対応漏れが問題となっている [3]．2021 年に深刻な脆弱性が発見されたロギングライブラリの Log4j では，2023 年 5 月時点においても，直近 1 か月間に日本から行われたダウンロードの 20% 程度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 [4]．

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 [5]．SBOM は，特定のソフトウェアを構成するパッケージやドキュメントなどの要素，それらのライセンス，および要素間の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，SPDX 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，利用しているソフトウェアやそのソフトウェアが依存するライブラリについて脆弱性の有無を確認したり，ソフトウェアの改ざんや破損の有無を確認したりすることができる．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 [6]．SPDX ドキュメントに記載された情報に基づいて，脆弱性診断とチェックサム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は，既存のツールを組み合わせることでも可能であるが，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として，SPDX ドキュメントの管理を支援するツール「Osmy」を作成した．

以降，**2.** では先行研究と課題について論じる．**3.** では脆弱性診断およびチェックサム検証の手法について記す．**4.** では Osmy の具体的な実装について記す．**5.** では Osmy の評価を行う．最後に，**6.** でまとめと今後の課題を述べる．

## 2. 先行研究と課題

### 2.1 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 [7] と OSV-Scanner [8] 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．指定された 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オープンソースソフトウェア（OSS）に関する脆弱性のデータベースである OSV データベース [9] から取得し，脆弱性の詳細情報を JSON 形式でファイルに出力する．

一方，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．指定された 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，脆弱性の詳細情報を Markdown の表形式や JSON 形式で標準出力に出力する．OSV-Scanner の入力とする SPDX ドキュメントにはパッケージの情報にパッケージ用の識別子である package URL [10] が記載されている必要があるが，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．

### 2.2 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，利用しているすべてのソフトウェアについて脆弱性診断を定期的に行うことが望ましい．しかし，**2.1** 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は，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，脆弱性診断の定期的な実行を支援する機能を持たない．

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，実際のシステム上に存在する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検証の手法について説明する．

### 3.1 脆弱性診断

ソフトウェアが実行時に依存するパッケージを識別し，識別された依存パッケージについて，SPDX ドキュメントの情報に基づいて 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．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などの実行時には依存しないパッケージの情報も記述可能であるため，SPDX 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，実行時に依存

コード 1　パッケージの情報の例

Code.1　Example of package information

```
{
  "name": "log4net",
  "SPDXID": "SPDXRef-Package-log4net",
  "downloadLocation": "NOASSERTION",
  "filesAnalyzed": false,
  "licenseConcluded": "NOASSERTION",
  "licenseInfoFromFiles": [ "NOASSERTION" ],
  "licenseDeclared": "NOASSERTION",
  "copyrightText": "NOASSERTION",
  "versionInfo": "2.0.8",
  "externalRefs": [],
  "supplier": "NOASSERTION"
}
```

コード 2　SPDX 要素間の関係情報の例

Code.2　Example of relationships between SPDX elements information

```
{
  "relationshipType": "DESCRIBES",
  "relatedSpdxElement": "SPDXRef-RootPackage",
  "spdxElementId": "SPDXRef-DOCUMENT"
},
{
  "relationshipType": "DEPENDS_ON",
  "relatedSpdxElement": "SPDXRef-Package-log4net",
  "spdxElementId": "SPDXRef-RootPackage"
}
```

するパッケージを識別する．

**3.1.1**　SPDX 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を記述した部分の例を示す．この例では，名前が log4net（2 行目），バージョンが 2.0.8（10 行目）であるパッケージ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．

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の例をコード 2 に示す．この例では，図 1 に示す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）がパッケージ（SPDXRef-RootPackage）を記述（DESCRIBES）することを示している．また，6 行目から 10 行目ではパッケージ（SPDXRef-RootPackage）がパッケージ（SPDXRef-Package-log4net）に依存（DEPENDS_ON）することが記述されている．

**3.1.2**　依存パッケージの識別と脆弱性情報の取得

SPDX ドキュメントに記載されたパッケージ間の関係情報

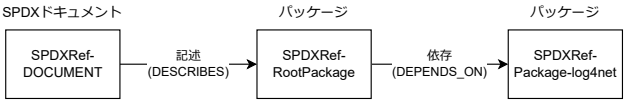


図 1　コード 2 で記述された関係

Fig. 1　Relationship described in Code.2

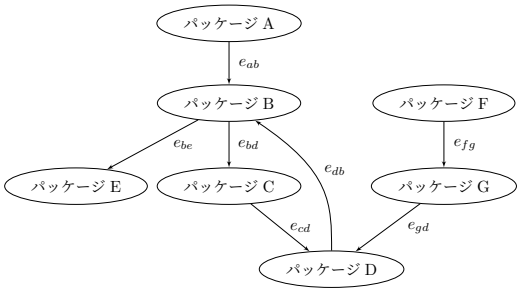


図 2　依存関係グラフの例

Fig. 2　Example of a dependency graph

に基づいて以下の方法で依存関係グラフを作成する．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，かつソフトウェアの実行時の脆弱性に影響するものに基づいてグラフに辺を追加する．

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 2 に示す．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 2 では，パッケージ A からパッケージ B に有向辺 $e_{ab}$ が存在しており，パッケージ A がパッケージ B に直接的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 に直接的にも間接的にも依存しないことが分かる．

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 に示す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．依存関係グラフではパッケージ A からパッケージ B への有向辺 $e_{ab}$ を追加する．表 2 に示す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．依存関係グラフではパッケージ B からパッケージ A への有向辺 $e_{ba}$ を追加する．表 3 に示す関係は，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，依存関係グラフでは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
Table 1　Relationships that if B is vulnerable, then A may also be vulnerable

| 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 |

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
Table 2　Relationships that if A is vulnerable, then B may also be vulnerable

| 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 |
| GENERATES | A は B を生成 |
| OPTIONAL_COMPONENT_OF | A は B の任意のコンポーネント |
| PATCH_APPLIED | A は B に適用されたパッチ |
| PREREQUISITE_FOR | A は B の前提条件 |
| DEPENDENCY_OF | A は B の依存要素 |
| OPTIONAL_DEPENDENCY_OF | A は B の任意の依存要素 |
| RUNTIME_DEPENDENCY_OF | A は B の実行時依存要素 |
| COPY_OF | A は B の複製 |
| PACKAGE_OF | A は B のパッケージ |
| VARIANT_OF | A は B の派生 |

表 3　一方が脆弱性を含めば他方も脆弱性を含む可能性がある関係
Table 3　Relationships that if one is vulnerable, then the other may also be vulnerable

| 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 |

依存パッケージの識別は，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）からの深さ優先探索によって行う．探索中に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，ソフトウェアの実行時の依存パッケージである．

識別されたソフトウェアの実行時の依存パッケージについて，パッケージが脆弱性を含むかを OSS の脆弱性を格納したデータベースである OSV データベースに問い合わせる．依存パッケージが脆弱性を含む場合は，その詳細情報を OSV データベースから取得する．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 [9] 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，広く利用されているライブラリの情報は複数の SPDX ドキュメントに重複して含まれるため，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．

コード 3　ファイルの情報の例
Code.3　Example of file information

```
{
  "fileName": "./log4net.dll",
  "SPDXID": "SPDXRef-File--log4net.dll",
  "checksums": [
    {
      "algorithm": "SHA1",
      "checksumValue": "40fdba136f864c8a2f3e3f9c9
          e3949f7582a6077"
    }
  ],
  "licenseConcluded": "NOASSERTION",
  "licenseInfoInFiles": [ "NOASSERTION" ],
  "copyrightText": "NOASSERTION"
}
```

### 3.2　チェックサム検証による改ざん・破損検出

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，各ファイルの名前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の例をコード 3 に示す．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-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，異なるアルゴリズムで計算された複数のハッシュ値を記述可能であるが，その内 SHA-1 ハッシュ値の記載は必須である．そのため，チェックサム検証処理では実際のシステム上に存在するファイルから計算した SHA-1 ハッシュ値と，SPDX ドキュメントに記載された値が一致することを確認する．値が一致しない場合はファイルの改ざんや破損が発生していると考えられる．

## 4.　ツールの実装

**3.** で説明した手法に基づいて SPDX ドキュメントの管理を行うツール「Osmy」を実装した．

### 4.1　開発言語と想定環境

Osmy は C#を用いて作成されており，実行には.NET 6.0 以上が必要である．また，SPDX ドキュメントは tag-value 形式，JSON 形式，XML 形式など様々な形式で記述することができるが，Osmy は tools-java [11] を用いて，それらを統一的に管理しているため，Java 11 以上が実行できる必要がある．なお，Osmy は SPDX のバージョン 2.2.2 [12] 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．

### 4.2　ツールの構成

図 3 に Osmy の構成を示す．Osmy 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

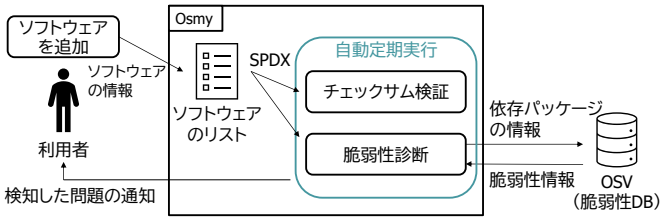


図 3　ツールの構成

Fig. 3　Composition of our tool

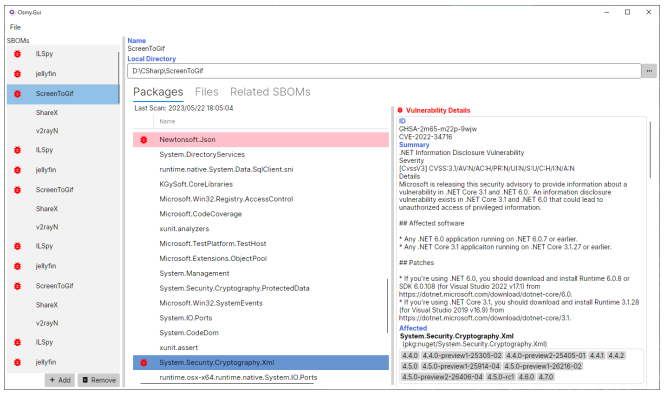

図 4　GUI クライアントの画面画像

Fig. 4　Screenshot of the GUI client

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検証を行う．ソフトウェアのリスト，チェックサム検証の結果，脆弱性診断の結果は，Osmy 内部の SQLite データベースに保存される．脆弱性診断とチェックサム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．通知は電子メールによって行い，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．

Osmy は，ソフトウェアのリストの管理と，脆弱性診断およびチェックサム検証を行うサーバープログラムと，サーバープログラムが持つ情報を操作するための GUI クライアントおよび CLI クライアントからなる．GUI クライアントの画面画像を図 4 に示す．GUI クライアントでは画面左部に管理対象のソフトウェアを一覧表示する．一覧で選択されたソフトウェアについて脆弱性診断の結果とチェックサム検証の結果を画面右部に表示し，脆弱性診断の結果表示では脆弱性を含むパッケージを赤く強調表示する．ソフトウェアの追加や削除は画面左下部のボタンから行うことができる．CLI クライアントにおいても GUI クライアントと同様の機能を提供する．

## 5.　評　　価

Osmy が実現する脆弱性診断およびチェックサム検証の定期実行手法と性能の評価を行い，Osmy の有用性を確認した．

### 5.1　手法の評価

#### 5.1.1　定期的な脆弱性診断の省力化

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，Osmy を用いて行う場合を比較する．

spdx-to-osv は引数で指定した SPDX ドキュメントについて，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える．なお，脆弱性の検出有無については出力されたファイルを確認する必要があり，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．

一方，Osmy では管理対象として追加された SPDX ドキュメントについて，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．自動実行で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．

このように，実行毎に必要であった診断結果の確認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る．また，脆弱性検出時に通知が行われるため，結果の確認忘れによる脆弱性の見過ごしが発生する可能性の低下も期待できる．

#### 5.1.2　定期的なチェックサム検証の省力化

チェックサム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，Osmy を用いて行う場合を比較する．

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 コマンドがある．このコマンドはファイルパスとハッシュアルゴリズムを指定すると，ファイルのハッシュ値が指定したアルゴリズムで計算するものである．SPDX ドキュメントに記載された各ファイルのハッシュ値をこのコマンドを用いて計算し，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検証が行える．

一方，Osmy では，管理対象として追加された SPDX ドキュメントについて，あらかじめ設定した間隔でチェックサム検証が定期実行される．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，検証結果の確認を定期実行毎に行う必要はない．

このように，チェックサム検証に必要であった手動作業が大幅に削減されるため，Osmy は有用であると考えられる．

### 5.2　ツールの性能評価

#### 5.2.1　評 価 方 法

Osmy が脆弱性診断およびチェックサム検証に要する時間を計測し，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検証の対象となる SPDX ドキュメントは，表 4 に示す 5 個の OSS から sbom-tool [13] を用いて作成した．脆弱性診断とチェックサム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，CPU が Intel Core i7-11700K であり，32GB の RAM を搭載する Windows10 マシンで行った．

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

表 4　SPDX ドキュメントの作成に用いた OSS
Table 4　OSS used to create SPDX documents

| | パッケージの数 | ファイル数 |
|---|---|---|
| v2rayN [14] | 36 | 60 |
| ShareX [15] | 11 | 557 |
| jellyfin [16] | 162 | 177 |
| ScreenToGif [17] | 75 | 1 |
| ILSpy [18] | 267 | 38 |
| 合計（ユニーク） | 551（525） | 833（833） |

表 5　Osmy の処理時間
Table 5　Time to perform vulnerability scan and checksum verification

| | 脆弱性診断（秒） | チェックサム検証（秒） |
|---|---|---|
| 1 | 8.7454216 | 0.8970842 |
| 2 | 6.9176972 | 0.8004870 |
| 3 | 7.6308489 | 0.9428804 |
| 4 | 6.2152772 | 0.8288414 |
| 5 | 9.9897929 | 0.9570103 |
| 平均 | 7.8998076 | 0.8852607 |

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．

#### 5.2.2 計 測 結 果

処理時間の計測結果を表 5 に示す．脆弱性診断の実行時間の平均値は約 8.6 秒であり，チェックサム検証の実行時間の平均値は約 5.3 秒であった．

#### 5.2.3 考　　察

計測結果から，Osmy は，脆弱性診断を 1 時間あたり約 450 個のソフトウェアについて，チェックサム検証を 1 時間あたり約 480 個のソフトウェアについて実行可能であると推定される．この推定では，Synopsys による調査 [19] に基づいて 1 ソフトウェアに含まれる OSS パッケージの数を平均 528 個と仮定した．また，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に基づいて，1 ソフトウェアに含まれるファイルの数を平均 7000 個と仮定した．

筆者が所属する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た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，Osmy を用いて脆弱性診断とチェックサム検証を行う場合の処理時間を上記の推定に基づいて計算すると，脆弱性診断の処理時間は約 1.4 時間，チェックサム検証の処理時間は約 1.3 時間である．したがって，Osmy 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検証を定期的に自動実行するツール「Osmy」を作成した．Osmy 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．

Osmy の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いるため，OSS 以外の脆弱性情報を格納するデータベースからも情報を取得することで，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ている．また，今回行った評価では，管理対象として用いる十分な数の SPDX ドキュメントを用意できなかったため，Osmy で管理した SPDX ドキュメントの数は少なかった．より多くの 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実施し，Osmy の改善を行いたい．

**謝辞**

本研究は, JSPS 科研費 JP23H03375, JP21K02862, JP19K20239, JP20H04166, JP21K18302, JP21K11829, JP21H04877, JP22H03567, JP22K11985，2023 年度南山大学パッヘ研究奨励金 I-A-2 の助成を得て行われた．

## 文　　献

[1] O.P.N. Slyngstad, A. Gupta, R. Conradi, P. Mohagheghi, H. Rønneberg, and E. Landre, “An empirical study of developers views on software reuse in statoil asa,” Proceedings of the 2006 ACM/IEEE International Symposium on Empirical Software Engineering, pp.242–251, ISESE ’06, 2006. https://doi.org/10.1145/1159733.1159770
[2] Y. Wang, B. Chen, K. Huang, B. Shi, C. Xu, X. Peng, Y. Wu, and Y. Liu, “An empirical study of usages, updates and risks of third-party libraries in java projects,” 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp.35–45, 2020.
[3] M. Alfadel, D.E. Costa, and E. Shihab, “Empirical analysis of security vulnerabilities in python packages,” 2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), pp.446–457, 2021.
[4] sonatype, “Log4j Exploit Updates,” https://www.sonatype.com/resources/log4j-vulnerability-resource-center.
[5] 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.
[6] B. Xia, T. Bi, Z. Xing, Q. Lu, and L. Zhu, “An empirical study on software bill of materials: Where we stand and the road ahead,” Proceedings of the 2023 IEEE/ACM 45th International Conference on Software Engineering, pp.2634–2646, 2023.
[7] “spdx-to-osv,” https://github.com/spdx/spdx-to-osv.
[8] “OSV-Scanner,” https://github.com/google/osv-scanner.
[9] “OSV,” https://osv.dev/.
[10] “package URL,” https://github.com/package-url/purl-spec.
[11] “tools-java,” https://github.com/spdx/tools-java.
[12] “Spdx v2.2.2,” https://spdx.github.io/spdx-spec/v2.2.2/.
[13] “sbom-tool,” https://github.com/microsoft/sbom-tool.
[14] “v2rayN,” https://github.com/2dust/v2rayN.
[15] “ShareX,” https://github.com/ShareX/ShareX.
[16] “jellyfin,” https://github.com/jellyfin/jellyfin.
[17] “ScreenToGif,” https://github.com/NickeManarin/ScreenToGif.
[18] “ILSpy,” https://github.com/icsharpcode/ILSpy.
[19] 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.