再利用されたライブラリに対するバージョン検出を利用した脆弱性検知ツール

杉森 遼[1]　伊藤 薫[2,a)]　神田 哲也[2,b)]　井上 克郎[2,c)]

概要：近年，ソフトウェアの脆弱性を利用した攻撃による個人情報の大規模漏洩などが発生している．開発中のソフトウェアにおいて既存のソースコードを再利用する際には，その再利用したコードにおける脆弱性への対処などのメンテナンスも重要な関心事である．再利用しているライブラリに対する脆弱性検出ツールは多数存在しており，例えば GitHub の標準機能である Dependabot では，ライブラリの依存関係が記述されたファイルを解析し，その情報から脆弱性情報の検出・通知を行う．しかし，このような手法はパッケージマネージャの管理下にないソフトウェアに対しては使用することができない．そこで本研究では，分析対象のソフトウェアとそこで再利用しているライブラリをソースコードレベルで比較し，脆弱性情報を出力するツールを開発した．また，ライブラリ側の脆弱性情報が公開されてからソフトウェア側で解消されるまでの期間の調査を行い，本ツールが有効な状況が存在することを確認した．

1. はじめに

ソフトウェアの開発現場において，オープンソースソフトウェア（Open Source Software，以降 OSS）を再利用することが一般的に行われている．OSS を再利用することで，独自に開発したものに比べて，信頼性や安定性が向上し，開発コストを削減できるためである [1]．しかし，OSS の再利用には，その中に含まれる脆弱性を同時に取り込む可能性があるという問題点が存在する．

脆弱性とは，プログラムの不具合や設計上のミスが原因で発生する情報セキュリティの欠陥のことを指す．近年では，脆弱性を利用した攻撃が高度化・複雑化しており，過去には大規模な個人情報の漏洩が発生している．脆弱性を用いた攻撃をさせないためには，開発者は開発しているソフトウェアに対して，脆弱性が含まれているかどうかを確認する必要がある．

ソースコードの脆弱性検出についての研究には，公開されている脆弱性情報を基に，脆弱性の検出を行う脆弱性更新通知機能 [2]，機械学習を用いての脆弱性検出 [3] などがある．これらの既存研究では脆弱性の解決策を提示しない点や実行時間などのコストへの考慮がないことから，メンテナンスについての支援は不十分である．

[1] 大阪大学基礎工学部情報科学科
[2] 大阪大学大学院情報科学研究科
[a)] ito-k@ist.osaka-u.ac.jp
[b)] t-kanda@ist.osaka-u.ac.jp
[c)] inoue@ist.osaka-u.ac.jp

これらの問題に対処するために，分析対象のソフトウェアとそこで再利用しているライブラリを比較し，脆弱性情報と再利用ライブラリ内の脆弱性修正に関する情報の出力を行う手法を提案し，ツールを開発した．提案ツールでは，ライブラリのバージョン検出に，軽量な類似度計算によるプロジェクト間のソースファイル集合の再利用検出手法を用いる [4]．この手法を用いることで，ソースコードを入力として与えるだけで，再利用しているライブラリのバージョン検出を行うことができる．脆弱性情報の検索には，CVE-Search という脆弱性情報の検索や処理を行うための OSS を使用する．CVE-Search にライブラリ名を入力して得られた脆弱性情報をもとに，バージョン番号などの情報を用いてフィルタリングする．その後，ライブラリの脆弱性修正情報とともに脆弱性情報を出力する．

評価実験では，提案ツールの性能を評価するために，開発ツールから出力される脆弱性情報と用意した正解集合を比較し，精度を計測した．また，脆弱性検出結果を用いて，脆弱性残留期間の調査と脆弱性の深刻度との関係性の調査を行い，ツールの有効性を確認した．

以降，2 章では本研究の背景として，ライブラリの再利用，脆弱性及び既存の脆弱性検出ツールについて説明する．3 章では本研究で開発したツールについて説明する．4 章ではツールの評価実験について，5 章ではツールを用いた調査についてそれぞれ説明する．6 章で妥当性への脅威について述べ，最後に 7 章ではまとめと今後の課題を記述する．

2. 背景

2.1 ライブラリの再利用

ソフトウェアを開発する際，開発者は機能の実装を独自に行うだけでなく，既にあるライブラリなどを再利用することがある．これにより，ソフトウェアの開発に必要となる機能の一部を実装する必要がなくなり，開発コストを削減できる．

また，ライブラリとして提供されているソフトウェアの中には，OSS として公開されているものがある．OSS は，独自あるいは組織内のみで開発されたソフトウェアに比べて，不特定多数の開発者がそのソフトウェアを使用，またはメンテナンスを行っている．そのため，独自での実装に比べて検証が十分に行われているため，信頼性が高いとされている [1]．このようなライブラリを再利用する場合，バイナリ形式でリンクするほかにソースコードを自身のソフトウェアにコピーする方法がある．ソースコードは元のまま使われるほか，利用するソフトウェア側で必要に応じて改変が行われる．

再利用しているライブラリには後から脆弱性が発見される場合があるため，開発したソフトウェア中に含まれる再利用しているライブラリに関して更新作業を行う必要がある．開発したソフトウェアのドキュメントやバージョン管理システムに残っている履歴内に，どのライブラリのどのバージョンをどこで利用しているのかが記載され管理されていれば，ライブラリの更新作業は比較的容易と言える．

しかし，担当者の変更やソフトウェア中の構成の変更などが生じることにより， 再利用した際の情報が正しく記載されていないことがあると Xia らにより報告されている [5]．Xia らはまた，プロジェクトの開発期間が長くなるにつれ，ソフトウェアの再利用情報が失われることがあることも報告している．さらに，開発者はライブラリを取り込んだ後，そのライブラリを独自に編集して利用することがある．このような場合，ファイルの完全一致では再利用されたライブラリのバージョンが判断できない．

2.2 ライブラリバージョン検出手法

ソースコードのコピーによるライブラリの再利用において，再利用されたライブラリのバージョン情報が欠落することがある．このような状況に対して，伊藤らは類似度計算に b-bit MinHash 法 [6] を用いた軽量で高精度な再利用検出手法を提案している．この手法では，入力として，分析対象のソフトウェアにおいてライブラリから再利用したファイルを格納しているディレクトリ D_Q とライブラリのバージョン管理システムのリポジトリを与えている．与えられた入力からディレクトリ D_Q とリポジトリの各バージョン間の類似度計算を行い，再利用されているライブラリのバージョンを出力する．入力をディレクトリとした理由について，再利用したライブラリのファイル群が 1 つのディレクトリにまとめて配置されることが多いためであると論文中で記述されている．なお，ここでのバージョンとは，開発者がライブラリを公開しているバージョン管理システム中で，タグと呼ばれるラベルが紐付けられているリビジョンを指している．

バージョンの特定には，ファイル間の類似度の合計を用いている．具体的には，ファイル間の類似度が与えられたとき，ライブラリのあるバージョンに対する入力ファイル群の類似度が最大となるようなバージョンを求めている．

2.3 脆弱性

脆弱性とは，ソフトウェアにおいて，プログラムの不具合や設計上のミスが原因で発生する情報セキュリティの欠陥のことである．過去には脆弱性を用いた情報漏洩が発生している．2017 年 3 月には，Java で開発されたライブラリである Apache Struts2 に任意でコードを実行できる脆弱性「CVE-2017-5638」が公開された．この脆弱性に対する修正パッチは脆弱性情報公開からすぐに公開されていたが，公開直後から攻撃が発生し，不正アクセスによる大規模な情報流出の可能性も発表された [7]．また，このパッチの適用を行っていなかったため，5 月から 7 月の間にこの脆弱性を利用され，情報漏洩するに至った事例もある [8]．

脆弱性情報を一元的に扱うことができるよう，識別番号やタイプ一覧などが整備されている．以下，本ツールでも扱う情報と，その検索ツールについて説明する．

2.3.1 CVE

CVE とは，共通脆弱性識別子のことであり，脆弱性を一意に識別するために必要となる番号である [9]．脆弱性検出ツールや脆弱性対策情報提供サービスの多くで利用されている．この識別番号を付与することにより，別組織が発行する脆弱性対策情報が同じ脆弱性に関する対策情報であることの判断や，対策情報同士の相互参照や関連付けが可能となる．

2.3.2 CWE

CWE とは，共通脆弱性タイプ一覧のことであり，脆弱性の種類を判別するためのものである [10]．脆弱性には様々なものがあり，アプリケーションのセキュリティ上の不備を意図的に利用し，データベースシステムを不正に操作する SQL インジェクション，Web サイトに，悪意のある第三者が罠を仕掛け，サイト訪問者の個人情報などを盗むクロスサイト・スクリプティングなどがある．このような脆弱性を識別するための，脆弱性の種類の一覧を体系化して提供している．

2.3.3 CVSS

CVSS とは，共通脆弱性評価システムのことであり，脆弱性の深刻度を評価するための手法である [11]．CVSS は 3 つの基準で脆弱性を評価する．一つ目の基準である基本

```
{
  "cvss": 7.5,
  "cwe": "CWE-189",
  "id": "CVE-2016-9841",
  "references": [
    "http://lists.opensuse.org/opensuse-updates/
     2016-12/msg00127.html",
    (中略)
  ]
}
```


図 1 CVE-Search の WebAPI インターフェースからの出力

評価基準は，脆弱性そのものの特性を評価する基準である．これは，情報システムに求められる 3 つのセキュリティ特性である機密性・完全性・可用性に対して，ネットワークから攻撃可能であるどうかという基準で評価される．二つ目の基準である現状評価基準は，脆弱性の現在の深刻度を評価する基準である．これは，その脆弱性を攻撃する攻撃コードの有無や対策情報が利用可能であるかといった基準で評価される．三つ目の基準である環境評価基準は，脆弱性を含む製品の利用者の利用環境も含め，最終的な脆弱性の深刻度を評価する基準である．これは，攻撃を受けた際の最終的な脆弱性の深刻さを二次被害の大きさや対象ソフトウェアの使用状況といった基準で評価される．値域は [0,10] であり，数値が大きいほど深刻な脆弱性であることを示す．

2.3.4 CVE-Search

CVE-Search は，CVE の情報を DB から検索可能にするツールである．図 1 はライブラリ zlib を入力とした際の出力の一部である．出力される脆弱性情報には，CVE の他に CWE，CVSS などが含まれている．図 1 では id が CVE を表す．この他に，参照情報や脆弱性が関係する製品やバージョンについての情報が含まれる．

2.4 脆弱性検出ツール

脆弱性による被害は深刻なものであり，それがライブラリなどに含まれている可能性があることから，脆弱性を利用した攻撃が行われる前にそれを検出することが重要である．GitHub というソフトウェア開発プラットフォームで使用できる Dependabot では，ライブラリのバージョン情報を記録するパッケージマネージャが自動生成したファイルを解析し，その情報から脆弱性情報の検出・通知を行なっている [12]．しかし，パッケージマネージャが自動生成したファイルが存在しない場合は使用することができない．

また，Balzarotti らのツールでは，分析するアプリケーションの一部を実行し，可能性のあるクロスサイト・スクリプティングや SQL インジェクション攻撃に対応する文字列の注入を行い，脆弱性の検出を行っている [13]．Chen らのツールでは，欠陥追跡システムやコミット，メーリングリストなど様々なソースを学習データとした機械学習モデルを用いて脆弱性の予測を行っている [3]．Wang らはファジングと呼ばれるソフトウェアの不具合を発見するためのテストを活用し，誤り検知符号の 1 つであるチェックサムの整合性から脆弱性を検出している [14]．これらは，脆弱性の検出を主な目的としており，脆弱性の解決策を提示しない点や実行時間などのコストへの考慮がないことから，メンテナンスについての支援は不十分である．

3. ツールの実装

本研究では，ソースコードからバージョンを検知し脆弱性検出を行うことを目的として，入力されたソフトウェアで再利用しているライブラリに由来するソースコード群が，判明している脆弱性を含んでいるのかどうか検出するツールの提案と試作を行う．

開発するツールは，利用者がソースファイル群と再利用元のライブラリを入力として与えると，再利用されたと考えられるファイルをソースファイル群の中から検出し，それらに対する脆弱性情報や関連ファイル，ファイル差分，最新バージョンの情報を出力する．手順は以下の通りである．

(1) 分析対象ソフトウェアとライブラリを指定
(2) バージョン検出
(3) 脆弱性情報の検索
(4) フィルタリング
(5) 検出された脆弱性情報と最新バージョンにおける修正情報を出力

手順 (1) では，ツール使用者が指定した分析対象のソフトウェアにおいて，ライブラリから再利用したファイルを格納しているディレクトリとライブラリのバージョン管理システムのリポジトリを入力に与える．

手順 (2) では，手順 (1) の入力を伊藤らの手法を用いて比較し，再利用したライブラリのバージョンを取得する．伊藤らの手法では，類似度の合計値が一致した場合，複数のバージョンが検出される．その場合は，検出された中でリリース日が最も新しいバージョンを使用する．

手順 (3) では，再利用したライブラリ名を用いて，CVE-Search から脆弱性情報を検索する．ここで用いるライブラリ名は，手順 (1) で用意したリポジトリの名前である．

手順 (4) では，取得したバージョン情報と再利用したライブラリ名を用いて，手順 (3) で検索した脆弱性情報のフィルタリングを行う．

手順 (5) では，手順 (4) でフィルタリングした関係する脆弱性情報，及び関連するファイル名とその差分を出力する．また，脆弱性解決の手段として最新バージョンについての通知を出力する．

ここで，本手法のうち既存ツールによらない手順 (4) と

```
cpe:2.3:{種別}:{ベンダ名}:{製品名}:{バージョン}:{アップデート}:{エディション}:{言語}:{ソフトウェアエディション}:{ターゲットソフトウェア}:{ターゲットハードウェア}:{その他}
```

図 2 CPE の FormattedString 形式の構成

手順 (5) の詳細について詳しく説明する．

3.1 手順 (4)：フィルタリング

手順 (3) での検索はライブラリ名によるキーワード検索であり，入力した単語と一致した関係のない脆弱性情報が偶然含まれる．また，バージョン番号の表現方法が手順 (2) における出力と異なる．そのため，適切な脆弱性検出のためにフィルタリングを行う．

フィルタリングには，手順 (3) の出力のうち関連製品情報に利用されている CPE (Common Platform Enumeration)[15] というソフトウェアなどを識別するための名称基準を用いる．CPE のテキスト表現は図 2 のようになる．

例えば，「CVE-2017-12652」の関連製品情報の一部は図 3 のようになる．このときの*は CPE の各分類に対して全てのものを指す記号である．「バージョン」の項目に着目すると，libpng のバージョン 1.2.38 からバージョン 1.2.41 が関連製品・バージョンであることが示されている．CPE においては，`beta` や `rc` のようなバージョン表記はバージョンとは別の「アップデート」という項目で扱われている．そこで，今回の手法では，評価実験の対象のライブラリのバージョン検出結果に含まれる文字列 beta と rc を特別な文字列として扱い，これらの特別な文字列を含むかどうかでフィルタリング手法を分け，脆弱性情報のフィルタリングを行う．

3.1.1 バージョン表記に特別な文字列を含まないときのフィルタリング

バージョン情報に特別な文字列を含まない場合，「アップデート」の項目は使用せず，フィルタリングには「製品名」と「バージョン」を CPE に合わせて結合させた文字列を使用する．ライブラリ libpng のバージョン 1.2.38 の場合，ライブラリ名である libpng を「製品名」とし，ライブラリのバージョン情報である 1.2.38 を「バージョン」として文字列`:libpng:1.2.38:`が関連製品情報内に含まれているかどうかでフィルタリングを行う．

3.1.2 バージョン表記に特別な文字列列を含むときのフィルタリング

脆弱性情報によっては「アップデート」を示す文字列を使用せずに，「バージョン」のみで表現していることがあるため，単純に「アップデート」の文字列を含むフィルタリングが行えない．そのため，バージョン情報に特別な文字

```
cpe:2.3:a:libpng:libpng:1.2.38:*:*:*:*:*:*:*
cpe:2.3:a:libpng:libpng:1.2.39:*:*:*:*:*:*:*
cpe:2.3:a:libpng:libpng:1.2.39:beta1:*:*:*:*:*:*
cpe:2.3:a:libpng:libpng:1.2.39:beta2:*:*:*:*:*:*
cpe:2.3:a:libpng:libpng:1.2.39:beta3:*:*:*:*:*:*
cpe:2.3:a:libpng:libpng:1.2.39:beta4:*:*:*:*:*:*
cpe:2.3:a:libpng:libpng:1.2.40:*:*:*:*:*:*:*
cpe:2.3:a:libpng:libpng:1.2.41:*:*:*:*:*:*:*
```

図 3 CVE-2107-12652 の関連製品情報の一部

```
1
分析対象のライブラリ curl のバージョンは最新のものではありません（現バージョン: curl-7_50_1）
最新バージョンは curl-7_74_0 です
最新のバージョンでは，再利用しているバージョンから 26 個の脆弱性が修正されています
```

図 4 ツールの実行例 1

列を含む場合，以下の 3 種類の文字列を用いてそのいずれかを含む CPE が関連製品情報内に含まれているかどうかでフィルタリングを行う．

- 製品名+バージョンのみ
- 製品名+バージョン+アップデート
- 製品名+バージョン+アップデート (数字部分を除く)

例えば libpng のバージョン 1.2.39beta2 の場合，以下の文字列を用いる．

- `:libpng:1.2.39:*`
- `:libpng:1.2.39:beta2:*`
- `:libpng:1.2.39:beta:*`

3.2 手順 (5) における出力例

Android で再利用されているライブラリ curl の脆弱性検出を行う例によりツールの出力を説明する．この例で使用した Android のバージョンは `android-7.1.1_r59` であり，再利用されている curl のバージョンは実行時点では古いものとなっている．このとき，ツールの出力は図 4、5、6 のようになる．説明のため本来の出力にはない枠を付してある．

ツールの出力は大きく 3 パートに分かれる．まず図 4 では，再利用しているライブラリのバージョン検出結果と，そこからそのライブラリの最新バージョンまでの脆弱性の修正件数を出力する．この例では curl の最新バージョンは `android-7.1.1_r59` 内で再利用しているバージョンより新しいリリースであり，その間に CVE に登録されている 26 の脆弱性が修正されていることがわかる．次に，図 5 でソフトウェア側と最新バージョン間でのファイル間の違いや追加，削除についての情報を出力する．この情報により，脆弱性に対処するためにはどのファイルを修正する必要が

```
2
以下のファイルでバージョン間に違いがあります
--------------------------------
include/curl/curl.h
include/curl/curlrules.h
(中略)
--------------------------------
以下のファイルが curl-7_50_1 と curl-7_74_0 の間で大規模
な変更，または追加・削除されています
--------------------------------
lib/curl_addrinfo.c
lib/curl_addrinfo.h
lib/curl_base64.h
(中略)
```

図 5 ツールの実行例 2

```
3
--------------------------------
脆弱性一覧
--------------------------------
CVE: CVE-2018-1000301
summary: curl version curl 7.20.0 to and including
curl 7.59.0 contains a CWE-126: Buffer Over-read
vulnerability in denial of service that can result
in curl can be tricked into reading data beyond the
end of a heap based buffer used to store downloaded
RTSP content.. This vulnerability appears to have
been fixed in curl < 7.20.0 and curl >= 7.60.0.
CVSS: 6.4
CWE: 125
CWE_Name: "Out-of-bounds Read"
CWE_Description:
"The software reads data past the end, or before
the beginning, of the intended buffer."
--------------------------------
```

図 6 ツールの実行例 3

あるか検討できる．また，ソフトウェア側と最新バージョン間の差分であるので、脆弱性対応のパッチを既に適用している場合にはこの一覧にはファイルが表示されない．最後に，図 6 のように，再利用しているバージョンに含まれる脆弱性情報について，CVE-Search から得た情報の出力を行う．

4. 評価実験

提案手法の有用性を評価するため，提案手法の出力結果と手作業で収集した正解集合を用いて検出された脆弱性情報の精度を計測する．また，実行時性能を評価するために，提案ツールの実行時間を計測する．本章では，実験対象と実験方法を説明したのちに，評価を行う．

4.1 実験対象

実験対象として，伊藤らの実験で使用されたソフトウェアと同じ 5 つのソフトウェアと 4 つのライブラリのプロジェクトを使用した．これは，再利用したライブラリの記録がドキュメント等により管理されているためである．また，規模が異なることや開発コミュニティが異なることを基準として，対象が選択されているためである．さらに，

表 1 実験対象のソフトウェア・ライブラリと解析対象バージョン数

リポジトリ名	バージョン数
android(libpng)	76
android(curl)	61
android(ogg)	31
android(zlib)	69
apitrace	12
fs2open	797
gecko-dev	98
v8monkey	72
libpng	1615
curl	200
ogg	12
zlib	72

既存手法ではこれらの対象に対して高い精度でのバージョン検出を行うことができているため，既存手法の精度が提案手法全体の精度に与える影響を小さくできる．

対象としたソフトウェア及びライブラリのバージョン数は，表 1 のようになる．全てのプロジェクトはバージョン管理システムの Git で管理されいる．また，表 2 にこれらのソフトウェアと利用されているライブラリの組合せを示す．

4.2 実験方法

本ツールの入力に表 2 のソフトウェア・ライブラリの組を与え，実験を行う．脆弱性検出の正確さは，実験対象のソフトウェアの各バージョンで再利用しているライブラリのバージョンの脆弱性情報検出結果が，正解集合と一致した割合によって求める．正解集合は，JPCERT/CC と IPA が共同運営している脆弱性対策情報ポータルサイト JVN [16] から今回の実験対象であるライブラリの脆弱性を手作業で収集して構築した．ライブラリやソフトウェア名で検索を行い，検索して得た脆弱性情報の中から，関連のある脆弱性情報を手作業で分別し，正解集合とした．

4.3 評価方法

4.2 節で述べた方法で，データセット Answer を用意する．提案手法に，ソフトウェアのライブラリから再利用したファイルを格納しているディレクトリとライブラリのリポジトリを入力として与え，ソフトウェアのバージョンごとの検出結果の集合 Result を得る．その後，用意した正解集合 Answer と Result を比較する．Result 中の，Answer

表 2 実験対象のソフトウェア・ライブラリの組み合わせ

ソフトウェア	libpng	curl	ogg	zlib
android	✓	✓	✓	✓
apitrace	✓			✓
fs2open	✓			✓
gecko-dev	✓		✓	✓
v8monkey	✓			✓

表 3 提案手法の実行結果

ソフトウェア	ライブラリ名	ソフトウェアのバージョン数	脆弱性検出バージョン数	検出数	正解数	Precision	Recall	実行時間 (ms)
android(libpng)	libpng	76	76	385	385	1	1	47,818
android(curl)	curl	61	46	739	739	1	1	100,940
android(ogg)	ogg	31	0	0	0	N/A	N/A	214,769
android(zlib)	zlib	69	23	92	92	1	1	38,265
apitrace	libpng	12	12	24	24	1	1	30,309
	zlib	12	8	32	32	1	1	32,726
fs2open	libpng	797	711	1,605	1,577	0.983	1	28,646
	zlib	797	146	584	584	1	1	33,993
gecko-dev	libpng	98	83	678	678	1	1	71,167
	ogg	98	0	0	0	N/A	N/A	208,845
	zlib	98	0	0	0	N/A	N/A	29,958
v8monkey	libpng	72	72	1,440	1,440	1	1	139,968
	zlib	72	0	0	0	N/A	N/A	29,815
全体		2,293	1,177	5,579	5,551	0.995	1	48,515

に含まれる脆弱性情報が含まれている数を数え，適合率 Precision と再現率 Recall を評価する．

4.4 結果

表 3 にソフトウェアとライブラリの組み合わせごとの正確さを計測した結果を示す．検出結果と収集した脆弱性情報を比較したところ，全体として Precision が 0.995，Recall が 1 となった．また，多くの組み合わせで Precision と Recall が 1 となった．Recall が高いほど検出漏れが少ないため，Recall が 1 であることから今回の実験データにおいては脆弱性を全て検出できた．Precision が 1 ではなかった fs2open と libpng の組み合わせでは，ライブラリの特定のバージョンの脆弱性検出において，そのバージョンのベータ版のみに含まれる脆弱性が検出された．この脆弱性は再利用しているライブラリのバージョンであるリリース版では修正されているため，不必要な脆弱性情報だった．一方で，libpng は「アップデート」に関する文字列を含むバージョンが検出されたが，これらのバージョンの脆弱性情報も検出できていることが確認できた．以上のことから，提案手法は検出されたバージョン情報から高い精度で脆弱性を検出することが可能であると言える．

4.5 提案ツールの実行時性能

提案手法の実行時性能を評価するために，脆弱性検出にかかる実行時間を測定する．ライブラリのバージョン検出に要する時間は伊藤らの研究結果により高速であることが確認できているため，ここではバージョン検出実行後から脆弱性情報とその修正情報を出力するまでに要した時間を計測する．計測する計算機環境の OS は macOS Mojave 10.14.6，CPU は Intel Corei5-8279U 2.4GHz， RAM は LPDDR3 2,133MHZ 8GB，ストレージは PCI-Express 接続の 512GB のもの，Java の実行環境は，OpenJDK11 である．時間の計測は，Java のシステムメソッドの一つである nanoTime メソッドの呼び出しを実装プログラム中に記述することで行う．また，マルチスレッド処理は使用せず，単一のスレッドで処理を行う．

表 3 に，提案ツールにおいてライブラリのバージョンを検出してから脆弱性情報とその修正情報を出力するまでに要した時間を計測した結果を示す．平均で 48,515ms，最大で 214,769ms，最小で 28,646ms の時間がかかった．ogg は脆弱性情報が検出されていなかったが，無関係な脆弱性情報が多数検出されたため，実行時間が長くなった．本手法では伊藤らのツールの実行時間に加えて，CVS-Search での検出時間とその結果に対してフィルタリングする時間が必要になる．今回の対象に対して，提案ツールはどの組み合わせでも実用的な時間で脆弱性情報を検出できることが確認できた．

5. 提案手法を用いた調査

開発したツールの有用性を示すため，ツールの応用例として，ツールを用いて再利用元のライブラリの脆弱性が公開されてから解消されるまでの脆弱性残留期間を調査する．以下ではこの日数のことを脆弱性残留期間と表記する．また，脆弱性残留期間と脆弱性の深刻度を示す CVSS との相関を調査する．調査対象は，4.1 節と同様のソフトウェアと再利用元のライブラリの各バージョンの組み合わせを用いる．

5.1 調査方法

試作したツールを用いて，各ライブラリの脆弱性情報が再利用先のソフトウェアのどのバージョンで検出されなくなるかを調べる．検出されなくなったバージョンの公開日をリポジトリから取得し，脆弱性情報の公開日からその日までを脆弱性残留期間とする．ソフトウェアごとの脆弱性

残留期間の対象は，ソフトウェアごとで再利用しているライブラリの脆弱性がそのソフトウェアにおいて解消されたソフトウェアのバージョンが公開されるまでの期間である．ライブラリごとの脆弱性残留期間の対象は，調査対象のソフトウェアで再利用している特定のライブラリの脆弱性が解消されたソフトウェアのバージョンが公開されるまでの期間である．

5.2 調査結果

ソフトウェアとライブラリの組ごとのそれぞれの脆弱性残留期間は表 4，5 のようになった．調査の結果，全体の平均脆弱性残留期間は 304 日であった．Android では，脆弱性残留期間の最小値から，脆弱性が公開された日のうちに，脆弱性の解消が行われた場合があることが確認できる．また，中央値や平均値が全体に比べて低いことから，ある程度脆弱性情報の追跡を行っており，脆弱性対策が行われていることが確認できる．さらに，Android のライブラリごとの脆弱性残留期間を示す図 9 の平均値や最大値から，同じソフトウェアでも脆弱性の対応に差があることが確認できる．apitrace は zlib による同バージョンで発生・解消された脆弱性のみが調査対象に含まれていたため，残留期間が同じとなり，分散が現れなかった．fs2open は，最小値が全体の中央値を上回るため，脆弱性の対応が他と比べて遅いことが確認できる．また，分散も大きいため，脆弱性によって対応に差があることが確認できる．gecko-dev は，全ての数値が低く，十分に脆弱性対策が行われていることが確認できる．ライブラリごとでは，libpng の分散が大きいことから，同じライブラリの再利用であっても，ソフトウェアによる脆弱性対応における期間の違いが大きく現れていることが確認できる．

CVSS と脆弱性残留期間の相関係数は，ソフトウェア・ライブラリそれぞれに対して，表 6，7 のようになった．ここでは，相関係数が低いということが，深刻な脆弱性ほど脆弱性残留期間が短いということを意味する．この結果から，脆弱性の深刻度とその解消までの期間には相関関係がないことが確認できる．一部，curl や gecko-dev の相関係数のように，非常に弱い負の相関関係を示す場合があるケースも見られた．

これらの結果から，ソフトウェア・ライブラリごとに脆弱性残留期間が大きく異なり，脆弱性対策に差があることが確認できた．また，深刻な脆弱性であっても早期に対応されている訳ではないことが確認できた．

6. 妥当性への脅威

6.1 実験対象の妥当性

実験対象のソフトウェアは，利用ライブラリのバージョンを適切に記録しているものだけを使用している．そのため，評価実験の結果には，それらのソフトウェアの特徴が影響している可能性がある．しかし，対象のソフトウェアはそれぞれ異なるコミュニティで開発されているため，開

表 4 脆弱性残留期間 (ソフトウェア)

	Android	apitrace	fs2open	gecko-dev	全体
平均値	190	691	656	64	304
中央値	127	691	611	50	133
最大値	914	691	1128	194	1128
最小値	0	691	200	3	0
分散	42512	0	98359	4385	96480
標準偏差	206	0	314	66	311

表 5 脆弱性残留期間 (ライブラリ)

	libpng	zlib	curl	全体
平均値	371	412	133	304
中央値	200	412	69	133
最大値	1128	691	473	1128
最小値	3	133	0	0
分散	119993	77841	19040	96480
標準偏差	346	279	138	311

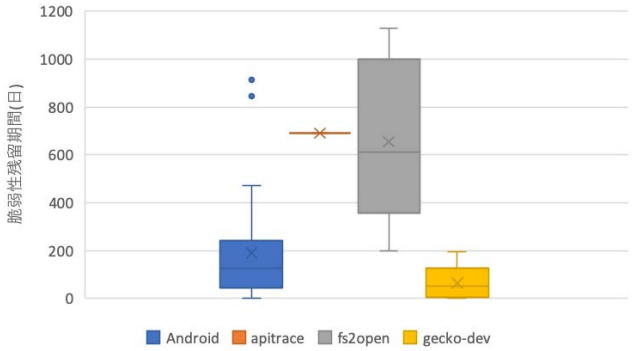

図 7 ソフトウェアごとの箱ひげ図

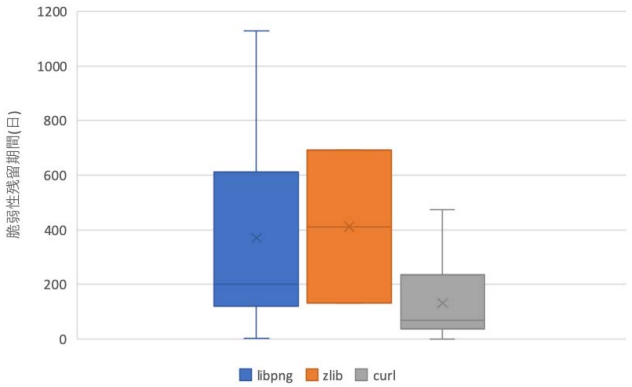

図 8 ライブラリごとの箱ひげ図

表 6 CVSS と脆弱性残留期間の相関係数 (ソフトウェア)

	Android	apitrace	fs2open	gecko-dev
相関係数	0.0825	N/A	0.0144	-0.283

表 7 CVSS と脆弱性残留期間の相関係数 (ライブラリ)

	libpng	zlib	curl
相関係数	0.156	0	-0.373

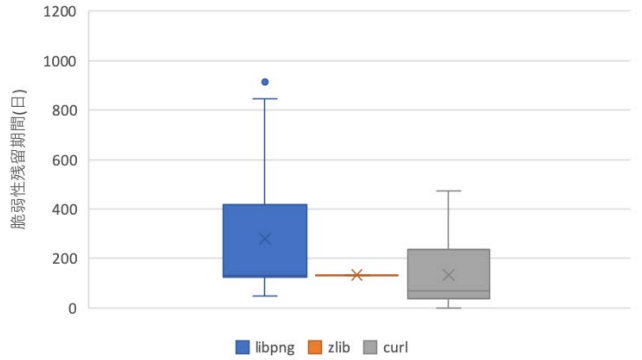

図 9 Android のライブラリごとの箱ひげ図

発者の再利用方法における偏りは少ないと考えられる．

6.2 検出手法の妥当性

本研究では，伊藤らが開発したライブラリのバージョンの検出手法を用いて脆弱性の検出を行っている．伊藤らの実験では，99.3%の正確さでバージョンの検出が行われているため，同一のデータセットを使用した今回の評価実験においての影響は少ないが，他の対象においてはバージョンの検出誤りによって出力される脆弱性が不正確のものとなる可能性がある．

7. まとめ

本研究では分析対象のソフトウェアとそこで再利用しているライブラリを比較し，脆弱性情報と再利用ライブラリ内の脆弱性情報を出力するツールを開発した．ライブラリのバージョン検出には，分析対象のソフトウェアと再利用したライブラリのバージョン管理システムのリポジトリの内容を比較し，バージョン検出を行う既存手法を用いた．

評価実験では，提案手法において 0.995 の適合率と 1 の再現率で脆弱性が検出されていることを確認した．また，ツールを利用した調査では，脆弱性情報が公開されてから解消されるまでにデータセット内の平均で 304 日間かかったことを確認した．ソフトウェア・ライブラリごとに脆弱性残留期間は大きく異なり，同様のライブラリを再利用していても，脆弱性の対応を行うまでの期間に大きな差が見られることを確認した．

今後の課題として，脆弱性検出後の出力結果に修正不要なファイルの情報を追加することが挙げられる．採用している伊藤らの手法では，ファイルごとのバージョン検出も行っているため，この検出されたファイルごとのバージョンが最新バージョンとして出力された場合，そのファイルは脆弱性の修正とは無関係であり，更新作業が不要なファイルであると判断できる．また，本ツールの有用性や扱いやすさをより正確に評価するために，被験者実験や他ツールとの比較などの実験を行うことも，今後の課題である．

謝辞 本研究は JSPS 科研費 JP18H04094，JP19K20239 の助成を受けた．

参考文献

[1] Mohagheghi, P., Conradi, R., Killi, O. M. and Schwarz, H.: An empirical study of software reuse vs. defect-density and stability, *Proceedings of the 26th International Conference on Software Engineering*, pp. 282–292 (2004).

[2] 田島浩一，岸場清悟，近堂 徹，渡邉英伸，岩田則和，西村浩二，相原玲二：脆弱性診断と脆弱性情報公開サイトを用いた脆弱性更新通知機能の試作，マルチメディア，分散，協調とモバイル (DICOMO2017) シンポジウム (2017).

[3] Chen, Y., Santosa, A. E., Yi, A. M., Sharma, A., Sharma, A. and Lo, D.: A Machine Learning Approach for Vulnerability Curation, *Proceedings of the 17th international Conference on Mining Software Repositories* (2020).

[4] 伊藤 薫，石尾 隆，神田哲也，井上克郎：軽量な類似度計算によるプロジェクト間のソースファイル集合の再利用検出，電子情報通信学会論文誌，Vol. J103-D NO.7, pp. 542–554 (2020).

[5] Xia, P., Matsushita, M., Yoshida, N. and Inoue, K.: Studying reuse of out-dated third-party code in open source projects, *Computer Software*, Vol. 30, No. 4, pp. 98–104 (2013).

[6] Li, P. and König, C.: b-bit minwise hashing, *Proceedings of the 19th International Conference on World Wide Web*, pp. 671–680 (2010).

[7] GMO ペイメントゲートウェイ株式会社：不正アクセスに関するご報告と情報流出のお詫び，`https://www.gmo.jp/news/article/5616/`. 2021 年 7 月 20 日閲覧.

[8] Equifax: Equifax Announces Cybersecurity Incident Involving Consumer Information, `https://www.equifaxsecurity2017.com/updates//-/announcements/a-progress-update-for-consumers`. 2021 年 7 月 20 日閲覧 (2017).

[9] MITRE Corporation: CVE - Common Vulnerabilities and Exposures (CVE), `https://cve.mitre.org/`. 2021 年 7 月 20 日閲覧.

[10] MITRE Corporation: Common Weakness Enumeration: CWE, `https://cwe.mitre.org/`. 2021 年 7 月 20 日閲覧.

[11] Forum of Incident Response and Security Teams: Common Vulnerability Scoring System v3.1: Specification Document, `https://www.first.org/cvss/v3.1/specification-document`. 2021 年 7 月 20 日閲覧.

[12] Dependabot: Dependabot, `https://dependabot.com/`. 2021 年 7 月 20 日閲覧.

[13] Balzarotti, D., Cova, M., Felmetsger, V., Jovanovic, N., Kirda, E., Kruegel, C. and Vigna, G.: Saner: Composing Static and Dynamic Analysis to Validate Sanitization in Web Applications, *Proceedings of the 2008 IEEE Symposium on Security and Privacy*, pp. 387–401 (2008).

[14] Wang, T., Wei, T., Gu, G. and Zou, W.: TaintScope: A Checksum-Aware Directed Fuzzing Tool for Automatic Software Vulnerability Detection, *Proceedings of the 2010 IEEE Symposium on Security and Privacy* (2010).

[15] Cheikes, B. A., Waltermire, D. and Scarfone, K.: Common Platform Enumeration: Naming Specification Version 2.3, (online), DOI: 10.6028/nist.ir.7695 (2011).

[16] 一般社団法人 JPCERT コーディネーションセンター, 独立行政法人情報処理推進機構：Japan Vulnerability Notes, `https://jvn.jp/`. 2021 年 7 月 20 日閲覧.