特別研究報告

題目

ライブラリのバージョン検出手法を用いた
脆弱性検知ツールの試作

指導教員
井上 克郎 教授

報告者
杉森 遼

令和 3 年 2 月 9 日

大阪大学 基礎工学部 情報科学科

令和 2 年度 特別研究報告

ライブラリのバージョン検出手法を用いた脆弱性検知ツールの試作

杉森 遼

内容梗概

ソフトウェア開発において，機能の一部を実現するために，オープンソースソフトウェアのソースコードを再利用することが一般的に行われている．ソフトウェアの再利用は，独自に開発した場合に比べて品質を向上させるが，再利用したライブラリに脆弱性が含まれている可能性がある．近年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ため，ソースコードを再利用する際にはメンテナンスが重要となる．

脆弱性検出ツールは多数存在しており，その中の 1 つに Dependabot が挙げられる．これは GitHub のマーケットプレイスから無料で導入できる脆弱性検出通知機能である．ソフトウェア内にあるパッケージマネージャが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・通知を行うのが主な機能である．しかし，パッケージマネージャによるライブラリの依存関係の記録を行ってい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．また，このツールの有用性を評価するために，評価実験と調査を行った．

評価実験では，開発したツールから出力された脆弱性情報と収集した脆弱性情報との一致する割合を求め，精度を計測した．その結果，0.995 の適合率と 1 の再現率という高い精度で脆弱性が検出できていることを確認した．また，提案ツールの実行時性能は，全体の実行時間で平均 153,276ms となることを確認した．

さらに，本研究で開発したツールを用いて，脆弱性情報が公開されてから解消されるまでの期間の調査を行った．その結果，脆弱性情報が公開されてから解消されるまでに平均で 304 日間，中央値 133 日間，最大で 1128 日間，最小で 0 日間かかっていることを確認した．また，同様のライブラリを再利用していても，脆弱性が解消されるまでの期間に大きな差があることを確認した．さらに，脆弱性の深刻度を示す CVSS とは一部非常に弱い相関を持つ場合があるが，基本的に相関が見られないことを確認した．このように，脆弱性検出における高い精度とこうした調査にツールが応用できることから，ツールの有用性を確認した．

主な用語

オープンソースソフトウェア
再利用分析
脆弱性検出

目 次

1 まえがき **5**

2 背景 **7**

2.1 ライブラリの再利用 7

2.2 脆弱性 8

2.2.1 CVE 8

2.2.2 CWE 9

2.2.3 CVSS 9

2.3 脆弱性検出ツール 9

3 提案手法 **11**

3.1 ライブラリバージョン検出手法 11

3.1.1 b-bit MinHash 法を用いた類似度計算 12

3.2 脆弱性情報の検索 13

3.2.1 バージョン情報に特定の文字列を含まないときのフィルタリング . . . 14

3.2.2 バージョン情報に特定の文字列を含むときのフィルタリング 15

4 ツールの実装 **16**

4.1 ツールの実行手順 16

4.1.1 ツールの実行例 17

5 評価実験 **20**

5.1 実験対象 20

5.2 実験方法 20

5.3 評価方法 21

5.4 結果 21

5.5 提案ツールの実行時性能 22

6 提案手法を用いた調査 **24**

6.1 調査方法 24

6.2 調査結果 24

7 妥当性への脅威 **29**

7.1 実験対象の妥当性 29

7.2 検出手法の妥当性 29

8　まとめ **30**

謝辞 **31**

参考文献 **32**

1　まえがき

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 の再利用を行うことにより，独自に開発したものに比べて，信頼性や安定性の向上，開発コストが削減できるためである [13]．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 [14]．例えば，2017 年には Java で開発されたライブラリである Apache Struts2 の脆弱性により，最大で 1 億 4300 万人の個人情報が公開された [7]．また，脆弱性が発見された日から，解消するための対処方法が確立されるまでに攻撃するゼロデイ攻撃も行われており，脆弱性に対する早期の対応が求められている．著名なゼロデイ攻撃として，2010 年に公表された複数の企業から情報を盗むことを目的とした攻撃である Aurora が挙げられる [2]．これらの事例や脅威があるため，開発者は開発しているソフトウェアに対して脆弱性の検出を行い，脆弱性が含まれているかどうかを確認する必要がある．そして，脆弱性が含まれている場合はメンテナンスを行い，脆弱性を解消する必要がある．

脆弱性に起因した危険性は脆弱性を持つ限り継続するものである．そのため，OSS のライブラリに限らず，ソースコードの脆弱性の検出についての研究が多く行われている．脆弱性更新通知機能を試作した研究では，公開されている脆弱性情報を基に，脆弱性の検出を行う [19]．しかし，脆弱性が解消されたライブラリのバージョンは通知に含まれないため，利用者は改めてどのバージョンにアップデートすべきかを調べる必要がある．また，近年では機械学習を用いての脆弱性検出 [4] が行われているが，脆弱性検出を行うための学習コストや実行時間について考慮されていないことが問題点として挙げられる．このように，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．

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 [18]．これは，分析対象のソフトウェアのソースファイル集合とライブラリのバージョンごとのソースファイル集合との間の類似度を比較することで，分析対象のソフトウェアが再利用しているライブラリのバージョンを検出する手法である．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．バージョン検出が完了すると，脆弱性情報を集めるためにライブラリ名を用いて脆弱性情報の検索を行う．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 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の出力を行う．

評価実験では，提案ツールの妥当性を評価するために，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，精度を計測する．また，提案ツールの実行時性能を評価するため，提案ツールの実行時間を計測する．さらに，提案ツールの有用性を示すための応用例として，評価実験で用いたソフトウェアの各バージョンごとの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う．

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 章では本研究で提案する手法について述べ，4 章では本研究で開発したツールについて説明する．5 章では本研究の評価実験について詳述し．6 章では提案手法を用いた調査について説明する．7 章では妥当性への脅威について述べ，8 章ではまとめと今後の課題を記述する．

2　背景

現在，パーソナルコンピュータの普及や性能向上により，ソフトウェア開発への参入障壁は低くなっている．また，スマートフォンの普及により，開発されるソフトウェアが増加し，普及する速度も上昇している．しかし，流行り廃りのサイクル速度の上昇により，開発者はより素早いソフトウェア開発を求められている．そのため，開発するソフトウェアの一部の機能を再利用可能な OSS を用いて実装することがある．

また，ソフトウェアの普及拡大に伴い，ソフトウェアが収集・蓄積した情報が狙われるようになった．開発者アカウントや顧客情報へのアクセス権限を持つアカウントを狙い，考えられる全ての組み合わせを試すブルートフォースアタックやデータベースシステムを不正に操作する SQL インジェクションなどが攻撃手段として用いられる．さらに，ソフトウェアが持つ脆弱性を攻撃者が利用し，個人情報を漏洩させたという事例が存在する [7].

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のソースコード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 [13]．Slyngstad らによる調査では，ソフトウェアの再利用におけるメリットは，開発時間の短縮，再利用可能なコンポーネントと標準化されたアーキテクチャの品質の良さであり，再利用と再利用によるリワークの増加は関係がないということが報告されている [15].

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報告されている [17]．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されている [17]．さらに，開発者は再利用するために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用いることがある．これにより，ファイルやディレクトリの完

全一致によってバージョンを判断することができない．そのため，伊藤らはライブラリから再利用しているディレクトリとその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としたライブラリのバージョン検出手法を提案した [18].

2.2 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．この脆弱性により，米国の消費者信用情報会社 Equifax が持つ最大 1 億 4300 万人の顧客情報がハッカーにより公開された [7]．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．しかし，同社は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．また，同様の脆弱性により，GMO ペイメントゲートウェイ株式会社が運営受託している東京都税クレジットカード支払いサイトおよび独立行政法人住宅金融支援機構の団体信用生命保険特約料のクレジットカード支払いサイトが不正アクセスされた．それにより，利用者のクレジットカード番号・有効期限など合計 72 万件近くの情報が流出した可能性があると発表している [8]．このように，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，脆弱性検出ツールを用いての検出やメンテナンスが必要であり，脆弱箇所を特定することは有用である．

脆弱性対策を行うにあたって，公開脆弱性情報などの情報を収集することが重要であり，脆弱性情報を集める上で必要となる用語や指標が存在する．一例として，以下のようなものが挙げられる．

- CVE (Common Vulnerabilities and Exposures)
- CWE (Common Weakness Enumeration)
- CVSS (Common Vulnerability Scoring System)

以下それぞれの用語・指標について説明する．

2.2.1 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 [21]．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．

2.2.2 CWE

CWE とは，共通脆弱性タイプ一覧のことであり，ソフトウェアの脆弱性の種類を判別するためのものである [20]．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．CWE の利点として，以下が挙げられる．

- ソフトウェアの脆弱性について，共通言語で議論することができる
- セキュリティツールの標準の評価尺度として使用できる
- 脆弱性の特定，軽減，防止のための共通基準として使用できる

2.2.3 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 [22]．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．

このように，CVSS は情報システムの脆弱性に対する汎用的な評価手法であり，ベンダーに依存しない共通の評価方法を提供している．また，脆弱性の深刻度を同一基準の下で定量的に比較でき，共通言語で議論可能なため，有用である．

2.3 脆弱性検出ツール

2.2 章で紹介した事例から，脆弱性の危険性とその被害は明らかである．よって，脆弱性を利用した攻撃が行われる前に，脆弱性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 [6]．GitHub のマーケットプレイスからインストールし，Dependabot を使用したいリポジトリを選択すると使用できるため，導入するのが容易である．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という問題点が存在する．

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 [1]．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 [4]．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 [16]．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　提案手法

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．本章では，ライブラリのバージョン検出に用いた手法や脆弱性情報の検索について説明する．検索に用いるフィルタリングについて，具体例を用いて示す．

3.1　ライブラリバージョン検出手法

本研究では，ソフトウェアが再利用しているライブラリのバージョン特定のために伊藤らの軽量な類似度計算による再利用検出手法を用いる [18]．使用手法では，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のファイルと類似度計算を行い，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．

ここで，バージョンの特定を行うにあたり，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．入力ファイルの 1 つ q に対して，バージョン v のなかで最も類似しているファイルの類似度は，以下の式で定義されている．ただし，ファイル間の類似度はある閾値 θ 以上の値の場合のみを記録し，θ 未満の場合は 0 としている．

$$
\begin{aligned}
S(q,v) &= \begin{cases} S_{\max}(q,v), & \text{if } S_{\max}(q,v) \geq \theta \\ 0, & \text{otherwise} \end{cases} \\
S_{\max}(q,v) &= \max_{f \in \mathrm{Files}(v)} \mathrm{sim}(q,f)
\end{aligned}
$$

このとき，$S_Q(v)$ の定義は以下の通りである．

$$
S_Q(v) = \sum_{q \in D_q} S(q,v)
$$

つまり，$S_Q(v)$ は $S(q,v)$ を合計することで，ソフトウェア内で再利用されているライブラリがバージョン v に由来するものかを表す．

3.1.1 b-bit MinHash 法を用いた類似度計算

ファイル間の類似度を計算する際，高速に計算することが重要である．Kawamitsu らは，類似度として，ファイル同士の最長一致部分列の長さを使用しているが，その計算には両方のファイルの長さの積に比例した時間が必要となるため，合計数千万行のリポジトリの分析に最大で 4 時間程度かかると報告している [11]．

ソースファイルの集合の類似度を求めるにあたり，Jaccard 係数 [9] が使用されている．Jaccard 係数とは集合間の類似度を計測する指標であり，以下の式で表される．

$$\mathrm{sim}(S_1, S_2) = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|}$$

このときの S_1, S_2 はそれぞれのファイルの内容を表現した集合としている．しかし，ソースファイル同士を直接比較するだけでは，ソースファイル単位での検出を行う場合と違いがない．そのため，上述の類似度の計算を高速で求める手法が必要となる．その手法として，MinHash 法 [3] が提案されている．

MinHash 法は，任意のハッシュ関数 h_i が与えられたとき，集合 S 中の各値をハッシュ関数に代入し，ハッシュ値が最小となる要素を取り出す関数 m_i は以下のように表される．

$$m_i(S) = \min_{s \in S}(h_i(S))$$

2 つの集合 S_1, S_2 に対して，$m_i(S_1), m_i(S_2)$ を計算すると，$m_i(S_1)$ と $m_i(S_2)$ が一致する確率は，Jaccard 係数と一致する．それらは複数のハッシュ関数を用意すれば，これらのハッシュ値の一致する割合から Jaccard 係数の推定値を求めることができる．

MinHash 法で用いられるハッシュ値は，偶然衝突する確率を低くするために 64 ビットの整数を用いることが多い．そのため，MinHash 法を適用すると 1 つの文書に対して，64 ビットの整数 $\times\, k$ 個の容量が必要となる．これは元の文書のサイズに関わらず一定であるため，保存に必要な容量が元より大幅に増える場合がある．また，ハッシュ関数の個数 k は誤差を軽減するために数百個必要であるため，大きな計算量が生じる．

そこで，MinHash 法を省メモリで実現する方法として b-bit MinHash 法 [12] が提案されている．この手法では，MinHash 法で計算したハッシュ値 $m_i(S)$ の下位 b ビットのみをハッシュ値として用いる．例えば $b = 1$ のとき，b-bit MinHash 法のハッシュ関数 $b_i(f)$ は以下の式で表される．

$$b_i(S) = \mathrm{LSB}(m_i(S))$$

ここでの LSB は Least Significant Bit を表す.

論文中では，計算量が最も少なくなるのは $b = 1$ のときであると述べられている．このとき，k 個のハッシュ関数を用いても k ビットの記憶域で済む．$P(S_1, S_2)$ の近似値として，k 個のハッシュ値の一致割合 $P_o(S_1, S_2)$ を用いて，集合 S_1, S_2 間の Jaccard 係数の推定値 $\mathrm{sim}(S_1, S_2)$ を計算することが可能である．

$$\begin{aligned} \mathrm{sim}(S_1, S_2) &= \left(P_o(S_1, S_2) - \frac{1}{2} \right) \times 2 \\ P_o(S_1, S_2) &= 1 - \frac{1}{k} \sum_{i=1}^{k} \mathrm{XOR}(b_i(S_1), b_i(S_2)) \end{aligned}$$

ここで，$\mathrm{sim}(x, y)$ はファイル集合 x, y 間の類似度であり，値域は [0, 1] である．XOR 演算を用いるため，$\mathrm{sim}(S_1, S_2)$ の計算量は $O(1)$ となり，ファイル間の比較を高速に行うことができる．伊藤らの手法はこの b-bit MinHash 法を用いることで，軽量な再利用検出を実現している．

3.2 脆弱性情報の検索

本研究では，3.1 節で説明した伊藤らの手法で検出したバージョン情報を用いて脆弱性情報の検索を行う．脆弱性情報の検索には，CVE-Search[5] という OSS を利用する．

CVE-Search とは，CVE を MongoDB というデータベースにインポートし，CVE の検索を容易にするツールである．このツールに入力を与えると，キーワード検索が行われ，そのキーワードに関連する脆弱性情報が出力される．今回の手法では，CVE-Search の WebAPI インターフェースを使用し，脆弱性情報の検索を行う．

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．関係するバージョン情報は膨大であり，入力には 1 単語しか使用できないため，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．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脆弱性が関連する製品やソフトウェア，関連するバージョン情報を用いる．ここには，CPE(Common Platform Enumeration) というソフトウェアなどを識別す

```
{
  "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

```
cpe:{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}
```

図 2: CPE の基本構成

るための名称を用いて，関連製品やバージョン情報が記載されている．基本構成は図 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．このように，関連製品情報は一定の規則に従って記載されているため，バージョン情報に特定の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．特定の文字列とは，CPE における「アップデート」を示す文字列を指す．図 3 における beta が特定の文字列に当てはまる．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情報に含まれる文字列 beta と rc を特定の文字列として扱う．

3.2.1 バージョン情報に特定の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定の文字列を含まない場合，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」として扱う．そして，以下の文字列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．

```
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

- :libpng:1.2.38:

3.2.2 バージョン情報に特定の文字列を含むときのフィルタリング

特定の文字列を含む場合，フィルタリングには複数の文字列を用いる．これは，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である．そのため，3 種類の文字列を用いてそのいずれかを含む CPE を持つ場合に，出力する脆弱性情報に含める．

- ライブラリ名+バージョン情報のみ
- ライブラリ名+バージョン情報+アップデート名
- ライブラリ名+バージョン情報+アップデート名 (バージョン番号除く)

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:*

4　ツールの実装

本研究では，提案手法を実装したツールを試作する．ツールを図解すると，図 4 となる．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. 検出脆弱性情報と最新バージョンにおける修正情報を出力

開発するツールは，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，脆弱性情報や関連ファイル，ファイル差分，最新バージョン情報を出力する．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．手順 3 では，再利用したライブラリ名を用いて，脆弱性情報を検索する．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．ソースファイル群と再利用元のライブラリの指定は，起動時に引数として与える．ここでのバージョンとは，伊藤らの手法と同様に，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．

4.1　ツールの実行手順

今回の研究で試作したツールの実行手順について述べる．ツールの引数として以下の要素を与える．ビット数とハッシュ関数の個数は伊藤らの手法に必要な引数である．

- バージョン検出するライブラリを持つディレクトリのパス
- ライブラリのパス
- ビット数
- ハッシュ関数の個数

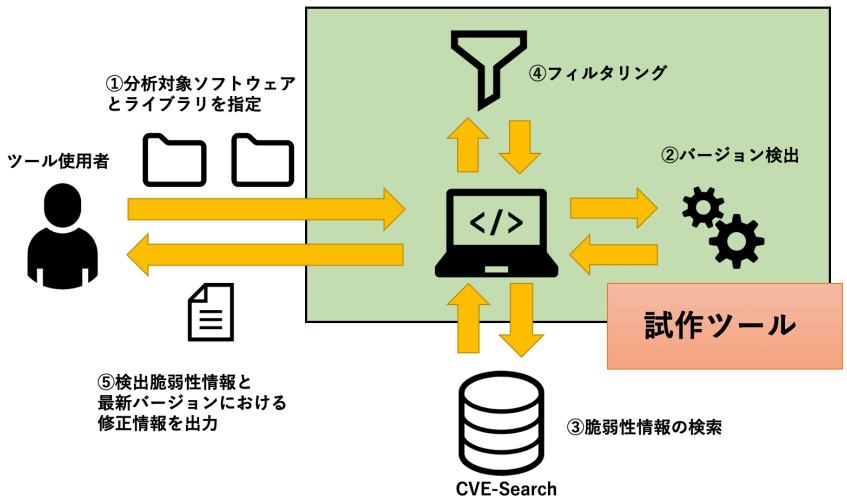

図 4: 試作ツールの概観

- 出力ディレクトリ

4.1.1 ツールの実行例

試作したツールを用いた実行例を挙げる．このとき，b-bit MinHash 法のビット数とハッシュ関数の個数は，伊藤らの手法と同様に $b = 1$，$k = 2048$ とした．

まず，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．このプログラムに対して，以下のような引数を与え，実行する．

- /path/to/android-curl
- /path/to/curl
- 1
- 2048
- /path/to/outputDirectory

本実行例で使用している Android のバージョンは， android-7.1.1_r59 である．上述の引数を与えて実行を行うと図 5 のようになる．実行すると，図 5 の 1 のように，再利用しているライブラリのバージョン情報とそのライブラリの最新バージョンにおける脆弱性の修正情報を出力する．これらにより，最新バージョンでどれだけ脆弱性が解消されているかを視覚化することができる．また，脆弱性修正情報を提示することで，最新バージョンへの更新を促している．その後，図 5 の 2 のように，バージョン間でのファイル間の違いや追加，削除

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

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)
```

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 5: ツールの実行例

についての情報を出力する．これにより，更新が必要となるファイルの見落としを減らす．最後に，図 5 の 3 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の出力を行う．脆弱性一覧には，以下の項目を出力する．これらの情報を出力し，どのような脆弱性が含まれているのかを視覚化し，更新作業を支援する．

- CVE
- 脆弱性情報の要約
- CVSS
- CWE

- 脆弱性の種類名
- 脆弱性の種類の説明

5　評価実験

提案手法の有用性を評価するため，ライブラリのバージョン情報を元に検出された脆弱性情報の正確さを計測する．提案手法による出力結果と手作業で収集した正解集合を用いて精度を計測する．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．

5.1　実験対象

実験対象として，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する．表 1 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．実験対象のソフトウェアは，伊藤らの実験で使用されたソフトウェアと同じものを使用している．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，対象を選択されているためである．さらに，既存手法で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る．そのため，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ためである．ソフトウェア及びライブラリのバージョン数は，表 2 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 で管理されており，表中のリポジトリ URL からクローンした．

5.2　実験方法

本ツールの入力に表 1 のソフトウェア・ライブラリの組を与え，実験を行う．伊藤の手法では，ファイルごとの類似度の合計が最も大きいバージョンを検出したバージョンとして出力する．そのため，類似度の合計値が一致することで，複数のバージョンが検出される場合がある．検出された場合，その中でバージョンのリリース日が最も新しいバージョン情報を使用する．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

表 1: 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 2: 実験対象のソフトウェア・ライブラリを格納しているリポジトリ

リポジトリ名	リポジトリ URL	バージョン数
android(libpng)	https://android.googlesource.com/platform/external/libpng	76
android(curl)	https://android.googlesource.com/platform/external/curl	61
android(ogg)	https://android.googlesource.com/platform/external/ogg	31
android(zlib)	https://android.googlesource.com/platform/external/zlib	69
apitrace	https://github.com/apitrace/apitrace.git	12
fs2open	https://github.com/scp-fs2open/fs2open.github.com.git	797
gecko-dev	https://github.com/mozilla/gecko-dev.git	98
v8monkey	https://github.com/zpao/v8monkey.git	72
libpng	git://git.code.sf.net/p/libpng/code	1615
curl	https://github.com/curl/curl.git	200
ogg	https://github.com/xiph/ogg.git	12
zlib	https://github.com/madler/zlib.git	72

るライブラリのバージョンの脆弱性情報に対して，検出結果が用意した正解集合と一致した割合によって求める．用意した正解集合は，JPCERT/CC と IPA の共同運営している脆弱性対策情報ポータルサイト JVN [10] から該当するライブラリの脆弱性を手作業で収集したものである．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から，関連のある脆弱性情報のみを正解集合とした．

5.3 評価方法

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

$$\text{Precision} = \frac{|\text{Result} \cap \text{Answer}|}{|\text{Result}|}$$

$$\text{Recall} = \frac{|\text{Result} \cap \text{Answer}|}{|\text{Answer}|}$$

5.4 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．脆弱性検出バージョン数がソフトウェアのバージョン数と異なる値であるが，これは脆弱性が検出されなかったためである．検出結果と収集した脆弱性情報は，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．

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
android(libpng)	libpng	76	76	385	385	1	1
android(curl)	curl	61	46	739	739	1	1
android(ogg)	ogg	31	0	0	0	N/A	N/A
android(zlib)	zlib	69	23	92	92	1	1
apitrace	libpng	12	12	24	24	1	1
	zlib	12	8	32	32	1	1
fs2open	libpng	797	711	1,605	1,577	0.983	1
	zlib	797	146	584	584	1	1
gecko-dev	libpng	98	83	678	678	1	1
	ogg	98	0	0	0	N/A	N/A
	zlib	98	0	0	0	N/A	N/A
v8monkey	libpng	72	72	1,440	1,440	1	1
	zlib	72	0	0	0	N/A	N/A
全体		2,293	1,177	5,579	5,551	0.995	1

脆弱性検出において，Recall の低さは脆弱性を見逃していることになるため，脆弱性の見逃がしがなかったことを意味する．Precision が 1 とならなかった fs2open の libpng 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で見ら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であった．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．これらのことから，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．

5.5 提案ツールの実行時性能

提案手法の実行時性能の評価するために，脆弱性検出にかかる実行時間を測定する．このとき，ライブラリのバージョン検出に要する時間は，伊藤らの計測値を参考値として記載する．そのため，本実験で実際に計測する時間は，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である．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 4 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

表 4: 提案ツールの実行時間

ソフトウェア	ライブラリ名	ライブラリのバージョン検出 [18](ms)	脆弱性検出 (ms)	合計 (ms)	検出脆弱性数の平均
android	libpng	242,371	47,818	267,189	5.07
	curl	95,864	100,940	196,784	12.11
	ogg	981	214,769	215,750	0
	zlib	7,250	38,265	45,415	1.33
apitrace	libpng	214,554	30,309	244,863	2.00
	zlib	6,156	32,726	38,882	4.00
fs2open	libpng	279,272	28,646	307,918	2.01
	zlib	9,045	33,993	43,038	0.73
gecko-dev	libpng	235,119	71,167	306,286	6.92
	ogg	9,988	208,845	218,833	0
	zlib	17,526	29,958	47,484	0
v8monkey	libpng	23,638	139,968	163,606	20.00
	zlib	12,141	29,815	41,956	0
平均		104,761	48,515	153,276	2.43

214,769ms，最小で 28,646ms の時間がかかった．ogg 以外のライブラリは，脆弱性が検出されるほど，実行時間が増加することが確認できる．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．参考値となるライブラリのバージョン検出時間の平均値と検出後の実行時間の平均値の合計は 153,276ms となり，実用的な時間で実行可能である．そのため，提案ツールは有用であると言える．

6　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．また，脆弱性残留期間が脆弱性の深刻度を示す CVSS と相関があるのかについて調査する．調査対象は，5.1 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．

6.1　調査方法

本研究で試作したツールを用いて，各脆弱性情報が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．ソフトウェアのバージョンが公開された日付は Git に記録されているため，脆弱性情報の公開日から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を脆弱性残留期間とする．これをソフトウェアごと・ライブラリごとの脆弱性残留期間を調べる．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．CVSS と脆弱性残留期間の相関についての調査には，相関係数を用いる．

6.2　調査結果

ソフトウェア・ライブラリごとのそれぞれの脆弱性残留期間は表 5，6 のようになった．また，脆弱性残留期間の分布はそれぞれ図 6，7 のようになった．図 9 から図 15 は各ソフトウェア・ライブラリの脆弱性残留期間の散布図である．調査の結果，全体の平均脆弱性残留期間は 304 日で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．Android では，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最小値から確認できる．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 8 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．fs2open は，最小値が全体の中央値を上回るため，脆弱性の対応が遅いことが確認できる．また，分散も大きいため，脆弱性の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．さらに，libpng の分散が大きいことから，同様のライブラ

表 5: 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 6: 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

リが再利用されていても，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．

また，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 7，8 のようになった．この結果から，基本的には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．一方で，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．負の相関は深刻な脆弱性ほど，脆弱性残留期間が短いということを意味する．そのため，相関が見られない，もしくは非常に弱い負の相関関係しか見られないことから，深刻な脆弱性ほど早期に対応されることもあるが，そうした傾向が低いことが確認できる．

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．Equifax 社の事例 [7] のように，脆弱性を長期間放置していた結果，個人情報の漏洩を招くことがある．そのため，脆弱性検出と発見時の早期対応を徹底する必要があると言える．また，このツールを用いることで，こうしたソフトウェア・ライブラリごとの脆弱性対策に関する調査を行えることが確認できた．

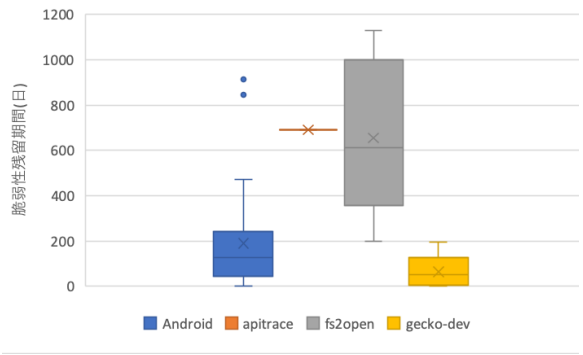

図 6: ソフトウェアごとの箱ひげ図

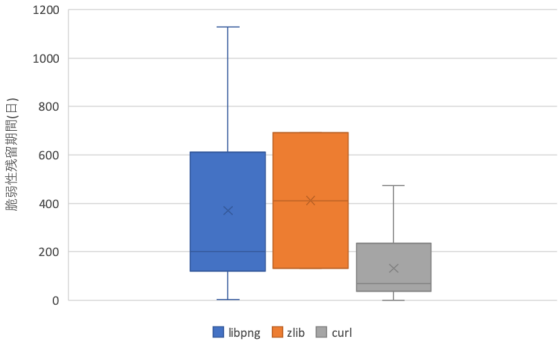

図 7: ライブラリごとの箱ひげ図

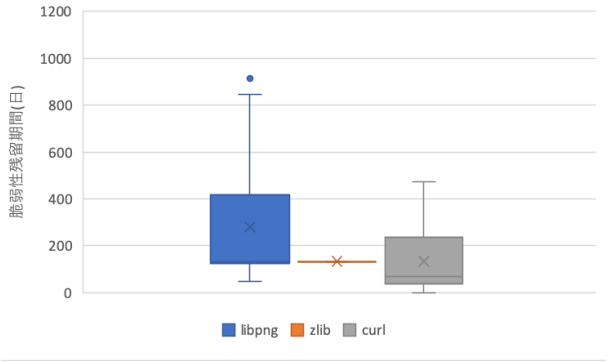

図 8: Android のライブラリごとの箱ひげ図

表 7: 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 8: 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

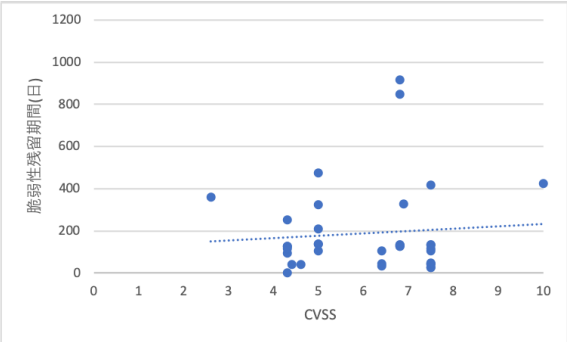

図 9: Android の脆弱性散布図

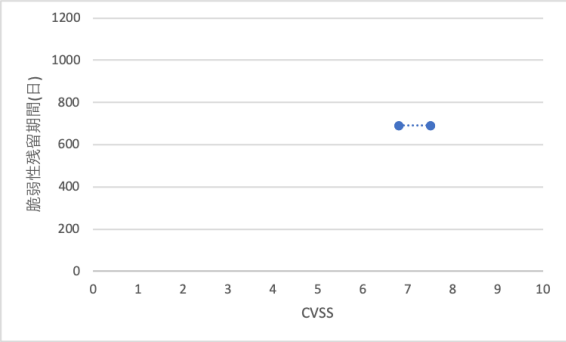

図 10: apitrace の脆弱性散布図

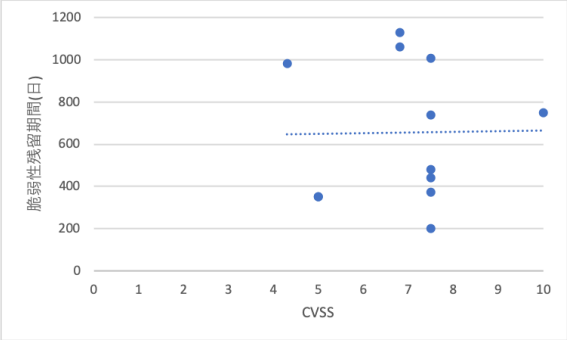

図 11: fs2open の脆弱性散布図

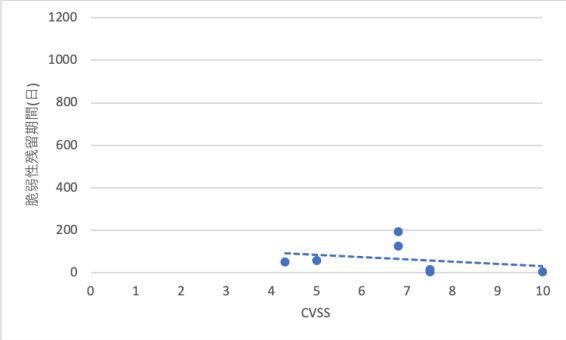

図 12: gecko-dev の脆弱性散布図

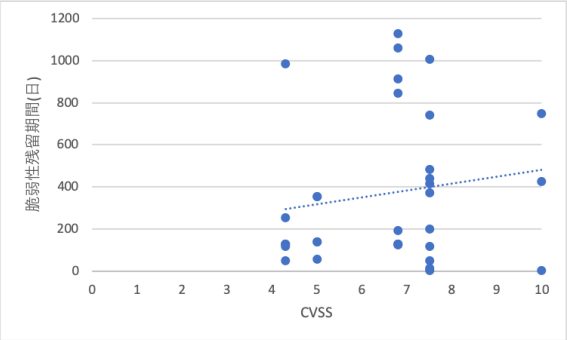

図 13: libpng の脆弱性散布図

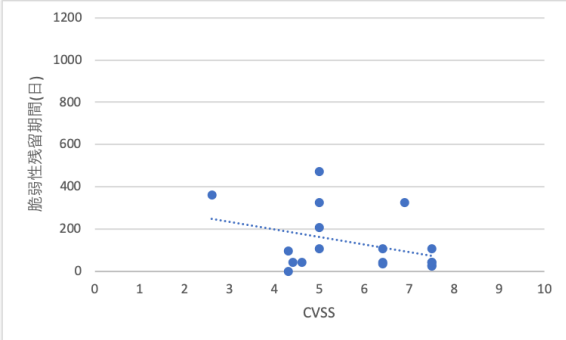

図 14: curl の脆弱性散布図

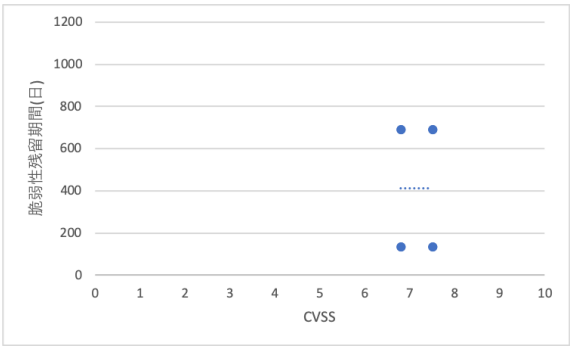

図 15: zlib の脆弱性散布図

7　妥当性への脅威

7.1　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，開発者の再利用方法における偏りは少ないと考えられる．

7.2　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．そのため，検出されるバージョン情報は伊藤らの手法に依存している．論文中の実験において，99.3%の正確さでバージョンの検出が行われている．そのため，本論文の評価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違い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．

8　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．そして，提案ツールの妥当性とツールの有用性を示すために，評価と調査を行った．

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，提案ツールの実行時間は平均 153,276ms であることを確認した．ツールを利用した調査では，全体で平均で 304 日間，中央値 133 日間，最大で 1128 日間，最小で 0 日間，脆弱性情報が公開されてから解消されるまでにかかったことを確認した．また，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．さらに，CVSS とは一部非常に弱い負の相関を示すことがあるが，基本的には相関を示さないことを確認した．これにより，深刻な脆弱性であっても早期に対処されていないことが明らかとなった．これらのように，評価実験における高い精度とこうした調査にツールが応用できることから，ツールの有用性を確認した．

今後の課題として，機能面の更なる充実や，ツールの導入の容易性の向上が挙げられる．追加する機能の例として，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．この出力を行うことで，脆弱性箇所の絞り込みが可能となる．採用している手法では，ファイルごとのバージョン情報の検出も行っている．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，そのファイルはバージョンが上がる際に編集されていないことが明らかとなる．そのため，更新作業が不要なファイルであると判断できる．導入の容易性については，脆弱性情報を検索するために使用した CVE-Search を提案ツールと同時に導入できるようにすることで，より利便性を向上させられると考える．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

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井上克郎教授には，非常に多忙な中，研究に関する適切な御指導及び御助言を賜りました．井上教授の御指導及び御助言のおかげで本論文を完成させることができました．井上教授に心より深く感謝いたします．

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，研究室での発表の機会において多くの御意見・御助言を賜りました．松下准教授に心より感謝いたします．

大阪大学大学院情報科学研究科神田哲也助教には，研究室での発表や評価内容について，大変貴重な御意見を賜りました．多くの御助言を頂いた神田助教に心より深く感謝いたします．

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伊藤薫氏には，日々の研究や発表に関する相談に乗っていただき，また本論文の修正に御協力していただく など研究の様々な場面で御助力いただきました．心より深く感謝いたします．

最後に，その他様々な御指導，御助言等をいただい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井上研究室の皆様に心より深く感謝いたします．

参考文献

[1] D. Balzarotti, M. Cova, V. Felmetsger, N. Jovanovic, E. Kirda, C. Kruegel, and G. Vigna. Saner: Composing static and dynamic analysis to validate sanitization in web applications. In *Proceedings of the 2008 IEEE Symposium on Security and Privacy*, pp. 387–401, 2008.

[2] L. Bilge and T. A. Dumitraş. Before we knew it: an empirical study of zero-day attacks in the real world. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pp. 833–844, 2012.

[3] A. Broder. On the resemblance and containment of documents. pp. 21–29, 6 1997.

[4] Y. Chen, A. E. Santosa, A. M. Yi, A. Sharma, A. Sharma, and D. Lo. A machine learning approach for vulnerability curation. In *Proceedings of the 17th international Conference on Mining Software Repositories*, 2020.

[5] cve-search Team. cve-search. `https://github.com/cve-search/cve-search`.

[6] Dependabot. Dependabot. `https://dependabot.com/`.

[7] Equifax. Equifax announces cybersecurity incident involving consumer information, 10 2017. `https://www.equifaxsecurity2017.com/updates//-/announcements/a-progress-update-for-consumers`.

[8] 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-pg.com/corp/newsroom/pdf/170310_gmo_pg_ir_kaiji.pdf`.

[9] P. Jaccard. The distribution of the flora in the alpine zone.1. *New Phytologist*, Vol. 11, No. 2, pp. 37–50, 1912.

[10] 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/`.

[11] N. Kawamitsu, T. Ishio, T. Kanda, R. G. Kula, C. D. Roover, and K. Inoue. Identifying source code reuse across repositories using lcs-based source code similarity. pp. 305–314, 2014.

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

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

[14] NPO 日本ネットワークセキュリティ協会. 2018 年情報セキュリティインシデントに関する調査結果〜個人情報漏えい編〜(速報版). https://www.jnsa.org/result/incident/data/2018incident_survey_sokuhou.pdf.

[15] O. P. N. Slyngstad, A. Gupta, R. Conradi, and P. Mohagheghi. An empirical study of developers views on software reuse in statoil asa. In *Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering*, pp. 242–251, 2006.

[16] T. Wang, T. Wei, G. Gu, and W. Zou. Taintscope: A checksum-aware directed fuzzing tool for automatic software vulnerability detection. In *Proceedings of the 2010 IEEE Symposium on Security and Privacy*, 2010.

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

[18] 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.

[19] 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.

[20] 独立行政法人情報処理推進機構. 共通脆弱性タイプ一覧 CWE 概説. https://www.ipa.go.jp/security/vuln/CWE.html.

[21] 独立行政法人情報処理推進機構. 共通脆弱性識別子 CVE 概説. https://www.ipa.go.jp/security/vuln/CVE.html.

[22] 独立行政法人情報処理推進機構. 共通脆弱性評価システム CVSS 概説. https://www.ipa.go.jp/security/vuln/CVSS.html.