OSS利用者の意思決定を支援するソースコード開発におけるセキュリティリスクの定義と可視化

光澤 加偉[1,a)]　近藤 賢郎[2,5,b)]　ルーク コリー[3,c)]　甲斐 賢[4,d)]　手塚 悟[5,e)]

概要：昨今では，OSS に対して悪意のある第三者が不正に管理者権限を奪取し悪意のあるコードが挿入される事案や，OSS に残存する脆弱性に起因したセキュリティ事故の事案が発生している．このため，ソースコード開発過程のリスクを管理・可視化するために OpenSSF の S2C2F やセキュリティスコアカードといったフレームワークやツールが開発されている．しかし，S2SC2F はソースコード開発過程でリスクが生じる原因の整理が不十分であり，セキュリティスコアカードは OSS 開発者による協力を前提としている点で OSS の利用者向けのリスクの可視化が不十分である．本稿では，ソースコード開発過程に潜むリスクをそれが生じる原因の観点から定義し，OSS の利用者向けにこれらのリスクを可視化する eSBOM と意思決定を支援するためのツールである SCMV を提案・実装した．そして，2022 年 7 月時点の npm パッケージ 3000 個のデータセットを使用し，リスクの存在を SCMV を利用して検証した．これらの評価の結果、本稿で定義したリスクがソースコード開発過内の脅威を包括的にカバーされ，定義が有効である事を確認した．

キーワード：software supply chain, Software Bill of Materials, the Source Code Development Process

Definition and Visualization of Security Risks in OSS Development to Support User's Decision-Making

Kai Mitsuzawa[1,a)]　Takao Kondo[2,5,b)]　Korry Luke[3,c)]　Satoshi Kai[4,d)]　Satoru Tezuka[5,e)]

Abstract: Recent security incidents have highlighted the risks of malicious code in Open Source Software (OSS) through unauthorized access and vulnerabilities. Frameworks such as S2C2F and tools such as security scorecards have been developed to manage and visualize these risks. However, they have the following limitations: S2C2F does not fully address the root causes of risk in the Source Code development process, while Security Scorecards rely on the cooperation of OSS developers, limiting their effectiveness for users. This paper defines risks in the source code development process in terms of their causes, and proposes and implements eSBOM which visualizes these risks for OSS users, and SCMV which is a tool to support decision-making for OSS users. Using a dataset of 3,000 NPM packages from July 2022, this paper validates the existence of these risks and confirms that the proposed risk definitions are comprehensive and valid in covering threats in the Source Code development process.

Keywords: software supply chain, Software Bill of Materials, the Source Code Development Process

[1] 慶應義塾大学 環境情報学部
Faculty of Information and Environment Studies, Keio University
[2] 北海道大学 情報基盤センター
Information Initiative Center, Hokkaido University
[3] 慶應義塾大学 大学院政策・メディア研究科
Graduate School of Media and Governance, Keio University
[4] 慶應義塾大学 SFC 研究所
Keio Research Institute at SFC, Keio University
[5] 慶應義塾大学 グローバルリサーチインスティチュート
Global Research Institute, Keio University
[a)] kaizawa97@sfc.wide.ad.jp
[b)] latte@iic.hokudai.ac.jp
[c)] koluke@sfc.wide.ad.jp
[d)] skai@sfc.keio.ac.jp
[e)] tezuka@sfc.keio.ac.jp

© 2024 Information Processing Society of Japan

This paper is work in progress and not peer-reviewed.

1. はじめに

近年のソフトウェア開発では，Open Source Software (OSS) [1] が積極的に用いられており，これらの第三者が開発した OSS パッケージを用いずにソフトウェアを構築する事は困難である．OSS への依存が高まる中で，OSS のソースコードの開発過程で利用されるソースコードレポジトリの管理者権限が奪取された事に起因して悪意のあるコードが挿入される事や [2]，OSS の開発プロジェクトの停滞によって脆弱性の修正に時間を要する等 [3]，OSS によるセキュリティインシデントが発生している．ESLint [2] の事案は毎週 3000 万ダウンロードされて利活用する OSS に対して， OpenSSL [3] の事案は TLS に基づいた暗号化通信路を確立する為のライブラリとしてデファクトスタンダードとなっている OSS に対して生じたインシデントである．この為， OSS に関わるセキュリティリスクを定量的に把握してそれを可視化する事は，ソフトウェアの安全な利活用を推進する上で重要だ．

OSS は，ソースコード開発過程，ソフトウェアパッケージ開発過程，パッケージマネージャ等を用いた配布過程の 3 つの過程を経て，Consumer に利活用される．そこで，OSS の開発プロジェクト毎に，ソースコード開発から配布までの 3 つの過程の全てで，セキュリティリスクの定義と可視化が行われる事が望ましい．Open Source Security Foundation (OpenSSF) [4] では，OSS 開発者向けのセキュリティフレームワークである Secure Supply Chain Consumption Framework (S2C2F)[5] を提供し，OSS が Consumer に利活用されるまでのセキュリティリスク整理を行っている．また，近年ソフトウェアを動作させる為に必要なソフトウェアパッケージの依存関係等を構成したリストである Software Bill of Materials (SBOM) [6] を用いてソフトウェアに内在する脆弱性を可視化・管理する取り組みがあり，GitHub 等の OSS のソースコードレポジトリでもその取り組みを採用している．しかし，ソースコード開発過程に関しては，既存の取り組みにおいてセキュリティリスクの定義やその可視化が不十分である．S2C2F ではソースコード開発過程で発生しうる脅威 (e.g., 悪意あるコードの挿入，悪意ある依存ライブラリの挿入) を整理するものの，それらの根本的な原因となるセキュリティリスクの定義が明確でない．さらに，OSS のソースコード開発では明示的な開発の End of Support (EoS) や End of Life (EoL) が宣言されないまま開発プロジェクトが停滞する場合もあるが，S2C2F では明示的な EoS や EoL が宣言された場合の脅威のみを考慮する．このように，ソースコードの開発過程におけるセキュリティリスクの定義が不十分な事からそれらを定量的に示す指標の作成が定まらず，既存の SBOM ではそれらのリスクを定量的に可視化する取り組みが実施されていない．

本稿では，OSS の開発プロジェクトにおけるソースコード開発過程に着目し，それが生じる原因の観点からセキュリティリスクを定義すると共に，それらのリスクを OSS の Consumer に対して定量的に可視化する Source Code Management Visualizer (SCMV) と Extended SBOM (eSBOM) を提案する．本稿では，ソースコード開発過程のセキュリティリスクを (i) Malicious privilege escalation, (ii) Well-known vulnerability left, (iii) 0day vulnerability possible の 3 つと定義する．また，それらのセキュリティリスクの各々をそれが生じる原因の観点から定義し，定量的かつ外形的に観測可能な指標として整理し，可視化する．そして，これらの指標を基に eSBOM を構成する．SCMV は Consumer がパッケージマネージャを用いて OSS を取得する際に当該 OSS に関する eSBOM を作成して Consumer に提示する．本稿では，2022 年 7 月にスクレイピングされた，Node Package Manager (npm) [7] で Consumer が使用している上位 3000 個のパッケージ情報が入った，パブリックに公開されているデータセット [8] を使用して，Risk1 から 3 について，SCMV ツールを使用し，実在しているかどうかを検証，またリスクの可視化が行えるかどうか検証を行った．これらの評価を行った結果， eSBOM と SCMV によって，既存の SBOM では可視化されていないソースコード開発過程のリスクを可視化出来る事が分かった．

2. OSS サプライチェーンの分析

2.1 ソフトウェアサプライチェーンライフサイクルステージ

OSS の開発では，プロジェクトやコミュニティといった組織に通常複数人が所属して，ソフトウェアの開発が運営されている．開発されたソースコードは，GitHub 等のソースコードレポジトリで Source Code が公開される．ソースコードレポジトリとは，Git 等のバージョン管理システムを指しており，様々な SaaS が存在する．

図 1 は Producer がソースコードレポジトリ上で OSS のソースコードを開発して，その OSS が Package Manager [9] を用いて Consumer に配布されるまでのソフトウェアサプライチェーンの手続きを示す．これらの手続きは，大きく分けて以下の 3 つの過程に分類される: (a) ソースコード開発過程，(b) パッケージ開発過程，(c) パッケージ配布過程．

(a) の過程ではまず Producer は Source Code を作成する (図 1-(1))．Producer が作成した Source Code は Not Main Branches に格納される．Not Main Branches に格納された Source Code のうちの一部が Main Branch にマージされるよう Producer は Pull Request する．Maintainer は Pull Request された Source Code を品質や開発方針の点からレビューしてから Main Branch にマージする (図

© 2024 Information Processing Society of Japan

1-(2)). 通常 Maintainer は Source Code のマージに併せて Main Branch のバージョンを更新し，OSS は正式なリリースとしてソースコードレポジトリ上でパブリックに公開される．その際に，Consumer がソフトウェアの真正性を検証出来るように Maintainer がコード署名を添付する事がある．

(b) の過程では Main Branch 上で公開された Source Code を元に Consumer に配布する Package を作成して Package Registry 上で公開する．ここでの Package には Binary Code を含む Binary Package と，Source Code を含む Source Package の 2 種類の Package が想定される．Binary Package は以下の手順で作成される: まず Main Branch 上で公開された Source Code をコンパイルして Binary Code を生成して (図 1-(3))，機能テストを経て (図 1-(4))，Binary Code と真正性の検証に用いるメタデータを含んだ Binary Package が作成される (図 1-(5))．Source Package の作成手順は Source Code のコンパイルの過程をスキップして，(図 1-(4)) と Source Code と真正性の検証に用いるメタデータを含んだ Source Package が作成される．作成された Source Package と Binary Package は Package Registry 上で公開される (図 1-(6))．

(c) の過程では，Consumer は Package Registry 上で公開された Source Package と Binary Package を Package Manager を用いて取得する (図 1-(7))．Package Manager は Consumer に配布された package に含まれるソフトウェアを動作させる為に必要な依存ライブラリや競合ライブラリ等の問題を自動的に解決する．

2.2 OSS のソフトウェアサプライチェーンのリスク分析

OSS に内在するリスクを Consumer の視点で認識可能とする為には，図 1-(a), (b), (c) の各過程における主要なセキュリティリスクを定義した上でその可視化手段について論じる必要がある．本稿ではソースコード開発過程 (図 1-(a)) に内在するセキュリティリスクの定義と可視化手段が，既存研究においては何れも不十分と考えている．その視点での既存研究の分析は 3 章で行い，ソースコード開発過程におけるリスクの定義とそれを元にしたリスクの定量的な可視化手段を 4 章で提案する．

本節では，パッケージ開発過程 (図 1-(b)) とパッケージ配布過程 (図 1-(c)) に関するセキュリティリスク定義と可視化の手段を論ずる．パッケージ開発過程では，Main Branch で公開された Source Code と Dependent Libraries List を元に Source Package / Binary Package を作成して，それらの package を Package Registry に登録する．当過程では，Source Code と，作成済みの Source Package / Binary Package が改ざんされるリスクを考慮する必要がある．ソースコード開発過程に内在するセキュリティリスクのうちソースコードレポジトリの管理者権限の奪取に関

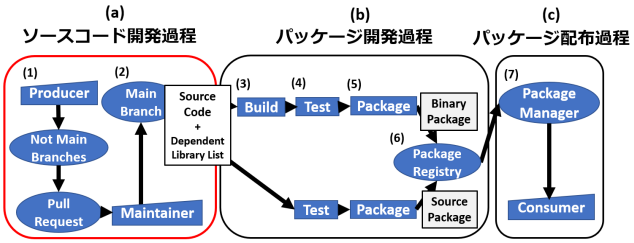

図 1 ソフトウェアサプライチェーンライフサイクルステージ

するリスクが十分に小さければ，Main Branch で公開された Source Code に対するコード署名は信頼出来る．この為，コード署名により真正と確認された Source Code と Dependent Libraries List をソースコードレポジトリから取得して Source Code と Source Package の改ざんリスクの有無は判断出来る．Binary Package は，ソースコードレポジトリ から取得した Dependent Libraries List と Source Code を元にコンパイルして生成された Binary Code を元にして Binary Package の改ざんリスクの有無は判断出来る．

パッケージ配布過程では，Package Manager が Binary Package / Source Package に含まれる Dependent Libraries List を元に取得する依存関係にあるライブラリの改ざんリスクを考慮する必要がある．この過程では，依存するライブラリについてもソースコード開発過程に内在するセキュリティリスクのうちソースコードレポジトリの管理者権限の奪取に関するリスクが十分に小さければ，パッケージ開発過程で示したのと同様の手順で依存ライブラリの Source Code や Source Package / Binary Package の改ざんの有無を判断出来る．

3. 関連研究

3.1 OSS サプライチェーンのセキュリティ構成管理

3.1.1 Taxonomy of Attacks on Open-Source Software Supply Chains

Taxonomy of Attacks on Open-Source Software Supply Chains[10] は，Producer 視点で悪意のある第三者によるソフトウェアサプライチェーンの攻撃についてまとめている．既存のパッケージへの攻撃について，ソースコード開発過程，ビルド過程，配布過程の 3 つの攻撃に分類が行われ，管理者権限等を所持していない人からの攻撃をスコープとしている．ソースコード開発過程のリスクでは，悪意のあるコードを善意の Maintainer と偽り Pull Request で挿入する事や，Git のような一般的なバージョン管理システムを侵害する等が挙げられている．当リスクを更に分類すると，Producer の開発環境に関する攻撃や管理者権限奪取の為にソーシャルエンジニアリング攻撃を行う等の事例が取り上げられている．

© 2024 Information Processing Society of Japan

3.1.2 Secure Supply Chain Consumption Framework (S2C2F)

S2C2F [5] は，OpenSSF が提唱する Consumer が OSS を安全に使用し，ソフトウェアサプライチェーン攻撃の防御し，リスクを軽減する為のフレームワークである．過去に発生したソフトウェアサプライチェーン攻撃を基に想定される脅威シナリオをいくつか提示し，それに対する対策案等が記載されている．

ソースコード開発過程のリスクは 5 種類定義され，攻撃者が意図的にソースコード内に脆弱性を仕込む，マルウェアの依存関係を追加，マルウェア等が無い良好なパッケージに対しての侵害，上流パッケージの脆弱性が改修されずに攻撃， OSS が EOL を迎えた為に脆弱性に対するパッチが提供されないリスクが記述されている．

3.2 OSS サプライチェーンの可視化

3.2.1 OpenSSF Scorecard – Security health metrics for Open Source

OpenSSF Scorecard [11] とは，OpenSSF が提供している，OSS のセキュリティ体制等を分析するツールである．当ツールは，Producer / Consumer 側の両方で使用する事ができ，18 のセキュリティ対策の指標を用いて当該パッケージの信頼度のスコアを算出し，サプライチェーン全体の脆弱性を検証する．検証を行うとそのレポジトリに対する OSS のパッケージ管理体制を示した点数が表示され，Producer 側は安全度を外部に公開する事が可能である．Consumer 側は，プロジェクトのセキュリティ姿勢を理解し，依存関係がもたらすリスクを評価出来る．

3.2.2 SBOM

SBOM [6] とは，ソフトウェアコンポーネントやそれらの依存関係，ライセンス等の情報を含めたリスト化したものであり，ソフトウェアの開発や選定，運用のそれぞれの場面で SBOM を使う利点がある．SBOM を構成する必要最低限の情報として，Data Fields, Automation Support, Practices and Processes の 3 つの分野に分類されると米国 National Telecommunications and Information Administration (NTIA) が発表した．これらの情報を用いる事により，ソフトウェアサプライチェーン全体を可視化する事に役立ち，脆弱性の管理やライセンス違反等をしていないかについて検証する事が可能である．

3.3 考察

3.3.1 リスク定義

一つ目に提示した，Taxnomy ではソフトウェアサプライチェーン全体のリスクについては，俯瞰してリスクの提起が行われている．しかし，本稿が提案するリスクであるソースコード開発過程の中で，悪意のある第三者による改ざんは考慮されているが，ソフトウェア脆弱性の残存リスクについては考慮されていない状態である．

二つ目の S2C2F の中で記述されている，Common OSS Supply Chain Threats は，過去のインシデント事例から，ソフトウェアサプライチェーン全体のリスクの定義をしている．しかし，S2C2F では，ソースコード開発の過程にスコープを絞った際に，リスクの発生原因に沿った整理が行われていない．具体的には，攻撃者が意図的にソースコード内に脆弱性を仕込む事や悪意のある依存関係をコードに追加するといった様々な事象のそれぞれをリスクと定義しているが，発生の根本原因を辿ると管理者権限が奪取される事が主な原因である．更には，開発の停滞で脆弱性の時間に修正を要する事案がカバーしきれておらず，Consumer がパッケージを安全かどうか判断する事は難しい現状である等，整理が不十分である．

3.3.2 リスクの可視化

OpenSSF Scorecard では，ソースコード開発過程のリスクの可視化が一部行われているが，Producer 視点と Consumer 視点での可視化が複合した指標となっている．そして，Producer 側でレポジトリの Admin 権限を所持している人しか得られない情報， Consumer 側からでは観測出来ない指標が含まれている．その為，Producer 側で，Scorecard を取り入れているような OSS では，勿論スコアを高くする為に対策を取るが，Scorecard を取り入れない場合は，スコアが低くなる等，セキュリティ上望ましくない OSS と捉えかねない．

SBOM は使用する OSS に関する依存関係を洗い出し，その情報を基に既知の脆弱性の情報と結びつける事により，OSS のセキュリティを管理しようとしている．しかし，それだけでは依存関係に関する脆弱性しか分からず，開発過程に関するリスクの可視化が不十分である．

4. 新規提案

本章では，2 つの新規提案を説明する．1 つ目は，OSS におけるソースコード開発過程のリスク定義の提案である．4.1 節で説明する．2 つ目は，定義したリスクを可視化する為のツール，Source Code Management Visualizer (SCMV) と Extended SBOM (eSBOM) についてである．4.2 節で説明する．

4.1 OSS におけるソースコード開発過程のリスク

本稿は OSS におけるソースコード開発過程のリスクを 4.1.1 – 4.1.3 節に示す通り 3 つ定義し，表 1 に示す．ソースコード開発過程のリスクは大きく分けて，二つの脅威モデルに分類でき，Risk 1 の悪意の第三者による改ざんリスクと Risk 2 / 3 のソフトウェア脆弱性の残存リスクである．後者の残存リスクは，公知となっている脆弱性と公知ではない，0day 脆弱性の二つに分けられる．

© 2024 Information Processing Society of Japan

表 1　脅威モデルにおけるリスクの定義について

分類		リスクの定義
悪意の第三者による改ざん		Risk 1: Malicious privilege escalation.
ソフトウェア脆弱性の残存	公知の脆弱性を認識済みだが改修していない	Risk 2: Well-known vulnerability left.
	0day 脆弱性が紛れ込む可能性が高い	Risk 3: 0day vulnerability possible.

4.1.1 Risk 1: Malicious Privilege Escalation

ソースコードリポジトリの Maintainer 権限が奪取され，悪意のあるコードが挿入されるケースを Risk 1 と定義した．パッケージに関する脆弱性等のバグが元々は存在せずとも，攻撃者がソースコードリポジトリへ不正アクセスをし，悪意のあるコードを挿入される事で侵害リスクとなる．昨今では Javascript 等を開発する上で使用される ESLint という npm のパッケージが攻撃を受け npm アカウントの認証情報を盗むコードが確認された [2]．当インシデントは，攻撃者がサードパーティの侵害によって不正に取得した，電子メールとパスワードを使用して npm アカウントにアクセスされた事が原因と発表している．悪意の第三者による改ざんでは，一つの OSS に何回か正常な Pull Request 等を送り，信頼を獲得した上で攻撃するといった内部犯行はスコープ外とする．

4.1.2 Risk 2: Well-known Vulnerability Left

パッケージに脆弱性が発見され Common Vulnerabilities and Exposures (CVE)[12] が発行されているが修正パッチが提供されていないケースを Risk 2 と定義した．

ソフトウェア脆弱性のライフサイクルについて，4 つの状態が存在する．(1) 脆弱性が発見されていない状態，(2) 脆弱性を誰かが見つけた状態 (0day), (3) 誰もが脆弱性を知っている (CVE が発行された状態) (4) セキュリティパッチが適用された状態．脆弱性が発見されていない (1) の状態は更に二つに分けられる．(1) - A 脆弱性を誰も見つけていない状態 (1) - B 脆弱性を故意に挿入されていない状態．誰もが脆弱性を知っている (3) の状態も更に二つに分けられる．(3) - A 脆弱性の存在が公知となっている状態 (3) - B 公知の脆弱性に対する修正パッチが提供されている状態である．

Risk2 は 3A で述べた，脆弱性の存在が公知であるにも関わらず，修正パッチが提供されていないといった放置状態を指す．OSS では Lodash という，npm のパッケージでプロトタイプ汚染の脆弱性が発見され，修正パッチを含む Pull Request を提出したが，2 か月もマージが行われなかったとの事例 [13] が存在する．マージするまでに 2 ヶ月もかかった直接的な原因は不明であるが，Maintainer 自身が修正する時間が無かった事やプロトタイプ汚染の脆弱性について理解していなかった事が考えられる．

プロプラエタリなソフトウエアである Windows でも同様の事例 [14] があった．Google の Threat Analysis Group が Windows に潜む脆弱性を発見し，Microsoft に通知したが 10 日後にはその情報を公開した．当脆弱性に対するパッチはまだリリースされておらず，最終的には 0day を Google に公開されてから 8 日後にパッチ適用となった．

4.1.3 Risk 3: 0day Vulnerability Possible

Risk 2 で述べた脆弱性ライフサイクルの 1A と 1B，(2) を前提とし，0day の脆弱性が紛れ込むケースを Risk 3 と定義した．当 Risk は，Maintainer の人的，資金的なリソース不足等の要因により， OSS のメンテンナンスが出来なくなり，開発プロジェクトをやむを得ず放棄する事や多数の機能アップデートによるコードレビューが追い付かない事といった，脆弱性が発生する前の状態にフォーカスを当てている [15]．2014 年に OpenSSL という SSL / TLS の仕組みを提供する OSS で発見された脆弱性が発生した原因の一つとして，人的，資金的なリソースが大幅に不足していた事が指摘されている．その為，コードレビューが追い付かず，Heartbleed といった脆弱性が発見された [3]．

4.2 ソースコード開発過程におけるリスクの可視化

4.2.1 eSBOM

eSBOM の詳細について，表 2 で説明する．Consumer 視点で外形的に観測可能である指標を用いて作成する．

Risk 1 を緩和する為に Active Maintainer 数と Branch Protection の有無，Protection のバイパスを可視化する．Active Maintainer 数を可視化する事により，Main Branch へマージするといった動作を行う Maintainer の人数を把握する事が出来る．Maintainer が 1 人では権限が集中している為，アカウントが乗っ取られた際等にレビューが行われないまま悪意のあるコードが挿入される可能性がある．デフォルトブランチの Protection Rule の有無を取得する事で，予め決められたルールに違反したマージが無いかを確認出来る．また，ルールを設定している事で，攻撃を隠す為にルールをバイパスしていないかを確認する事が出来る．具体的には，レビューがされていない，CI / CD のテストが成功せずにマージが行われている場合には，意図的な脆弱性が挿入されようとしている事を推察する事が出来る．

Risk 2 を緩和する為に，対象のパッケージのパッチの管理状態を確認する．National Vulnerability Database (NVD) [16] で CVE にパッチタグ [17] が付与されているかを可視化する．過去の脆弱性で CVE が発行されているにも関わらず，脆弱性が放置されていないか，対象のパッケージにパッチが提供されているかどうかを可視化する．

Risk 3 である 0day 脆弱性が紛れ込む可能性を推測する為に，ソースコード開発体制の状態であるを確認する．具体的には，どれぐらいのコードが変更されたか，初めてコ

© 2024 Information Processing Society of Japan

表 2　eSBOM のデータ形式

Risk	可視化の対象	可視化の情報
Risk1	Maintainer 権限の状態	•Active Maintainer 数 •Branch Protection の有無 •Branch Protection をバイパスしていないか
Risk2	パッチの管理状態	•NVD でのパッチタグ付与の状態: 'Third Party Advisory', 'Vendor Advisory', 'Patch'
Risk3	ソースコード開発体制の状態	• 各コミットのコード変化量 • コミットに参加した人の中で何人が初めてのコミットか • アーカイブかどうか • レポジトリの最終コミット日時 •Consumer によるダウンロード数

ミットした人はどの程度いるかについてを確認する．一般的に脆弱性が生まれやすい要因として 3 つ考えられる [19]．コードの複雑さ，一つのファイルのコードの更新量，開発者の活動メトリクスである．1 つ目については，コード自体の問題であり，異常検知をするには静的解析等の技術である為，スコープ外とする．2 つ目については，過去のコミットでファイルがどの程度変更されたかを可視化する．Maintainer の数と見合わない大量のコードやファイルを追加，変更していた場合には，セキュリティのバグを見逃す可能性が高くなる．3 つ目については，初めて当レポジトリにコミットや Pull Request をした人を確認する事で，当該パッケージに関するコードのセキュアコーディングに慣れていないコードが紛れ込む可能性を可視化する事が出来る．

当該パッケージのメンテナンスが困難な状態になっていないかといった事を可視化する為に，アーカイブ [18] かどうか，レポジトリの最終コミット日時，Consumer 側でダウンロード数がどれくらいかを確認する．アーカイブを宣言している場合は，メンテナンスが行わない事の意思表示が行われている為，脆弱性が発見された場合でも放置する．また，宣言していなくとも，180 日以上コミットがされていない場合には，放棄されたと考える [20]．パッケージが破棄されていないかについて，Producer 側の推測だけでなく，ダウンロード数等から Consumer 側でも放置がされていないか，コードを確認する人が減っていないかを確認する．これらの情報からソースコード開発体制を可視化する事で，0day が紛れ込む可能性を推測する．

4.2.2　SCMV

Consumer が OSS のパッケージをインストールする際，SCMV はそのパッケージが孕む Risk 1, Risk 2, Risk 3 各々のリスクの情報を取得し，eSBOM の形式に沿って情報を格納する．そして，それらの情報を使用し，OSS のパッケージをインストールするユーザーに対して，リスクを観測可能な指標に落とし込み可視化する．SCMV を使用する事によって，Consumer 自身がインストールを試みる OSS のパッケージのリスクの程度を把握した上で，OSS をインストールするか判断する事が出来る．

5.　実装: eSBOM と SCMV

SCMV は Python3.11 で動作するように実装を行った．また，eSBOM は他の SBOM ツールと連携しやすいように JSON 形式で格納する設計を行った．

本実装では，JavaScript を対象としたパッケージマネージャである npm[7] を取り扱う．パッケージマネージャは npm 以外にも PyPI や RubyGems 等が存在するが，パッケージインストール時に任意のスクリプトの実効を許容しており，比較的容易に悪意あるソフトウェアの拡散を許容する事 [21]，またパッケージ総数や年間ダウンロード数の点で Python や Ruby より Node.js の方が広く使用されている事を考慮して，本稿では npm を選択した．

5.1　SCMV と eSBOM の動作

図 2 に Consumer が OSS をインストールする際に SCMV を使用し，eSBOM を提示されるまでの手順を示す．

A （3）でパッケージと依存関係のパッケージの情報を取得すると並行して，その情報をパッケージマネージャから SCMV に送信する．

B A で取得した情報（ソースコードリポジトリの情報）を基に Risk 1 や Risk 3 を緩和する為に必要なソースコードの状態や Active Maintainer 数等の情報を取得するリクエストを出す．

C1 eSBOM に Risk 1 及び Risk 3 を可視化する為の情報を格納する．

C2 また，Risk2 を可視化する為の，パッケージに関連するパッチ関連の情報を NVD から取得し，C1 と同様に eSBOM に格納する．

D B や C で得られた eSBOM の情報を SCMV に送信する．

E SCMV が eSBOM の情報をユーザーに伝える．

5.2　Risk 毎の情報の取得内容と実装について

本章では，Risk 毎にどのような情報を取得するか，そし

© 2024 Information Processing Society of Japan

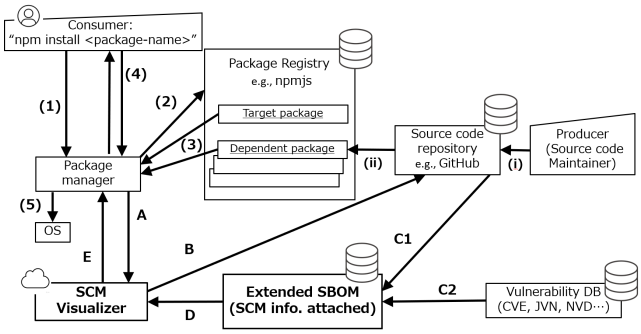

図 2 SCMV によるソースコード開発過程リスクの可視化手法

てどのように取得し，実装を行ったかについて述べる．リスクを可視化する為に，Risk 1 と 3 では，GitHub の API を使用し，Risk 2 では NVD の API を使用した．

5.2.1 Risk 1

Risk 1 の Active Maintainer については，Main Branch へのマージの動作を行ったユーザーを指す．4.2.1 でも書いた通り，Maintainer が MFA を導入しているかやウイルススキャンの結果等の情報については，攻撃者から狙われやすくなるといった弊害が生まれてしまう為，GitHub の API といった各種ソースコードレポジトリでは取得が出来ない．その為，Maintainer 権限を推察する為に，Pull Request からマージする人を Maintainer として，過去 30 件の Pull Request から誰がマージしたかをカウントした．

他には，Branch Protection の有無を確認し，レビューが行われているか，最新のコミットに対してのレビューが Approval になっていない，テストに成功していないにも関わらずマージされていないか等を確認する．

5.2.2 Risk 2

Risk 2 のパッチタグについては，レポジトリ名を NVD の CVE キーワード検索にかける．そして，CVE が存在した場合には最新の 10 件の CVE の情報を取得する．NVD には，Reference Tag と呼ばれる，Third Party Advisory や Vendor Advisory といった参考情報にタグを付けるシステムが存在する．Reference Tag から'Third Party Advisory', 'Vendor Advisory', 'Patch' といったパッチに関するタグが付与されているかどうかを確認する．

5.2.3 Risk 3

Risk 3 のソースコード開発体制の状態については，過去 30 件の Pull Request の中で，コードの増加，減少，ファイルが変更されたかについてを確認する．また，1 回の GitHub API のリクエストで取得出来る限界の 100 コミットの中と直近 30 コミットを比較し，初めてのコミットである人は誰かを確認する．また，レポジトリの放置状態については，アーカイブを宣言していないか，レポジトリの最終コミット日時を GitHub の API から取得し，npm の API から週のダウンロード数を取得する．

6. 定量評価

本章では，本稿で提案した OSS のソースコード開発過程のリスク定義 (Risk1 - Risk3) が妥当か，リスクの可視化が有効かどうかについて SCMV を使用し，評価を行った．

6.1 方法

可視化の為の情報の収集や可視化の内容については，5.2 節でも書いた通りである．npm は 2022 年 9 月時点で，210 万以上のパッケージがレジストリに登録されている．その中でも，2022 年 7 月にスクレイピングされた，Consumer が使用している上位 3000 個の npm パッケージの情報が入ったパブリックデータセット [8] を使用して，Risk1 から 3 について，SCMV ツールを使用し，実在しているかどうか，リスクの可視化が行えるかどうか検証を行った．

6.2 結果と考察

3000 個のパッケージ内で存在しない，参照が出来なかったパッケージが 121 個存在し，除くと 2879 個であった．Risk1 と Risk3 については，URL の情報が無ければ，Maintainer，レポジトリの状態や最後にいつコミットしたか等と言った情報を取得する事が出来ない．

6.2.1 SCMV ツールを実行した結果

Risk 1 の Active Maintainer については，最大 30 人で，0 人が 85 個，1 人が 1214 個，2 人以上が 1580 個であった．0 人は，Main Branch に直接 Push を行う事や，マージという動作は行わずに特定のブランチから Force-Push を行っている場合である．Maintainer 以上の権限を所持している人が複数に存在し，コミュニティ内でマージを実行する担当者が決定している事も考えられるが，それでもその 1 人に権限が集中している事が考えられる．Branch Protection を付けているパッケージが 1656 個，付けていないパッケージが 1223 個であった．その中でも，バイパス疑いのある，レビューが無い事やテストが通っていないにも関わらず，マージされているパッケージは，0 回が 90 個のパッケージ，1 回以上が 1566 個である事が分かった．

Risk 2 のパッチタグについては，3 つのパターンに整理を行った．1. CVE にそもそもパッケージが登録されていない為安全であるか，脆弱性は存在するが，報告されていないケース．2. CVE に登録されているが，パッチが提供されていないケース．当ケースの場合は，実際に CVE には登録されている為，脆弱性が報告されているが，パッチが提供されていない為，使用する事は明確に危険だと分かる．3. CVE に登録されているが，パッチが提供されているケースである．当ケースの場合は，脆弱性が報告された場合に，パッチも提供するリポジトリである事が確認されているケースであり，これからもパッチが提供される確率

© 2024 Information Processing Society of Japan

が高いと考えられる．1 のケースは 1829 個，2 のケースは 72 個，3 のケースは 978 個であった．

Risk3 のソースコード開発体制については，アーカイブされているパッケージが 115 個存在した．最終のコミットについては 180 日以上経過したパッケージは，681 個存在している事が分かった．各コミットのコード変化量やダウンロード数については，API から取得する事は出来たが，パッケージ毎のコードの総量等によってリスクが変わる為，それぞれで評価をする必要がある．

6.2.2 リスクの可視化が有効かどうか

SCMV ツールによって，Consumer が使用している上位 3000 個の npm パッケージの中にリスクが明確に存在する事が分かった．Risk 1 については，そもそも 0 人という Pull Request を経由せず，レビュー等が行われずに開発が行われているといった攻撃を受けやすいパッケージが可視化出来る事が分かった．また，ブランチプロテクションを付けていないパッケージが多い事や付けていてもバイパス疑いと思われるパッケージがほとんどである事が分かった．Risk 2 については，CVE に登録されているが，パッチが提供されていないという，脆弱性が明らかに放置されているパッケージが可視化出来る事が分かった．Risk 3 については，アーカイブと宣言し，メンテナンスを行わないまた，コミットが 180 日行われておらず，破棄されていると推測また，可視化出来る事が分かった．コードの変化量を見る事で，Active Maintainer に比例していないコード変更を可視化する事やダウンロード数を見る事でまだメンテナンスを維持するのではないか等と推測する事が出来た．

7. まとめと今後の課題

本稿では，OSS におけるソースコード開発過程に注目し，リスク定義を行った．OSS の利用者に対してリスクを可視化し，意思決定をする為の機構，eSBOM と SCMV ツールを設計した．NPM の 3000 件のデータセットを使用し，SCMV を使って，リスクを可視化してみた結果，3 つのリスクの定義が妥当である事，直観的にリスクを解釈する事が出来る事が分かった．今後の課題としては，今回提案したツールでは，昨今発生した xz パッケージのバックドアが設置出来る脆弱性を生み出された原因の一つである，ソーシャルエンジニアリング攻撃のリスクを可視化出来ていない．そのような攻撃が，今後も増加していくと考え，将来的には SCMV のツールを拡張し，そのような攻撃に対してのリスクを可視化出来るようにしたい．

参考文献

[1] P. Bruce, Perens Open Source Definition LG #26, Linux Gazette, 1998

[2] ESLint, Postmortem for Malicious Packages Published on July 12th, 2018. 2018, https://eslint.org/blog/2018/07/postmortem-for-malicious-package-publishes/ Online Available.

[3] J. Walsh, grc-daily Free can make you bleed. http://http://security.grc-daily.com/dsp_getFeaturesDetails.cfm?CID=3482, Online Available.

[4] OpenSSF, About OpenSSF, 2023. https://openssf.org/about/. Online Available.

[5] OpenSSF, Secure Supply Chain Consumption Framework (S2C2F), 2023. https://github.com/ossf/s2c2f. Online Available.

[6] NTIA, SBOM at a Glance, 2021

[7] E. Wittern, P. Suter, S. Rajagopalan. A look at the dynamics of the JavaScript package ecosystem, MSR'16, Pages 351–361, 2016

[8] Hugging Face, K Soni, S. Gautam, K. D. Reddy, Top NPM Packages Dataset, 2022. https://huggingface.co/datasets/deepklarity/top-npm-packages. Online Available.

[9] S. G. Hegde and G. Ranjani, Package Management System in Linux, ASIANCON'21, pp. 1-6, 2021

[10] P. Ladisa and H. Plate and M. Martinez and O. Barais, Taxonomy of Attacks on Open-Source Software Supply Chains, arXiv:2204.04008, 2022

[11] OpenSSF, Security scorecards for open source projects, 2024. https://scorecard.dev/Online Available.

[12] MITRE, CVE, 2024. https://cve.mitre.org/. Online Available.

[13] Jake, Lodash: Understanding the recent vulnerability and how we can rally behind packages, 2020. https://medium.com/@jake_52237/lodash-understanding-the-recent-vulnerability-and-how-we-can-rally-behind-packages-c19a2a630254 Online Available.

[14] Google, N. Mehta, B. Leonard, Threat Analysis Group, Disclosing vulnerabilities to protect users 2016. https://security.googleblog.com/2016/10/disclosing-vulnerabilities-to-protect.html Online Available.

[15] G. Avelino, E. Constantinou, M. Valente, A. Serebrenik, On the abandonment and survival of open source projects: An empirical investigation, arXiv:1906.08058, 2019

[16] NIST, National Vulnerability Database General Information, 2023. https://nvd.nist.gov/general. Online Available.

[17] NIST, National Vulnerability Database Understanding Vulnerability Detail Pages, 2023. https://nvd.nist.gov/vuln/vulnerability-detail-pages. Online Available.

[18] GitHub, Archiving repositories, 2023. https://docs.github.com/en/repositories/archiving-a-github-repository/archiving-repositories. Online Available.

[19] Y. Shin, A. Meneely, L. Williams and J. A. Osborne, Evaluating Complexity, Code Churn, and Developer Activity Metrics as Indicators of Software Vulnerabilities, IEEE Transactions on Software Engineering, vol. 37, no. 6, pp. 772-787, 2011.

[20] X. Li, S. Moreschini, F. Pecorelli and D. Taibi, OSSARA: Abandonment Risk Assessment for Embedded Open Source Components, in IEEE Software, vol. 39, no. 4, pp. 48-53, 2022

[21] Marc Ohm, Henrik Plate, Arnold Sykosch, and Michael Meier, Backstabber' s Knife Collection: A Review of Open Source Software Supply Chain Attacks, arXiv:2005.09535, 2020

© 2024 Information Processing Society of Japan