オープンソースソフトウェアに対するセキュリティリスク指標の提案と評価

葛野 弘樹[1,a)]　矢野 智彦[2]　山内 利宏[3]

概要：オープンソースソフトウェア（OSS）を用いた情報システムが主流となり，OSS に対するセキュリティ対策は急務である．OSS の開発は開発者コミュニティが担うことが多く，利用者による独自の修正は容易ではない．また，OSS に脆弱性が発見された場合において，情報システムの利用環境毎に脆弱性への対応が必要であるかの判断は難しい．既存のセキュリティ研究では，ソフトウェアの脆弱性の分類，脆弱性の危険性の推定，ならびに脆弱性の攻撃利用可能性に関する解析手法が提案されている．しかし，運用中の情報システムで利用中の OSS の脆弱性や開発状況を分析し，攻撃を受ける可能性の把握や，脆弱性が発見された際に危険性の有無を判断することは困難であり，依然として課題である．本稿では，このような課題を解決するため，OSS のセキュリティリスク指標を提案し，情報システムで利用する OSS の危険性の継続的な把握を可能とする．提案手法では，脆弱性に関する情報と OSS の開発状況を結び付け，セキュリティリスク指標を算出可能とする．情報システムの運用において，提案する OSS のセキュリティリスク指標をセキュリティ対策の判断基準として用いることで，セキュリティ保守の実施判断に利用可能になる．評価においては，提案するセキュリティリスク指標を用いて，複数の OSS に対して危険性を把握可能か検証を行った．また，Linux ディストリビューションに提案手法を適用し， OSS のセキュリティリスク指標の算出にかかるコストを検証した．

Design and Evaluation of Security Risk Indication for Open Source Software

Hiroki Kuzuno[1,a)]　Tomohiko Yano[2]　Toshihiro Yamauchi[3]

Abstract: Open source software (OSS) become a software mainstream. The security of OSS is an important topic for information systems using OSS. When vulnerabilities are discovered in OSS, it is difficult to fix or address OSS for each information system environment. Existing security studies proposed classifying vulnerabilities, estimating vulnerability risks, and analyzing vulnerability exploitability. However, it is still difficult to understand the vulnerabilities threat, and development status of OSS used in information systems operation. The challenge is to determine whether vulnerabilities and OSS development status are dangerous or not. In this paper, we propose a security risk Indication for OSS to tackle these problems. The proposed method calculates security risk indications by combining vulnerability information and the development status of OSS. The proposed security risk Indication of OSS as a criterion for security measures in the operation of information systems. In the evaluation, we verified whether the proposed security risk indication can be used to identify the threats of multiple OSS and the cost of calculating security risk indications.

[1] 神戸大学 大学院工学研究科
Graduate School of Engineering, Kobe University, Japan
[2] セコム株式会社 IS 研究所
Intelligent Systems Laboratory, SECOM Co., Ltd., Japan
[3] 岡山大学 学術研究院自然科学学域
Faculty of Natural Science and Technology, Okayama University, Japan
[a)] kuzuno@port.kobe-u.ac.jp

1. はじめに

情報システムにおいて，オープンソースソフトウェア（OSS）の利用は主流となりつつある．攻撃に利用可能なソフトウェアの実装不備は脆弱性とされ，脆弱性への対応としてソフトウェアの修正と更新が必要になる．OSS の

開発は，開発者コミュニティで行われることが多く， OSS に対し，情報システムの運用者や開発者が独自に修正を行うことは困難である．情報システムにおいて，多数の OSS が利用されている場合，OSS 毎の開発状況は把握できない可能性がある．また，情報システムの動作環境や脆弱性を利用した攻撃の動向により， OSS のセキュリティ対策として，脆弱性への対応が必要かの判断は難しい．

脆弱性情報は Common Vulnerabilities and Exposures（CVE）として一意に識別可能な番号が付与され，National Vulnerability Database（NVD）にて脆弱性に関する情報が管理されている [1], [2], [3], [4] （詳細は 2 節を参照）．

既存のソフトウェアの脆弱性に対する研究において，脆弱性の与える影響度の分析 [5]，脆弱性の危険性に関する評価 [6], [7] ，ならびに脆弱性の攻撃利用に関する調査 [8] が行われている．また，報告された脆弱性毎に攻撃での利用可能性に関する解析手法が提案されている [9], [10].

情報システムの運用において，OSS に対するセキュリティ対策は運用者の必要と判断したタイミングで行うことが望ましい．そのため，既存の手法では，次の課題がある．

課題： OSS のセキュリティリスク把握

既存手法では，運用中の情報システムにおける OSS に対し，報告された脆弱性に基づき，セキュリティにおける脅威を把握し，セキュリティ対策としてソフトウェア更新可否の判断を行う．しかし，第三者の開発する OSS のセキュリティリスクを適切に捉えるには，脆弱性に対する修正や開発が継続されているかに基づき，OSS の開発状況に関する情報も重要である．OSS の危険性の有無を迅速かつ継続的に把握することで，OSS の更新や利用の見直しなど，適切なセキュリティ対策の実施が必要である．

本稿では，情報システムで利用する OSS に対してセキュリティ対策を行うかどうか判断するため， OSS の開発状況ならびに脆弱性情報を紐づけて算出するセキュリティリスク指標を提案する．OSS のセキュリティリスク指標により，脆弱性有無や開発状況を継続して網羅的に把握し，セキュリティ対策実施の判断基準に用いることを可能とする．

提案するセキュリティリスク指標の算出のため，脆弱性情報は NVD より収集を行い， OSS 情報は，Linux ディストリビューションの管理するパッケージ一覧から収集を行う．収集した情報に対して，脆弱性情報に含まれるソフトウェアの識別情報を利用し，OSS 情報と脆弱性情報の紐づけを行う．次に，脆弱性のリスクを数値化した Common Vulnerability Scoring System（CVSS），ならびに OSS の開発状況を数値化する Criticality Score を利用し，OSS のセキュリティリスク指標を算出する．

情報システムで利用する OSS に対し，脆弱性情報と OSS 情報を関連付け，セキュリティ対策実施の判断基準となるようなセキュリティ指標の提案はまだない．本研究は，情報システムでの利用が広まる OSS を網羅し，危険性の把握を継続的に行うことで，いち早くセキュリティ対策に繋げるための基準を提供するものである．

本稿での研究貢献は以下の通りである：

（1） 情報システムで利用される OSS に対するセキュリティ対策に活かすため， 脆弱性情報，ならびに OSS 情報を紐づけた OSS のセキュリティリスク指標を提案した．Linux ディストリビューションにおいて，OSS のセキュリティリスク指標を算出，各種 OSS のセキュリティリスク指標を提供可能とした．

（2） 提案する OSS のセキュリティリスク指標の評価として，OSS の危険性を把握可能か検証を行った．また，特定の Linux ディストリビューションに含まれる複数のパッケージに対し，セキュリティリスク指標の算出にかかる計算コストを評価した．

表 1 Types of Vulnerabilities [13]

Exploit	Content
DoS	Forcing shutdown of software
Code Execution	Arbitrary program execution
Overflow	Breaking of stack or heap memory space
Memory Corruption	Illegal overwriting in software memory space
SQL Injection	Arbitrary SQL insertion and execution
XSS	Arbitrary HTML or JavaScript code in Web application
Directory Traversal	Reading arbitrary directory
HTTP Response Splitting	Poisoning of cache by illegal HTTP response
Bypass Something	Evading of access limitations
Gain Information	Information gaining through illegal control
Privilege Escalation	Getting of administrator privilege
CSRF	Illegal HTTP request is accepted by Web application
File Inclusion	Forced reading of malicious file reading

表 2 Vulnerabilities Information

Item	Description
CVE	A list of common identifiers for publicly known cybersecurity vulnerabilities
CPE	A structured naming scheme for systems, software, and packages
CVSS	A characteristics and severity of software vulnerabilities

2. 背景知識

2.1 脆弱性情報

脆弱性は攻撃に利用可能なソフトウェア動作に影響を与える実装不備である．脆弱性に関して，**表 1** に攻撃内容から脆弱性を 14 種類に分類している．

本稿で用いる脆弱性情報を**表 2** に示す．CVE は共通脆弱性識別情報として，脆弱性毎に一意の番号が割当てられる． Common Platform Enumeration（CPE）は，共通プラットフォーム一覧として，脆弱性の報告されたハードウェアやソフトウェアを示す．また，CVSS は共通脆弱性スコアリングシステムとして，脆弱性の数値化に利用される．

CVSS は CVSSv2，ならびに CVSSv3 がある．**表 3** に CVSSv3 で用いられる要素を示す．脆弱性を利用した攻撃に必要な要素や脅威度に応じて複数の要素を加味して数値

表 3 CVSSv3 Metrics[4]

Score Metrics	Metrics	Description
Base Score Metrics	Exploitability Metrics	It reflects the ease and technical means by which the vulnerability can be exploited
	Impact Metrics	It reflects the direct consequence of a successful exploit
Temporal Score Metrics		It reflects exploit techniques or code availability, the existence of any patches or workarounds, or the confidence
Environmental Score Metrics	Exploitability Metrics	It customizes the CVSS score depending on the importance of the affected IT asset to a user's organization
	Impact Metrics	It customizes the Confidentiality, Integrity, and Availability Impacts
	Impact Subscore Modifiers	It customizes the Confidentiality, Integrity, and Availability Requirements

表 4 OpenSSF Criticality Score [11]

Parameter (S_i)	Description
created_since	Time since the project was created (in months)
updated_since	Time since the project was last updated (in months)
contributor_count	Count of project contributors (with commits)
org_count	Count of distinct organizations that contributors belong to
commit_frequency	Average number of commits per week in the last year
recent_releases_count	Number of releases in the last year
closed_issues_count	Number of issues closed in the last 90 days
updated_issues_count	Number of issues updated in the last 90 days
comment_frequency	Average number of comments per issue in the last 90 days
dependents_count	Number of project mentions in the commit messages

化が行われる．

2.2 OSS 情報

2.2.1 OSS Criticality Score

OSS のうち，オペレーティングシステム，ライブラリ，データベース，ならびに言語処理系などの基盤ソフトウェアは多数の組織で利用されており，OSS 毎の開発状況の把握は困難である．Open Source Security Foundation（OpenSSF）では，OSS の開発状況を Criticality Score として，**表 4** の情報に基づき，次の式による数値化を提案している [11]．

$$C_{project} = \frac{1}{\sum_i \alpha_i} \sum_i \alpha_i \frac{\log(1+S_i)}{\log(1+\max(S_i, T_i))} \qquad (1)$$

ただし，$C_{project}$ は OSS プロジェクトとし，表 4 のパラメータ S_i 毎に，重み α_1，閾値 T_i が定義される． Criticality Score は，$0 \leq C_{project} \leq 1$ の範囲をとり，0 の場合はクリティカルではない（least-critical），1 の場合はクリティカル（most-critical）としている．Criticality Score により，OSS の開発状況を数値化して捉えることが可能である．OSS としての開発の活発度に応じて変動することから，一定の指標として利用可能である．

2.2.2 OSS パッケージ管理

Linux ディストリビューションでは，OSS のパッケージ管理により， OSS の導入を容易している．Linux ディストリビューションの 1 つである Debian GNU/Linux においては，`deb` パッケージ形式により，OSS を管理し，配布している [12]．

`deb` パッケージに含まれる開発状況に関する情報を**表 5** に示す． `changelog` ファイルにて，パッケージバージョン，更新に関する日時や情報， `control` にアーキテクチャや OSS のレポジトリ情報が含まれている．

3. 想定モデル

本稿における想定モデルでは，脆弱性情報，ならびに OSS 情報として開発状況から高いセキュリティリスク指標が算出されることとする．

3.1 想定環境

想定モデルにおいて，セキュリティリスク指標を算出可能な環境は以下とする．

- OS，および OSS
  - OS：OSS をパッケージとして管理（例，Debian GNU/Linux）
  - OSS：OS にてパッケージ管理されており，開発元情報が参照可能（例，GitHub レポジトリ）
- 脆弱性情報： OSS に関し，CVE 番号が割当てられた脆弱性情報が公開され，CVSSv3 の数値が算出可能
- OSS 情報：GitHub に登録されており，OSS Criticality Score が算出可能

3.2 想定シナリオ

想定シナリオとして，OSS のセキュリティリスク指標は変動することを考慮する．OSS において，開発状況は一定ではない可能性がある．また，過去の脆弱性に加え，新たな脆弱性が発見され，CVE 登録と CVSS が追加されることが予想される．

例として，計算機上にて多数の OSS から利用されるパッケージ化されたライブラリにおいて，開発リポジトリの更新頻度は減少していく．また，過去の脆弱性が一定数報告されており，新規に発見された脆弱性に対して，対応されず修正が行われない．

提案する OSS のセキュリティリスク指標では，想定モ

表 5　Debian changelog and control information [12]

File	Field	Description
changelog	package name	The source package name
	version	The version number
	distribution	The distributions where this version should be installed when it is uploaded
	urgency	The value for the Urgency field (i.e., low, medium, high, emergency, or critical)
	Maintainer	The maintainer name and email address
	Date	The date format (e.g., day-of-week, dd month yyyy hh:mm:ss +zzzz)
control	Source	This field identifies the source package name
	Maintainer	The package maintainer’s name and email address
	Version	The version number of a package
	Section	An application area into which the package has been classified.
	Priority	How important it is that the user have the package installed
	Homepage	The URL of the web site for this package
	Package	The name of the package
	Architecture	Debian machine architecture
	Depends	It requires certain binary packages
	Description	A description of the binary package

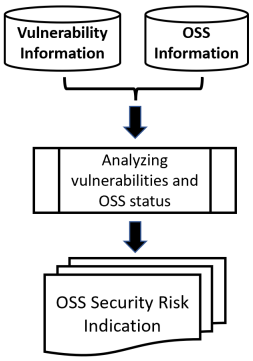

図 1　提案手法の概要

デルにおける OSS が想定シナリオに従う場合，OSS のセキュリティリスク指標は増加することを仮定する．

4. 提案手法

4.1 要件

提案する OSS のセキュリティリスク指標では，計算機に導入された OSS に対し，監視対象を指定し，脆弱性情報ならびに OSS 情報に基づいて，OSS のセキュリティリスク指標を算出可能とする．設計において，次の要件を満たすことを目指した．

要件：　計算機に導入された OSS に対する脆弱性を利用した攻撃（例，特権奪取攻撃や DoS 攻撃）や開発状況を利用した攻撃を想定する．監視対象とする OSS の脆弱性情報，OSS 情報を定期的に取得し，得られた情報に合わせて一定周期毎にセキュリティリスク指標を算出する．

4.2 設計

提案する OSS のセキュリティリスク指標の概要を **図 1** に示す．要件を満たすため，脆弱性情報の一覧，監視対象とする計算機上の OSS 情報の一覧を保持する．OSS のセキュリティリスク指標の算出において，脆弱性情報，ならびに OSS 情報からセキュリティリスク指標として数値化する．

4.2.1 脆弱性情報

提案する OSS のセキュリティリスク指標において，計算機上で動作する OSS の脆弱性の把握に用いる脆弱性情報は次の通りである．

- 脆弱性情報： 脆弱性を一意に示す情報（e.g., CVE），脆弱性を有するソフトウェアを示す情報（e.g., CPE），ならびに脆弱性を数値化した情報（e.g., CVSS）．

4.2.2 OSS 情報

提案する OSS のセキュリティリスク指標の算出において，計算機上で動作する OSS の開発状況の把握に用いる OSS 情報は次の通りである．

- OSS 情報： OSS の開発状況を数値化した情報（e.g., Criticality Score）．

4.2.3 セキュリティリスク指標

定期的な OSS の危険性の把握のため，脆弱性情報，ならびに OSS 情報を利用して提案する OSS のセキュリティリスク指標の算出を次のように行う．

- 脆弱性情報と OSS 情報の紐づけ：脆弱性情報に含まれるソフトウェアの識別子を利用し，OSS 情報との紐づけを行う．
- セキュリティリスク指標の算出：OSS 情報と紐づけされた脆弱性の全てのスコア値（e.g., CVSS）の調和平均，ならびに OSS 情報から得られたスコア値（e.g., Criticality Score）を組合わせ，セキュリティリスク指標の算出を行う．

4.3 実現方式

計算機上で動作する OSS を監視対象とするため，実現方式を考案した．実現方式の想定する環境は，OS を Debian GNU/Linux，x86_64 CPU アーキテクチャとしている．

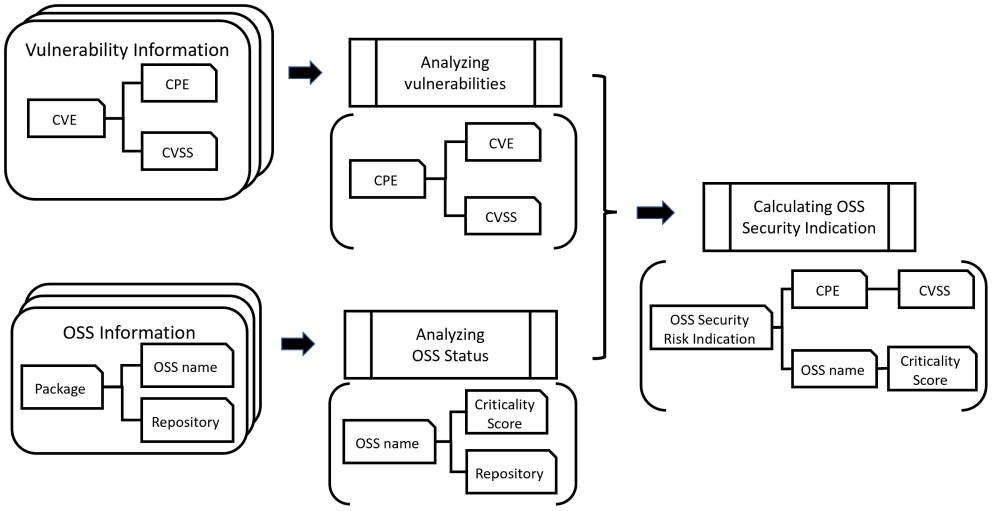

図 2 OSS のセキュリティリスク指標算出の流れ

提案する OSS のセキュリティリスク指標の算出の流れを 図 1 に示す．実現方式において，NVD より脆弱性情報の取得と解析，パッケージから OSS 情報の取得と解析を行い， CVSS および Criticality Score から OSS のセキュリティリスク指標の算出を行う．

4.3.1 脆弱性情報の取得と解析

実現方式を備えた計算機上にて， 定期的に脆弱性情報の取得を行い，OSS 情報と紐づけ可能とするための解析を行う．脆弱性情報の取得と解析処理は次のとおりである．

（1） NVD より脆弱性情報を取得

（2） NVD に含まれる全ての脆弱性情報を解析

（a） 全ての脆弱性に対して，OSS 情報との紐づけに用いるため，CVE 毎に CPE からソフトウェア識別子を調査し，CPE に対する CVE リストを作成

（b） 全ての CVE に対して CVSSv3 の有無を調査し，CPE に対する CVSSv3 のリストを作成

4.3.2 OSS 情報の取得と解析

実現方式を備えた計算機上において， 定期的に OSS 情報の取得を行い，脆弱性情報と紐づけ可能とするための解析を行う． Debian における OSS 情報の取得にあたり，`deb` パッケージの管理機能ならびに `deb` パッケージに含まれるファイルの記載情報を利用する．OSS 情報の取得，ならびに解析処理は次のとおりである．

（1） `deb` パッケージの管理機能を利用し，計算機上の OSS のパッケージ一覧情報を取得

（2） `deb` パッケージの管理機能を利用し，OSS 毎の `deb` パッケージのソースコードを取得

（a） 全ての `deb` パッケージに対して，`control` ファイルを調査し，レポジトリ情報を取得

（b） レポジトリ情報が存在する場合，OSS の `deb` パッケージ毎に Criticality Score を算出し記録

（c） OSS の `deb` パッケージ毎に CPE と紐づけ可能なソフトウェア識別子を生成

4.3.3 脆弱性情報と OSS 情報の紐づけ

実現方式を備えた計算機上において，OSS 毎に脆弱性情報，ならびに OSS 情報の紐づけを行い，OSS に関する CVSS と Criticality Score から OSS のセキュリティリスク指標を算出する．

（1） `deb` パッケージの管理機能を利用し，計算機上の OSS のパッケージ一覧情報を取得

（2） 全ての `deb` パッケージに対し，次の処理を順に行う

（a） `deb` パッケージに対し，CPE と紐づけ可能なソフトウェア識別子を利用し，CPE から CVE 一覧，CVSS 一覧を取得

（b） `deb` パッケージに対し，算出した Criticality Score 一覧を取得

（c） `deb` パッケージに対し，CVS 一覧，ならびに Criticality Score 一覧から OSS のセキュリティリスク指標を算出

4.3.4 OSS のセキュリティリスク指標

OSS のセキュリティリスクの算出には，$0 \leq cvss_i \leq 0.5$ の範囲で正規化した OSS に関する CVSS 一覧 $\{cvss_1, cvss_2, \ldots, cvss_n\}$，ならびに $0 \leq cs_i \leq 0.5$ の範囲で正規化した Criticality Score cs_{date} を利用する．OSS として，`deb` パッケージのセキュリティリスク指標 S_{oss} を次のように算出する．

$$S_{oss} = \frac{\sum_{i=1}^{n} w1_i}{\sum_{i=1}^{n} \frac{w1_i}{cvss_i}} + cs_{date} \tag{2}$$

ただし，$\{w1_1, w1_2, \ldots, w1_n\}$ は各 CVSS に対する重み，cs_{date} は Criticality Score 測定日時の値とする．S_{oss} は，$0 \leq S_{oss} \leq 1$ の範囲をとり，0 の場合は危険性はない，1 の場合は危険性を有するとする．OSS のセキュリティリスク指標により，OSS に関する脆弱性の数値化である CVSS ならびに，開発の活発度を示す Criticality Score を複合的に数値化して捉えることが可能である． OSS に関する危険性の変動を把握でき，一定の指標として利用可能である．

5. 評価

5.1 評価の目的

提案する OSS のセキュリティリスク指標を想定モデルに基づく環境上の OSS に対して適用し，OSS の危険性を把握可能か評価を行った．また，性能評価として，特定の OS 環境において，多数のパッケージを監視対象とした際にかかるコスト把握を目的とし，算出にかかる時間を確認した．評価項目と内容を以下に示す．

(1) OSS に対するセキュリティリスク指標の評価
提案する OSS のセキュリティリスク指標について，OSS の危険性の有無を網羅的に把握可能か評価するため，11 のプログラミング言語における OSS 144 個に対して算出した．

(2) OSS のセキュリティリスク指標の算出にかかる時間
提案する OSS のセキュリティリスク指標を Linux 環境におけるパッケージ管理対象の OSS に対して，算出にかかる時間を計測した．

5.2 評価環境

OSS のセキュリティリスク指標の評価を行う計算機環境として，CPU Intel(R) Core(TM) i7-12700H (2.30GHz, 14 コア)，Memory 16 GBytes を備えた計算機を用い， OS は Debian GNU/Linux 12.0 (Linux kernel 5.4.0, x86_64) とした．提案手法である OSS のセキュリティリスク指標の算出プログラムは Python 言語にて 610 行で実現した．また，NVD に含まれる CVSSv3 は Base Score Metrics による数値 [1] とし，Criticality Score 算出の閾値はデフォルトのパラメータを利用した [11].

5.3 OSS に対するセキュリティリスク指標の評価

提案する OSS のセキュリティリスク指標の評価として，11 個のプログラミング言語 (C，C++，C#，Java 系，JavaScript 系，PHP，Go，Python，Ruby，Rust，Shell) の合計 144 個の OSS に対する OSS のセキュリティリスク指標の推移を**表 6**, **表 7** に示す． 各 Criticality Score の測定日において，過去 1 年間の CVE と CVSS を用いている．

表 6，表 7 より，2022 年 8 月 15 日時点でセキュリティリスク指標が最も高い OSS は，rails であり，脆弱性の報告による高い CVSS，ならびに Criticality Score の増加が確認できた．また，2022 年 8 月 15 日時点でセキュリティリスク指標の最も低い OSS は，oss-fuz であり，過去 1 年間で脆弱性報告がなく，また Criticality Score も増加していないことが確認された．その他の OSS においても同様にセキュリティリスク指標の増減を確認できる．

一定の指標を用いることで，複数の OSS を網羅的に調査し，期間ごとの脆弱性報告数，ならびに開発状況の推移を OSS の状況として比較可能である．そのため，OSS のセキュリティリスク指標を起点に OSS に関する危険性に変化が生じることを把握できるといえる．

表 6 OSS セキュリティリスク指標 1 (各言語から 5 個を抜粋)

OSS (CPE)	Lang	2020/12/24	2022/6/7	2022/8/15
git	C	0.6668	0.6035	0.6223
linux	C	0.5982	0.7597	0.7511
php-src	C	0.9544	0.7990	0.8218
openssl	C	0.8123	0.5717	0.6160
systemd	C	0.8066	0.6943	0.8260
cmssw	C++	0.4807	0.4202	0.4756
tensorflow	C++	0.6716	0.6006	0.6448
ceph	C++	0.8595	0.4265	0.9144
bitcoin	C++	0.5848	0.7066	0.7338
electron	C++	0.9327	0.9091	0.7503
mono	C#	0.7969	0.3947	0.8571
azure-powershell	C#	0.4132	0.3840	0.4006
runtime	C#	0.6439	0.8311	0.7188
aspnetcore	C#	0.4075	0.3848	0.4148
Umbraco-CMS	C#	0.4050	0.3876	0.3985
kubernetes	Go	0.9932	0.6618	0.7223
go	Go	0.6434	0.7710	0.7640
cockroach	Go	0.4271	0.4061	0.4400
origin	Go	0.9234	0.8838	0.7306
istio	Go	0.8911	0.7224	0.8205
spark	Scala	0.9834	0.4313	0.9065
gradle	Groovy	0.5843	0.9096	0.9097
flink	Java	0.6628	0.4087	0.4388
hadoop	Java	0.9244	0.4125	0.9343
spring-boot	Java	0.4242	0.6751	0.9184

表 7 OSS セキュリティリスク指標 2 (各言語から 5 個を抜粋)

OSS (CPE)	Lang	2022/12/24	2022/6/7	2022/8/15
DefinitelyTyped	TypeScript	0.4671	0.4112	0.4588
webpack	JavaScript	0.6314	0.4099	0.4157
ant-design	TypeScript	0.4387	0.4075	0.4408
babel	JavaScript	0.4375	0.8032	0.8202
angular	TypeScript	0.9347	0.7907	0.9477
symfony	PHP	0.6954	0.9394	0.9679
magento2	PHP	0.4644	0.4209	0.4572
PrestaShop	PHP	0.5057	0.9158	0.7011
framework	PHP	0.9116	0.5631	0.6181
joomla-cms	PHP	0.4328	0.4106	0.4471
salt	Python	0.7130	0.8755	0.8937
core	Python	0.9357	0.7650	0.8090
cpython	Python	0.4340	0.3992	0.4515
scikit-learn	Python	0.9303	0.4102	0.4386
numpy	Python	0.4299	0.9095	0.9358
rails	Ruby	0.4621	0.6751	0.9636
homebrew-core	Ruby	0.4488	0.4002	0.4326
homebrew-cask	Ruby	0.4471	0.4154	0.4424
metasploit-framework	Ruby	0.4421	0.4118	0.4371
brew	Ruby	0.4288	0.7110	0.7281
rust	Rust	0.9900	0.6453	0.7924
servo	Rust	0.4389	0.7662	0.7527
cargo	Rust	0.4017	0.5519	0.5728
rust-clippy	Rust	0.3853	0.3905	0.4033
tokio	Rust	0.3665	0.8721	0.7791
gentoo	Shell	0.7000	0.4265	0.4526
opam-repository	Shell	0.3858	0.3711	0.3794
bioconda-recipes	Shell	0.3815	0.3662	0.3954
core	Shell	0.8799	0.7331	0.7429
oss-fuzz	Shell	0.3756	0.3799	0.3875

5.4 OSS のセキュリティリスク指標の算出にかかる時間

提案する OSS のセキュリティリスク指標の算出にかかる時間として， 144 個の Debian パッケージに対して OSS のセキュリティリスク指標の算出を行い評価した．OSS 毎に 10 回計測し，平均値を算出し，合計した． NVD は解析

表 8　OSS セキュリティリスクの算出時間

Item	Proposed mechanism
Total	24 m 40.0 s
Average	10.27 s
Median	6.0 s
linux	4.0 s
Cataclysm-DDA	2 m 28.0 s

済みとし，deb パッケージのダウンロード時間は含めない．評価結果を**表 8** に示す．提案する OSS のセキュリティリスクの算出では，合計で 24 分 40 秒を必要とした．1 つの deb パッケージあたりにかかる平均時間は 10.27 秒，中央値は 6.0 秒である．また，最も算出に時間を要した OSS は Cataclysm-DDS であり，2 分 28 秒を示した，また，最も算出時間が少ない OSS は Linux であり，4 秒を示した．

6. 考察

6.1 評価に対する考察

提案する OSS のセキュリティリスク指標において，特定の OSS に対する脆弱性の報告時，および開発状況の停滞時を想定し，セキュリティリスク指標の推移から危険性を把握可能であるか試みた．提案する OSS のセキュリティリスク指標の算出結果より，いずれの OSS の状況もセキュリティリスク指標の変動が確認され，状況の変化を把握可能なことを示した．提案する OSS のセキュリティリスク指標は動作中の計算機において，多数の OSS を監視対象とする．実際に利用している OSS のセキュリティに関する状況を継続して把握可能であり，脆弱性情報と開発状況の変動を監視するために利用可能である．

性能評価結果より，OSS セキュリティリスク指標の算出は 1 つのパッケージ数あたり，平均 10.27 秒かかる．計算機上にて，定期的に OSS セキュリティリスク指標を算出する場合，計算機性能へ影響を与える可能性がある．しかし，主なコストはパッケージ情報の取得，ならびに通信にかかる時間である．継続して，定期的な OSS セキュリティリスク指標の算出を行う場合，計算機の負荷の少ないタイミングで行うことで， OS およびアプリケーション動作への直接的な負荷は低減できると考えている．

6.2 提案手法の考察

提案する OSS のセキュリティリスク指標の算出においては， OSS の特定の情報として，脆弱性情報として脆弱性のリスクを数値化した CVSS，ならびに OSS の情報として，開発レポジトリから得られる Criticality Score を対象とし，危険性の把握を行っている．提案する OSS のセキュリティリスク指標では，脆弱性の特定は行わないが，攻撃に利用される可能性のある脆弱性の存在を把握した場合，ならびに開発が停滞するに従いセキュリティリスクは高くなり，将来的に攻撃を受ける可能性があると考えられる．

OSS を利用した情報システムの運用を継続するためには，セキュリティリスク指標の結果から攻撃緩和策の適用を進めなければならない．OSS のセキュリティリスク指標をセキュリティ対策の判断基準とするため，計算機上で動作中の OSS の危険性推定が必要である．脆弱性や開発状況毎に攻撃につながる可能性の把握をさらに進め，常時提供できるような枠組みを検討している．

6.3 限界

6.3.1 OSS のセキュリティリスク指標

提案する OSS のセキュリティリスク指標における OSS の情報の取得は，各 OS におけるパッケージ情報を起点としている．パッケージ情報に開発元情報が含まれていない場合， Criticality Score の算出を行えない可能性がある．また，パッケージ管理以外から導入した OSS の場合，利用状況を補足することが難しく，OS 全体の実行ファイルの調査と解析が必要である．OSS のセキュリティリスク指標を直接的に算出するためには，動作中の OSS のみを調査し，監視対象とする必要がある．

6.3.2 セキュリティリスク指標の算出対象

提案する OSS のセキュリティリスク指標のみでは，OSS の開発状況，修正状況，ならびに提供状況を確実には把握できていない可能性がある．今後の課題として，OSS の開発における，ソースコード変更内容，修正状況としてバグトラッカーとパッチ内容，メーリングリスト等のコミュニケーションツールの解析，そして OSS 配布に関する情報が重要である．各情報の必要性を加味し，セキュリティリスク指標の算出に活かせる項目の検討を進める予定である．

7. 関連研究

脆弱性の分析

脆弱性の分析として，攻撃に利用される脆弱性の動向の分類を行う手法 [14]，複数の脆弱性が利用される可能性に関する分析手法が提案されている [15]．また，攻撃が行われたタイミングについて調査されており [8]，脆弱性のリスクや攻撃利用時の影響について議論されている [5], [6]．

脆弱性の攻撃利用予測

報告された脆弱性について，攻撃で利用される可能性が指摘されており [16], [17]， VEST では，脆弱性の攻撃可能性と公開のタイミングに関する分析手法を提案している [18]．また，EPSS は脆弱性に関して，実際に攻撃が行われる可能性についての分析手法を提案している [9], [10]．

脆弱性の自動分類

脆弱性の種別や概要を分析し，ディープラーニングや機械学習を用いて自動的に分類する手法が提案されている [19], [20]．また，CVSS を利用し，脆弱性の危険性や対策の優先度に関する推定手法が提案されている [7], [21]．

脆弱性の管理

脆弱性管理として，脆弱性の開示タイミングの調査や効

果について分析されており [22], [23]，脆弱性の管理やパッチ提供のタイミングも調査されている [24], [25], [26].

8. おわりに

本稿では，情報システムで利用が進む OSS に対し，危険性を把握可能とするため，セキュリティリスク指標を提案した．提案手法では，脆弱性情報として CVE，CPE，CVSS，ならびに OSS の開発状況の数値化を紐づけ，OSS のセキュリティリスク指標を算出する．

提案する OSS のセキュリティリスク指標は，OSS の脆弱性情報と OSS 情報をふまえ，OSS のセキュリティリスクを継続的に把握できるため，セキュリティ対策を実施する際の判断基準の一つとして参照可能である．

評価において，提案する OSS のセキュリティリスク指標を利用し，複数の OSS に対する危険性の有無を把握可能か検証した．また，Linux ディストリビューションに対してセキュリティリスク指標を適用し，OSS に対するセキュリティリスク指標の算出にかかるコストを検証した．

謝辞 本研究の一部は，JSPS 科研費 JP19H04109，JP22H03592，ならびに 2022 年度国立情報学研究所公募型共同研究（22S0302）の助成を受けたものです．

参考文献

[1] NIST, National Vulnerability Database (online), available from ⟨https://nvd.nist.gov/⟩ (accessed 2022-08-18).

[2] MITRE, Common Vulnerabilities and Exposures (online), available from ⟨https://www.cve.org/⟩ (accessed 2022-08-18).

[3] NIST, Official Common Platform Enumeration Dictionary (online), available from ⟨https://nvd.nist.gov/products/cpe⟩ (accessed 2022-08-18).

[4] FIRST, Common Vulnerability Scoring System SIG (online), available from ⟨https://www.first.org/cvss/⟩ (accessed 2022-08-18).

[5] Allodi, L.: Economic Factors of Vulnerability Trade and Exploitation. *Proc. the 24th ACM SIGSAC Conference on Computer and Communications Security*, pp. 1483–1499 (online), DOI: 10.1145/3133956.3133960 (2017).

[6] Allodi, L., et al.: Security Events and Vulnerability Data for Cybersecurity Risk Estimation. *Risk Analysis*, vol. 37, no. 8, pp. 1606-–1627 (online), DOI: doi:10.1111/risa.12864 (2017).

[7] Nikonov, A., et al.: System for Estimation CVSS Severity Metrics of Vulnerability Based on Text Mining Technology. *Proc. the 2021 Information Technology and Nanotechnology*, pp. 1-5 (online), DOI: 10.1109/ITNT52450.2021.9649232 (2021).

[8] Householder, D, A., et al.: Historical Analysis of Exploit Availability Timelines, *Proc. the 13th USENIX Workshop on Cyber Security Experimentation and Test*, (2020).

[9] Jacobs, J., et al.: Improving Vulnerability Remediation Through Better Exploit Prediction, *Journal of Cybersecurity*, vol. 6, no. 1, DOI: 10.1093/cybsec/tyaa015 (2020).

[10] Jacobs, J., et al.: Exploit Prediction Scoring System, *Digital Threats Research and Practice*, vol. 2, no. 3, DOI: 10.1145/3436242 (2021).

[11] OpenSSF, Open Source Project Criticality Score (Beta) (online), available from ⟨https://github.com/ossf/criticality_score⟩ (accessed 2022-08-18).

[12] Debian Project, Debian GNU/Linux (online), available from ⟨https://www.debian.org/⟩ (accessed 2022-08-18).

[13] CVE details, Vulnerabilities By Type (online), available from ⟨https://www.cvedetails.com/vulnerabilities-by-types.php⟩ (accessed 2022-05-05).

[14] Williams, A., W., et al.: Analyzing Evolving Trends of Vulnerabilities in National Vulnerability Database. *Proc. 2018 IEEE International Conference on Big Data*, pp. 3011–3020 (online), DOI: 10.1109/BigData.2018.8622299 (2018).

[15] Gong, X., et al.: Joint Prediction of Multiple Vulnerability Characteristics Through Multi-Task Learning. *Proc. 24th International Conference on Engineering of Complex Computer Systems*, pp. 31–40 (online), DOI: 10.1109/ICECCS.2019.00011 (2019).

[16] Goyal, P., et al.: Discovering Signals from Web Sources to Predict Cyber Attacks. arxiv (online), available from ⟨https://arxiv.org/abs/1806.03342⟩ DOI: 10.48550/arXiv.1806.03342 (2018).

[17] Martin, H., et al.: Survey of Attack Projection, Prediction, and Forecasting in Cyber Security. *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 640–660, DOI: 10.1109/COMST.2018.2871866 (2018).

[18] Chen, H., et al.: VEST: A System for Vulnerability Exploit Scoring & Timing. *Proc. the Twenty-Eighth International Joint Conference on Artificial Intelligence*, pp. 6503–6505 (online), DOI: 10.24963/ijcai.2019/937 (2019).

[19] Minh, L, H, T., et al.: DeepCVA: Automated Commit-level Vulnerability Assessment with Deep Multi-task Learning. *Proc. 36th IEEE/ACM International Conference on Automated Software Engineering*, pp. 717-729 (online),DOI: 10.1109/ASE51524.2021.9678622 (2021).

[20] Siewruk, G., et al.: Context-Aware Software Vulnerability Classification Using Machine Learning. *IEEE Access*, vol. 9, pp. 88852-88867, DOI: 10.1109/ACCESS.2021.3075385 (2021).

[21] Walkowski, M., et al.: Automatic CVSS-based Vulnerability Prioritization and Response with Context Information. *Proc. International Conference on Software, Telecommunications and Computer Networks*, pp. 1–6 (online), DOI: 10.23919/SoftCOM52868.2021.9559094.559094 (2021).

[22] Mitra, S., et al.: The effects of vulnerability disclosure policy on the diffusion of security attacks. *Information Systems Research*, vol. 26, no. 3, pp. 565-–584, (2015).

[23] Boechat, F., et al.: Is Vulnerability Report Confidence Redundant? Pitfalls Using Temporal Risk Scores. *IEEE Security & Privacy*, vol. 1, pp. 2–11, (2021).

[24] Walkowski, M., et al.: Vulnerability Management Models Using a Common Vulnerability Scoring System. *Applied Sciences*, vo. 11, no. 8735. DOI: 10.3390/app11188735

[25] Dey, D., et al.: Optimal policies for security patch management. *INFORMS Journal on Computing*, vol. 27, no. 3, pp. 462–477, (2015).

[26] August, T., et al.: Market segmentation and software security: Pricing patching rights. *Management Science*, vol. 65, no. 10, DOI: 10.1287/mnsc.2018.3153 (2019).