Visualization Method for Open Source Software Risk Related to Vulnerability and Developmental Status Considering Dependencies

Yano, Tomohiko

Kuzuno, Hiroki

(Citation)
Journal of Information Processing,32:767-778

(Issue Date)
2024

(Resource Type)
journal article

(Version)
Version of Record

Journal of Information Processing Vol.32 767–778 (Sep. 2024)

[DOI: 10.2197/ipsjjip.32.767]

Regular Paper

Visualization Method for Open Source Software Risk Related to Vulnerability and Developmental Status Considering Dependencies

Tomohiko Yano[1,a)] Hiroki Kuzuno[2,b)]

Received: December 6, 2023, Accepted: June 10, 2024

Abstract: In recent years, Open-source software (OSS) has become a mainstream technology essential to information systems. However, its secure application requires a comprehensive understanding of its various security risks. One of them is vulnerability risk. A vulnerability risk involves the discovery of a new vulnerability in the OSS in use, which must be immediately addressed by security administrators, such as software updates. On the other hand, developmental risks involve OSS that are not in active development. If the development of an OSS is stalled, an alternative OSS should be considered because newly identified vulnerabilities may not be fixed. Therefore, a specialized method is required to analyze vulnerability and developmental risks of OSS, while accounting for their dependencies. This paper proposes a method that identifies such security risks of OSS by extracting, linking, and visualizing the vulnerabilities, development status, and dependency information. The proposed method enables security administrators to check visualization results, identify OSS with security risks, and consider appropriate countermeasures. We experimentally evaluate the adequacy of the visualizations for the purpose of the identification of security risks, and calculate the processing time required to visualize the risks.

Keywords: open-source software, vulnerability management, visualization

1. Introduction

The utilization of Open-Source Software (OSS) has become increasingly popular in information systems, with many organizations already using OSS in their commercial products and services. According to a report by Synopsys Inc., 96% of audited codebases contain OSS [2].

However, the secure use of OSS requires a comprehensive understanding of their security risks. The use of vulnerable OSS compromises the integrity, confidentiality, and availability of confidential assets owing to security flaws. To protect OSS against vulnerability attacks, their vulnerability and developmental risks must be analyzed in detail.

(1) **Vulnerability risks**: These involve publicly disclosed vulnerabilities that can be exploited to compromise the security of information systems. To prevent such risks, publicly disclosed vulnerabilities must be assessed and appropriate countermeasures, such as updating the software.

(2) **Developmental risks**: These are potential risks in OSS whose development has stagnated. If the development of an OSS is stalled, newly identified vulnerabilities may not be fixed. To prevent such risks, the development status of each OSS should be checked and alternative OSS should be used, if necessary.

Moreover, besides the OSS directly in use, its dependent OSS should also be considered. For example, Apache Log4j, in which CVE2021-44228 [3] is an easily exploitable risk, is often used in the back-end of other software packages. Even if you are not using Apache Log4j directly, you may be using it within an OSS dependency used by your organization.

Thus, **vulnerability**, **developmental**, and **dependency** information have to be analyzed comprehensively to identify risks during OSS utilization.

However, existing studies address only a few of these factors such as the analysis and management of vulnerabilities [4], [5], [6], proposal of risk indicators for repositories [7], and development of tools for the visualization of dependencies [8].

In addition, to enable security administrators to take immediate action based on the linked information, a representing method is also important. This highlights the need for visualizing vulnerability, developmental, and dependency information in an appropriate format.

The aforementioned considerations lead us to the central research problem considered in this study:

Research problem: The secure use of OSS requires a thorough understanding of their vulnerability and developmental risks, considering their dependencies. However, existing studies have not considered these factors simultaneously.

[1] Intelligent Systems Laboratory, SECOM CO., LTD., Mitaka, Tokyo 181–8528, Japan

[2] Graduate School of Engineering, Kobe University, Kobe, Hyogo 657–8501, Japan

[a)] tomo-yano@secom.co.jp

[b)] kuzuno@port.kobe-u.ac.jp

This paper is an extended version of a paper published in the Eleventh International Symposium on Computing and Networking (CANDAR 2023) [1]

Therefore, a visualization method is required to aid security administrators to rapidly process such types of information.

In this paper, we propose a method to link and visualize vulnerability, development, and dependency information. First, the proposed method obtains the Common Vulnerability Exposure (CVE) as vulnerable information, the status of GitHub repository as developmental information, and the package management system as dependency information. It also calculates a development score representing developmental risks based on the repository information. Then, it links this information by extracting vulnerability, development, and dependency information related to the OSS names. Finally, the information is visualized as a directed graph.

The proposed method visualizes OSS that a particular OSS depends on (depend-OSS) and those that depends on it (revdepend-OSS) in the form of a directed graph, using vulnerabilities and OSS as keys. If an OSS exhibits vulnerabilities or a high development score, the corresponding information is also linked to them.

The output visualization can be used by security administrators to highlight potentially affected OSS when a vulnerability is disclosed, or to identify vulnerability and developmental risks in dependencies associated to a given OSS during regular operation. Therefore, it helps security administrators identify OSS with security risks and formulate suitable responses.

The major contributions of this study are as follows:

(1) **Visualization of information:** The proposed method enables the visualization of vulnerability, developmental, and dependency information. This enables security administrators to check the visualized results and decide on appropriate action to protect against OSS with vulnerability and developmental risks, considering dependencies.

(2) **Evaluation of visualized results:** The proposed method is evaluated in terms of output visualizations by adding new vulnerabilities and changing the development score. The proposed method is also evaluated in terms of processing time to ensure practical applicability.

The remainder of this paper is structured as follows. The background knowledge required to understand the rest of this paper is discussed in Section 2, and the assumptions regarding the components and considered scenarios of the proposed method are described in Section 3. In Section 4, the proposed method is introduced in detail, and it is evaluated in Section 5. The evaluation results are analyzed and the limitations of the proposed method are discussed in Section 6, and related work is overviewed in Section 7. Finally, the study is concluded in Section 8.

2. Background

2.1 Vulnerability Information

Vulnerabilities are technical flaws in any software that can be exploited for attacks [9]. Well-known software vulnerabilities are indexed as Common Vulnerabilities and Exposures (CVE) by MITRE, a non-profit organization, and each vulnerability is assigned a unique ID.

CVE information is provided on MITRE [10], National Vulnerability Database (NVD) [11], and OS vendor websites. For example, the Debian Security Bug Tracker [12] provides vulnerability information in the JavaScript Object Notation (JSON) format. The information provided is presented in **Table 1**. An example of JSON format provided by the Debian Security Bug Tracker is presented in Listing 1.

Table 1 Vulnerability information of Debian Security Bug Tracker [12].

Field	Description
description	The description of the vulnerability
debianbug	Related ID in the Debian Bug report logs
scope	Scope of impact of vulnerability exploitation
repositories	Current version in Debian
status	Status of fixed vulnerabilities
fixed_version	Fixed version in Debian
urgency	Urgency of the vulnerability

```
{
    "apache-log4j2": {
        "CVE-2021-44228": {
            "description": "Apache Log4j2 2.0-beta9
                 through 2.15.0 (excluding security
                 releases 2.12.2, 2.12.3, and 2.3.1)
                 JNDI features used in configuration,
                 log messages, and parameters do not
                 protect against attacker controlled
                 LDAP and other JNDI related endpoints.
                 ...",
            "debianbug": 1001478,
            "scope": "local",
            "releases": {
                "bookworm": {
                    "status": "resolved",
                    "repositories": {
                        "bookworm": "2.19.0-2"
                    },
                    "fixed_version": "2.15.0-1",
                    "urgency": "not yet assigned"
                },
                "bullseye": {
                    "status": "resolved",
                    "repositories": {
                        "bullseye": "2.17.1-1~deb11u1",
                        "bullseye-security": "2.17.0-1~
                             deb11u1"
                    },
                    "fixed_version": "2.15.0-1~deb11u1",
                    "urgency": "not yet assigned"
                },
                ...
            }
        },
        ...
    },
    ...
}
```

Listing 1 Example of vulnerability information from Debian Security Bug Tracker [12]

2.2 Developmental Information

Because the source code of an OSS is publicly available, its repository information, such as commits, issues, and release information, can be obtained using the GitHub REST API [13].

The proposed method uses the Criticality Score (CS) [14] proposed by Open-Source Security Foundation (OpenSSF), which quantitatively represents the importance of the OSS. It is defined as follows (Eq. (1)).

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

CS comprises ten parameters, $S_0 - S_9$. These parameters are

Table 2 OpenSSF Criticality Score [14].

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

Table 3 Control information [19].

Field	Description
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 has the package installed
Homepage	The URL of the website 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

```
$ apt-cache show dkms
Package: dkms
Version: 2.8.4-3
Installed-Size: 295
Maintainer: Dynamic Kernel Modules Support Team <
     dkms@packages.debian.org>
Architecture: all
Provides: dh-sequence-dkms
Depends: kmod | kldutils, gcc | c-compiler, dpkg-dev,
     make | build-essential, coreutils (>= 7.4), patch,
     dctrl-tools
Pre-Depends: lsb-release
Recommends: fakeroot, sudo, linux-headers-686-pae |
     linux-headers-amd64 | linux-headers-generic | linux
     -headers
Suggests: menu, e2fsprogs
Description-en: Dynamic Kernel Module Support Framework
DKMS is a framework designed to allow individual kernel
     modules to be upgraded
without changing the whole kernel. It is also very easy
     to rebuild modules as
you upgrade kernels.
Description-md5: b7b6bb6a6b083b2245e0648e7752a459
Multi-Arch: foreign
Homepage: https://github.com/dell-oss/dkms
Tag: admin::kernel, devel::buildtools, devel::lang:c,
     devel::library,
devel::packaging, implemented-in::c, implemented-in::
     shell,
interface::commandline, role::devel-lib, role::program,
     scope::utility,
suite::debian, works-with::software:source
Section: kernel
Priority: optional
Filename: pool/main/d/dkms/dkms_2.8.4-3_all.deb
Size: 78240
MD5sum: a89a700dd0c1758a86713ebb7be5fea1
SHA256: 34
     f45872c4c164e838092ddfb098b800d9fd38c25aaf7a2624
81d6c8b87dffdf
```

Listing 2 Example of control from apt-cache command

described in **Table 2**. Each S_i is assigned a threshold, T_i, and a weight, α_i. Collectively, they are indicative of the development status of the OSS. The proposed method uses the criticality score as the development score by adjusting α_i such that $C_{project}$ is higher for OSS with less active development.

2.3 Dependency Information

The proposed method uses the package management system to obtain dependency information. Package management system is used in OSs and programming languages to automate the management of programs and libraries installed on computers and resolve dependencies. As programming languages and package management system, pip is used for Python [15], and gem is used for Ruby [16]. On the other hand, dpkg and apt are used as OS package management system for Debian Linux [17], and rpm is used for RedHat Linux [18]. In Debian Linux, OSS is managed in the form of deb packages, and package information is recorded in a control file (Control). Control includes the fields listed in **Table 3**. An example of Control is presented in Listing 2.

3. Assumption

3.1 Components

This section describes the component, including the target computer to which the proposed method can be applied, and the flow of information. The proposed method comprises the following components (**Fig. 1**):

Target computer: The computer to be analyzed and information about the OSS (package), including dependencies, is extracted. The OS is Debian GNU/Linux, and the OSS is managed by the package management system.

Package information: The package information can be obtained from Control. The dependency information can be obtained from the "Depends" field and the repository URL can be obtained from the "Homepage" field.

Vulnerability information: CVE information is available on the OSS (e.g., the Debian Security Bug Tracker [12]).

Repository information: Repository information can be obtained from GitHub repository using GitHub API.

Analysis computer: The analysis computer accepts the package, vulnerability and repository information as input, analyzes them, and outputs the directed graph.

Administrator: The administrators take action based on the output directed graphs. Specific scenarios are described in the scenarios (Section 3.2).

3.2 Considered Scenarios

This section describes considered scenarios. Security administrators can respond to OSS security risks during the regular operation and during disclosure of new vulnerabilities. Therefore, the following describes the scenarios for each response.

(1) **Security administrator during regular operation:** Security administrators regularly output directed graphs describing dependencies with the OSS as a key. If an unaddressed vulnerability exists within the dependencies, appropriate action is taken, e.g., the OSS is updated. If an OSS exhibits a high development score, its use is terminated and alternative OSS is considered.

(2) **Security administrator during disclosure of new vulnerabilities:** When a new vulnerability is disclosed, security administrators output graphs describing the dependencies with the vulnerability as a key. The output directed graph indicates the affected OSS, and the administrator takes appropriate actions based on the graph. For example, if an OSS that is widely used in the organization is listed as one of the dependencies, the administrator updates the vulnerable OSS immediately; otherwise, the action is addressed later, such as during scheduled maintenance.

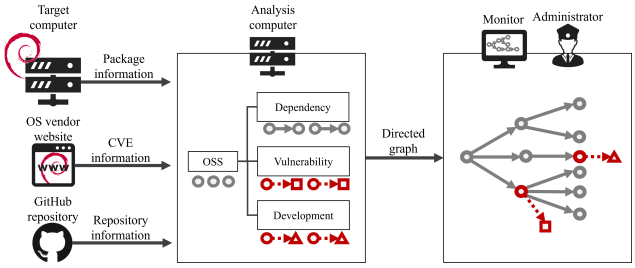


Fig. 1 Components: Components consist of a target computer that is assessed, an analysis computer that analyzes the obtained information and outputs a directed graph, and a monitor that displays the directed graph. The administrator checks the results displayed to the monitor.

4. Methods

4.1 Requirements

The proposed method enables security administrators to identify vulnerabilities and OSS with high developmental risk easily within dependencies. We aimed to satisfy the following requirements:

Requirements: The proposed method requires visualization of depend-OSS or revdepend-OSS with vulnerability and development score information. Therefore, obtaining and linking vulnerability, developmental, and dependency information is essential.

4.2 Overview

An overview of the proposed method is presented in **Fig. 2**.

The proposed method can be subdivided into three stages —gathering information, linking information, and constructing the directed graph. In the first stage, CVE information is obtained as vulnerability information, repository information as developmental information, and package information as dependency information. In the second stage, vulnerability, developmental, and dependency information are extracted and linked based on the obtained information, using the OSS name. Dependency information extracts both depend-OSS and revdepend-OSS. Finally, a directed graph is constructed based on the linked information.

4.3 Definition of Directed Graph

The output directed graph is represented by $G = (V; E)$, where V denotes the set of vertices, v_i; and $E \subset V \times V$ is the set of edges, $e_{ij} = (v_i; v_j)$. V represents OSS information $o_i \in O$, vulnerabilities $c_i \in C$, and development scores $s_i \in S$. E represents dependencies, $depend_{o_i o_j}$, and additional information on vulnerabilities, $vuln_{o_i c_j}$, and development score, $score_{o_i s_j}$. $depend_{o_i o_j}$ is represented by o_i, depending on o_j.

An example of a directed graph is depicted in **Fig. 3**. Its edges indicate the dependencies of the OSS. For example, OSS A in Fig, 3 depends on OSS B, whereas OSS C depends on OSS A. Thus, in the case of OSS A, the "depend-OSS" is OSS B, and the "revdepend-OSS" is OSS C. A high vulnerability score and

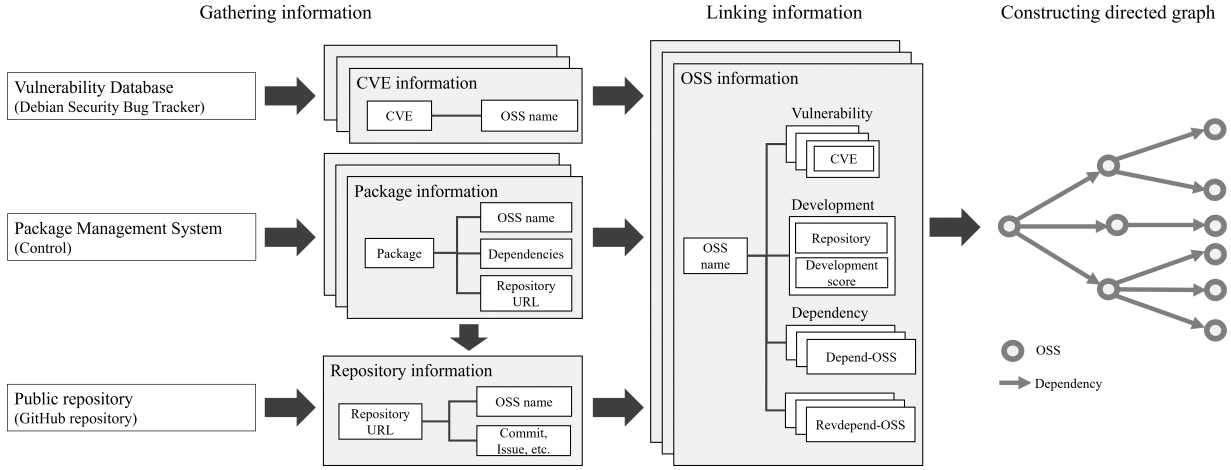


Fig. 2 Overview of the proposed method: The proposed method consists of gathering information (Section 4.4.1), linking information (Section 4.4.2), and constructing directed graph (Section 4.4.3).

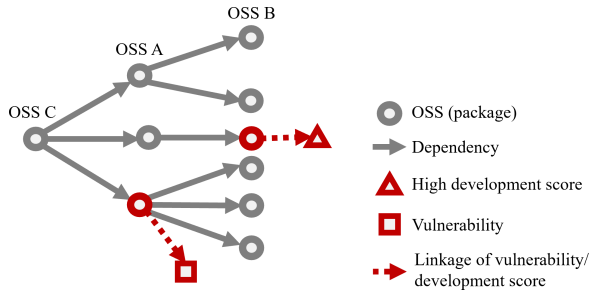


Fig. 3 The example of a directed graph (depend-OSS).

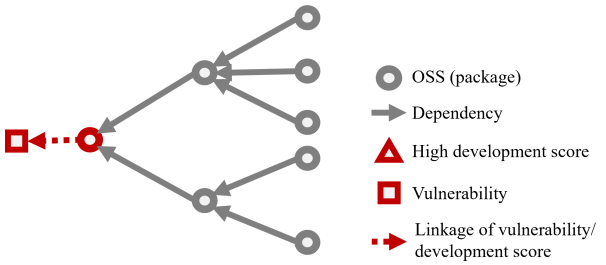


Fig. 4 The example of a directed graph (revdepend-OSS).

a high development score are also observed in the figure. The directed graph depicts not only the depend-OSS of a particular OSS, as depicted in Fig. 3, but also its revdepend-OSS with respect to vulnerabilities, as depicted in **Fig. 4**. In Scenario 1 (Section 3.2), the security administrator highlights the depend-OSS, as depicted in Fig. 3, and in Scenario 2, the revdepend-OSS is highlighted, as depicted in Fig. 4.

4.4 Proposed Method

4.4.1 Gathering Information

First, the proposed method obtains the vulnerability information from the vulnerability database (Debian Security Bug Tracker [12]), and the package information using the package management system (Control). The vulnerability information $c \in C$ contains the OSS name, $c.ossname$, and the package information contains the OSS name, $p.ossname$; dependency, $p.DEPEND$; and repository URL, $p.url$. Note that the dependency information only contains depend-OSS information, not revdepend-OSS information.

Moreover, the proposed method obtains the repository information $s \in S$ from the repository URL, $p.url$, using GitHub API and calculates the development score using repository information. The repository information, s contains the OSS name, $s.ossname$ and the development score, $s.score$.

4.4.2 Linking Information

The proposed method links all information based on OSS names. Algorithm 1 presents the proposed linkage method. First, the OSS information $o \in O$ is obtained corresponding to the OSS name, $ossname$; vulnerability information, C; package information, P; and repository information, S.

(1) Vulnerability linkage: The proposed method links all vulnerability information by searching for it and enumerating the CVE-related OSS names.

(2) Development linkage: The proposed method links the development scores by searching for repository information related to the OSS name.

Algorithm 1 Linking information based on OSS name

Input: $ossname$ ▷ Input OSS name
C ▷ Vulnerability information
P ▷ Package information
S ▷ Repository information
Output: o ▷ OSS information

```
1: o ⇐ ϕ
2: for each c ∈ C do                                  ▷ (1)
3:     if c.ossname = n then
4:         o.CVE.APPEND(c)
5:     end if
6: end for
7: for each s ∈ S do                                  ▷ (2)
8:     if s.ossname = n then
9:         o.score ⇐ s
10:    end if
11: end for
12: o.REVDEPEND ⇐ ϕ
13: for each p ∈ P do
14:    if p.ossname = n then                          ▷ (3)
15:        o.DEPEND ⇐ p.DEPEND
16:    end if
17:    for each d ∈ p.DEPEND do                       ▷ (4)
18:        if d = p.ossname then
19:            o.REVDEPEND.APPEND(d)
20:        end if
21:    end for
22: end for
```

(3) depend-OSS linkage: The proposed method links dependency information by searching for package information related to the OSS name.

(4) revdepend-OSS linkage: The proposed method links all dependency information by searching all package dependencies and enumerating the packages with related OSS names in the dependency information.

Finally, OSS $o_i \in O$ links the vulnerability, $o_i.CVE$; the development $o_i.score$; depend-OSS $o_i.DEPEND$, and revdepend-OSS $o_i.REVDEPEND$ information based on the OSS name, $o_i.ossname$.

4.4.3 Constructing the Directed Graph

Algorithm 2 describes the construction of the directed graph. Based on the input, comprising target OSS information $o_{init} \in O$, OSS list O, and development score threshold th, a directed graph G is defined using the set of vertices, V, and the set of edges, E. The directed graph aids the enumeration of depend-OSS and revdepend-OSS. The enumeration of depend-OSS is described below. The case of revdepend-OSS can also be enumerated similarly.

(1) Add the target OSS to the set of vertices.

(2) Enumerate the dependencies of the target OSS, add the enumerated OSS to the set of vertices, and add each dependency to the set of edges.

(3) For each new depend-OSS, enumerate the dependencies of the OSS. If the enumerated OSS has not yet been included in the set of vertices, add the enumerated OSS to the set of vertices and add the dependency to the set of edges.

(4) Repeat step (3) until no additional OSS is added to the set of vertices.

(5) For each OSS stored in the set of vertices, corresponding CVEs are added to the set of vertices, and the association

Table 4 List of installed package.

Category	Package name
essential	build-essential, openssh-server, openssl, ca-certificates, curl, gnupg, lsb-release
programming language	python3, python3-pip, golang, ruby-full, default-jdk, rubygems, php-common, libapache2-mod-php, php-cli
build linux kernel	libncurses-dev, bison, flex, openssl, libssl-dev, libelf-dev, git, bc
docker	docker-ce, docker-ce-cli, containerd.io, docker-compose-plugin, docker-compose
kubernetes	kubectl
database	mysql-common, postgresql, mongodb-org, influxdb

Algorithm 2 Constructing a directed graph

Input: o_{init} ▷ Initial OSS
O ▷ OSS information list
th ▷ Threshold of development score
Output: $G = (V, E)$

1: $V.APPEND(o_{init})$ ▷ (1)
2: **for** $o_d \in o_{init}.DEPEND$ **do** ▷ (2)
3: $V.APPEND(o_d)$
4: $E.APPEND((o_{init}, o_d))$
5: **end for**
6: $O_{prevadd} \Leftarrow o_{init}.DEPEND$
7: **repeat**
8: $O_{newadd} \Leftarrow \phi$
9: **for each** $o_{prevadd} \in O_{prevadd}$ **do** ▷ (3)
10: **for each** $o_d \in o_{prevadd}.DEPEND$ **do**
11: **if** $o_d \notin V$ **then**
12: $O_{newadd}.APPEND(o_d)$
13: $V.APPEND(o_d)$
14: $E.APPEND((o_{prevadd}, o_d))$
15: **end if**
16: **end for**
17: **end for**
18: $O_{prevadd} \Leftarrow O_{newadd}$
19: **until** $O_{newadd} = \phi$ ▷ (4)
20: **for each** $o \in V$ **do**
21: **for each** $cve \in o.CVE$ **do** ▷ (5)
22: $V.APPEND(cve)$
23: $E.APPEND((o, cve))$
24: **end for**
25: **if** $o.score \geqq th$ **then** ▷ (6)
26: $V.APPEND(score)$
27: $E.APPEND((o, o.score))$
28: **end if**
29: **end for**

with the OSS is added to the set of edges.

(6) For each OSS stored in the set of vertices, if the development score exceeds a predefined threshold, it is added to the set of vertices, and the association with the OSS is added to the set of edges.

5. Evaluation

In this section, the proposed method is implemented and evaluated. We evaluate the following terms experimentally by visualizing depend-OSS and revdepend-OSS based on the scenario described in Section 3.2.

(1) **Evaluation in terms of visualization quality:** This assesses if the considered scenarios could be understood adequately using the proposed method.

(2) **Evaluation in terms of calculation time:** This measures the processing time required by the proposed method, and evaluates its practical applicability in this regard.

5.1 Implementation and Visualization

The JSON file in Debian Security Bug Tracker is used as the source for vulnerability information, the apt-cache command for package information, and GitHub API for repository information. Information is obtained and linked, and the directed graph is constructed and visualized using Python.

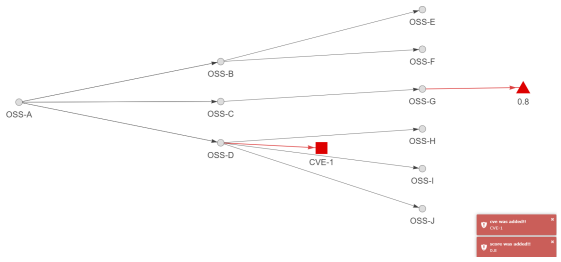


Fig. 5 The example of visualization: The visualization results are displayed in the web browser. When a new vulnerability or a development score exceeding the threshold is added, a pop-up notification appears in the lower right corner of the display, and the corresponding vertex is highlighted in red.

An example of a visualization is depicted in **Fig. 5**. The result is illustrated in web browser. The black circles represent packages, red squares represent vulnerabilities, and red triangles represent development scores. If a new vulnerability or over-threshold development score is added, a pop-up notification is displayed at the bottom-right corner of the screen, and the corresponding vertex is highlighted in red. The web browser is reloaded regularly and the results are reflected when various pieces of information are updated.

5.2 Environment

Evaluation is conducted using two virtual machines running on a single-host PC. The specifications of the host PC and virtual machines are as follows:

- Host PC: Windows 11 Pro with Intel(R) Core(TM) i7-1255U (1.70 GHz, x86-64, 10 cores), 32 GB DDR4 RAM.
- Virtual machine: Debian GNU/Linux 11 (Bullseye) with four CPU cores and 8 GB of memory. One virtual machine is used as the target computer and the other as the analysis computer.

The packages listed in **Table 4** and the dependent packages are installed on a virtual machine of the target computer. In addition to the aforementioned packages, a dummy package with a development score of 0.1 is added. The development score threshold for visualization is set to 0.6. All values of C_i and T_i are calculated using default values.

5.3 Experimental Scenarios

5.3.1 Scenario 1

This experimental scenario focuses on Python3.9 and assumes that a depend-OSS of python3.9 is visualized. First, a dummy vulnerability (CVE-YYYY-XXXX) is added to the dummy pack-

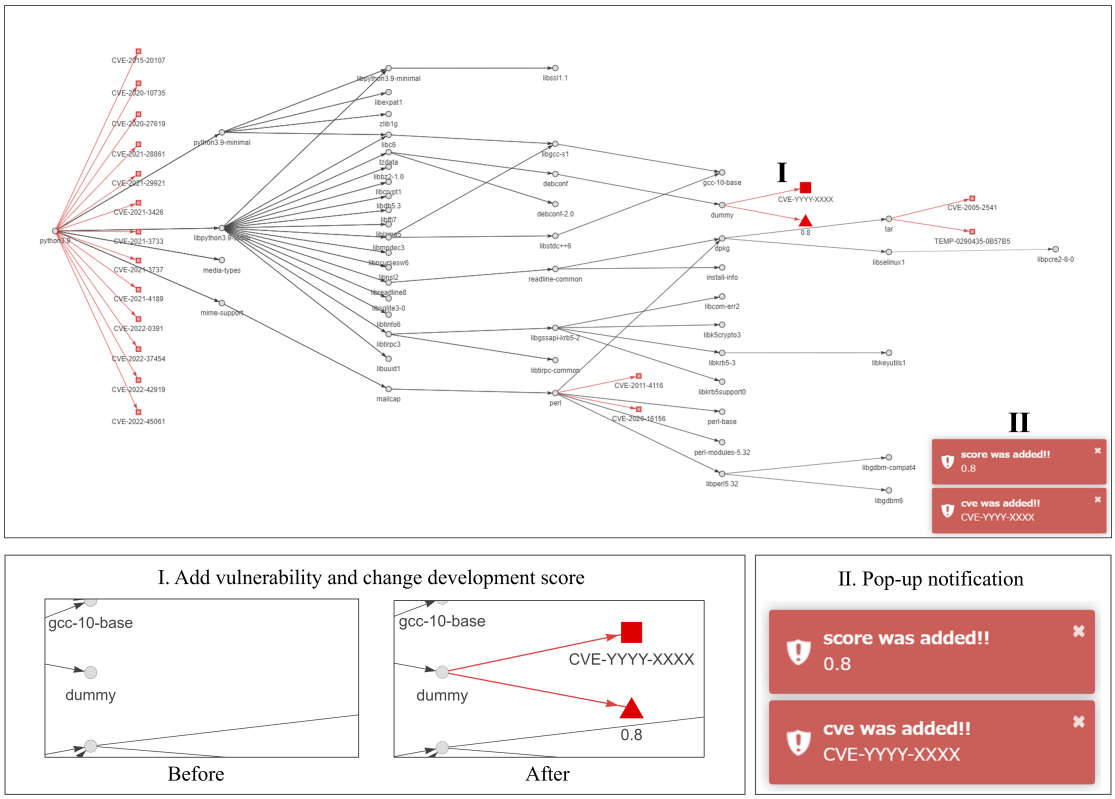


Fig. 6 Evaluation result related to scenario 1: This is the result of visualizing depend-OSS in python3.9. The figure shows when vulnerabilities and high development scores are added (I) and pop-ups appear (II).

age. This is in assumption of vulnerabilities, such as Apache Log4j CVE-2021-44228, in the dependencies of OSS. Then, the development score is changed to 0.8. The security administrator confirms the added CVE and the higher score and takes action.

5.3.2 Scenario 2

This experimental scenario assumes the visualization of revdepend-OSS that may be affected by three CVEs (refer to Section 5.4.1). The security administrator confirms the revdepend-OSS and takes appropriate action.

5.4 Evaluation

5.4.1 Evaluation 1

(1) We executed scenario 1 and confirmed that the addition of CVE and changes in scores are visualized in the visualization results using Python3.9 [23].

(2) We executed scenario 2 and confirmed the packages that may be affected by the vulnerability. We used CVE-2022-42919 [20] which is a vulnerability of Python, CVE-2022-3602 [21] which is a vulnerability of OpenSSL, and CVE-2023-38545 [22] which is a vulnerability of Curl.

5.4.2 Evaluation 2

We executed and the execution times were measured for the following tasks:

(1) **Obtain package information:** Time to obtain package information installed on the target computer.

(2) **Obtain vulnerability information:** Time to obtain all vulnerability information stored in Debian Security Bug Tracker in JSON format.

(3) **Obtain repository information:** Time to obtain repository information for packages with identified repository URL installed on the target computer.

(4) **Construct directed graph:** Time to construct directed graph based on the package, vulnerability and repository information.

(5) **Visualization:** Time to draw the created directed graph.

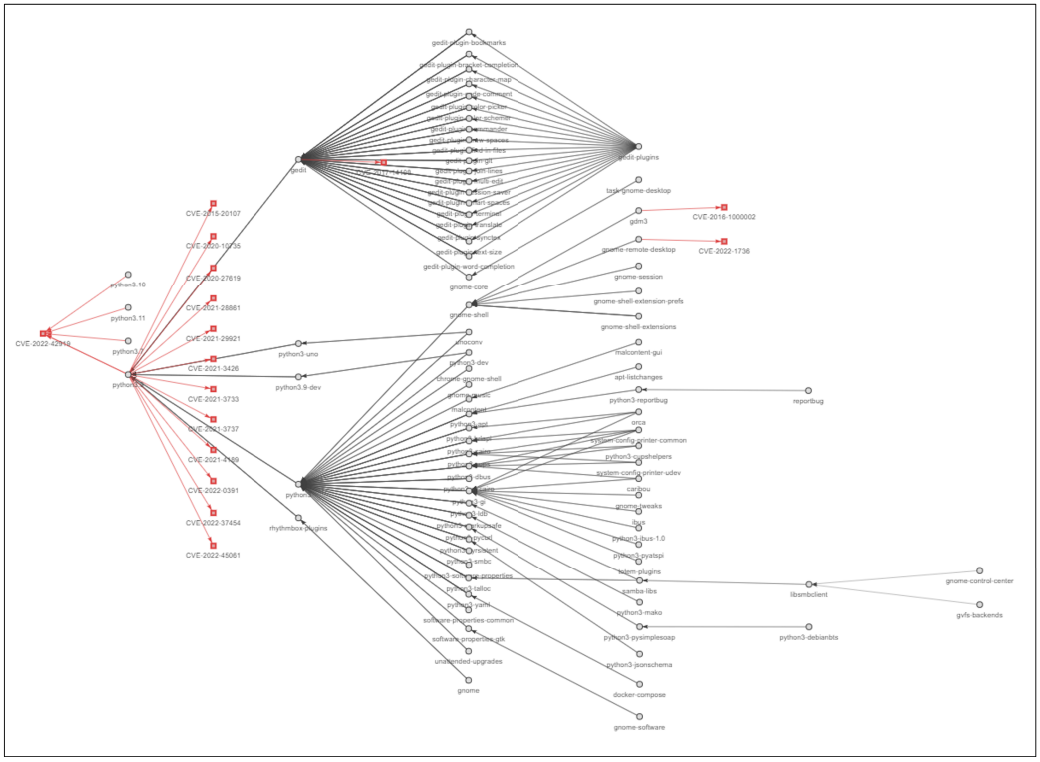
The "Construct directed graph" and "Visualization" items were measured to assess the depend-OSS and revdepend-OSS of Python3.9 [23], OpenSSL 1.1.1n [24], and Curl 7.74.0 [25].

5.5 Result

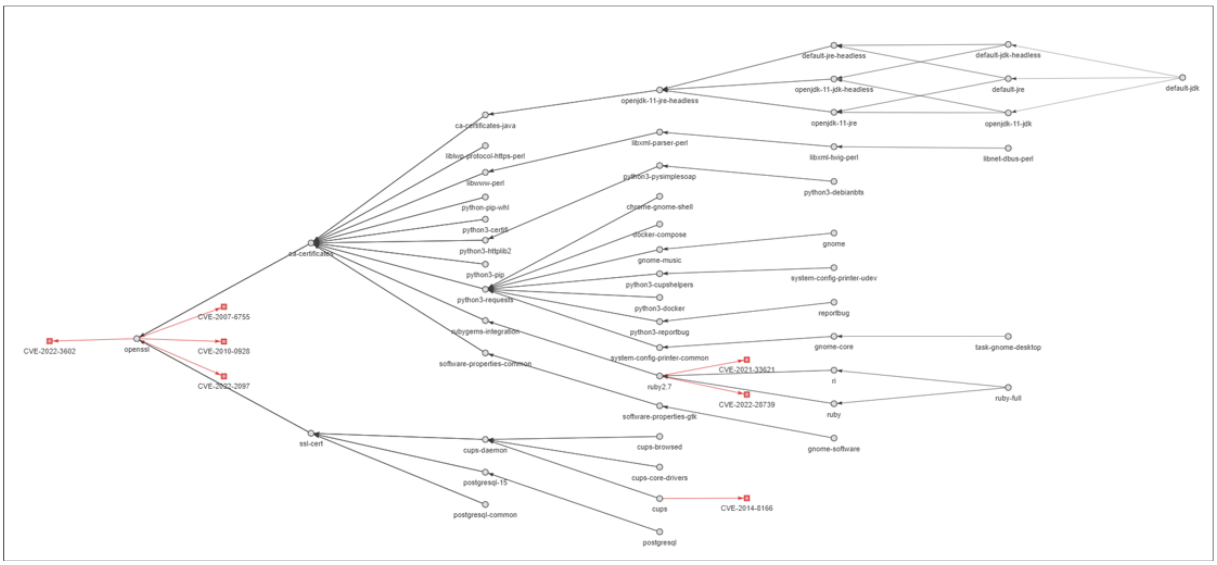
Evaluation 1: The visualizations for the result of executing scenario 1 for Python3.9 before and after the addition of vulnerabilities and development score changes are depicted in **Fig. 6**, alongside the OSS dependencies, vulnerabilities, and development score. The differences between the results are presented in the lower left of Fig. 6. The results before adding vulnerabilities and development score changes are presented on the left, and those after the changes are presented on the right. Additionally, there are two pop-up notifications about vulnerability and development score as shown in the lower right of Fig. 6.

The visualization for the results of executing scenario 2 for CVE-2022-42919, CVE-2022-3602, and CVE-2023-38545 is depicted in **Fig. 7**. As in Fig. 6, the OSS dependencies, vulnerabilities, and development scores are shown. However, it is visualized the revdepend-OSS.

Evaluation 2: The processing time of the proposed method to



(a) CVE-2022-42919(Python)



(b) CVE-2022-3602(OpenSSL)

(c) CVE-2023-38545(Curl)

Fig. 7 Experimental results for scenario 2: These are the results of visualizing the revdepend-OSS of CVE-2022-42919, which is a vulnerability of Python, CVE-2022-3602, which is a vulnerability of OpenSSL, and CVE-2023-38545, which is a vulnerability of Curl.

Table 5 Process time (Obtain information).

Contents	Process time(s)
Obtain package information	34.0
Obtain vulnerability information	23.0
Obtain repository information	11150.7

Table 6 Process time (Construction and visualization).

Contents	Package	Depend	Process time(s)
Construct directed graph	Python3.9	depend-OSS	2.36
		revdepend-OSS	3.51
	OpenSSL	depend-OSS	0.43
		revdepend-OSS	2.12
	Curl	depend-OSS	1.64
		revdepend-OSS	0.33
Visualization	Python3.9	depend-OSS	0.34
		revdepend-OSS	0.55
	OpenSSL	depend-OSS	0.22
		revdepend-OSS	0.30
	Curl	depend-OSS	0.27
		revdepend-OSS	0.21

acquire and visualize the information is presented in **Table 5** and **Table 6**. According to Table 5 and Table 6, most contents were completed in a matter of seconds. On the other hand, obtaining repository information requires as much as 3 hours.

6. Discussion

6.1 Visualization Results

Figure 6 shows that it is possible to inform the security administrators based on color changes and pop-up notifications when vulnerabilities are added or when the development scores are changed. The results are deemed to be adequate for security administrators to identify vulnerability and developmental risks of OSS, enabling them to avoid using OSS with high risk scores or adopt countermeasures against newly added vulnerabilities.

Figure 7 shows that it is possible to show security administrators the OSS that are potentially affected with the vulnerability by displaying revdepend-OSS. For example, Fig. 7 (b) shows that CVE-2022-3602, which is a vulnerability in OpenSSL, affects many OSS, including Python, OpenJDK, Ruby, and PostgreSQL. Therefore, it is necessary to take immediate action such as updating the vulnerable OSS. On the other hand, Fig. 7 (a) shows that CVE-2022-42919, which is a vulnerability in Python affects almost only packages related to python, and Fig. 7 (c) shows that CVE-2023-38545, which is a vulnerability in Curl, affects only InfluxDB. Therefore, when Python related packages in the case of CVE-2022-42919 and Curl and InfluxDB in the case of CVE-2023-38545 are not used in critical systems, immediate action is not necessary. The results are deemed to be adequate for security administrators to measure the impact of the vulnerability on the organization and to determine the vulnerability response, such as responding immediately or at the next scheduled maintenance.

6.2 Processing Time Consideration

By Table 5, the acquisition of repository information requires more time than the acquisition of package information, vulnerability information, construction of directed graphs, and visualization. This may be attributed to the limitations of the GitHub API in obtaining various types of information about the repository. However, the risk score does not change significantly within a day, and information is retrieved only once a week. Therefore, this approach is considered practical.

6.3 Emulation of the Time Required for a Security Administrator to Execute a Scenario

The following procedure is required to execute Scenarios 1 and 2 without the proposed method.

Scenario 1

(1) Extract the depend-OSS of the package using apt-cache command.
(2) Execute the script [14] to calculate the development score for the extracted depend-OSS.
(3) Check for vulnerabilities in the extracted depend-OSS by using websites.
(4) Examine dependencies of the extracted depend-OSS.
(5) Repeat steps 1–4 until there are no more packages to be investigated.

Scenario 2

Revdepend-OSS of the package is extracted using apt-cache command. However, apt-cache command does not include the dependencies of the package. Therefore, it is necessary to extract the depend-OSS information for all installed packages.

In this case, the time S_{c1} and S_{c2} required to execute each scenario are Eq. (2) and Eq. (3).

$$S_{c1} = (s_d + s_s + s_v)N \tag{2}$$

$$S_{c2} = s_d NP \tag{3}$$

Where s_d is the time to enumerate depend-OSS by executing apt-cache command, s_s is the time to calculate development score, s_v is the time to check vulnerability of packages, N is the number of packages to extract in the whole process, and P is the number of packages installed on the target computer.

In contrast, the proposed method can execute scenarios simply by accessing it with a browser.

6.4 Portability Limitations

6.4.1 Supported OS

The proposed method only supports Debian due to the use of Debian Security Bug Tracker. To support other OSs, the following methods are required:

(1) A method to link the package name of package information with the package name of vulnerability information.
(2) If the OS manages its own version of the package, a method to link the version of the vulnerable package to the version of the original package managed by the OS.

If the OS does not manage its own version control, NVD can be used for CVE information. However, if NVD is used, the OSS name in the vulnerability information obtained from the NVD may not agree with the OSS information, such as package information. This problem can be resolved by using Common Platform Enumeration (CPE). On the other hand, if the OS has its own version control, it is necessary for each OS to have information linking the vulnerabilities to the version of the target package, similar to the information provided by the Debian Security Bug Tracker.

6.4.2 Identification of Repository URL

The proposed method obtains the repository URL from the

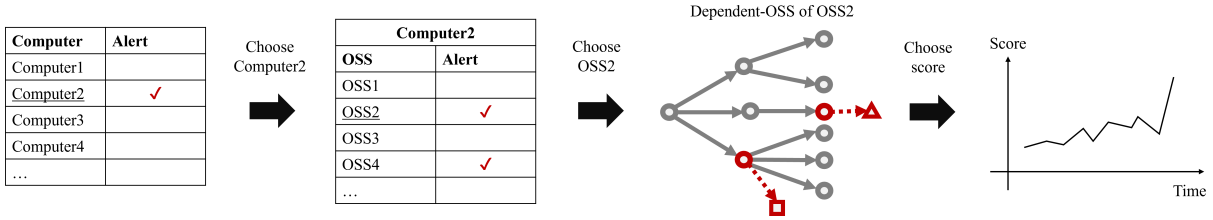


Fig. 8 The example of indication flow for multiple computers.

Homepage field of Control. However, not all packages contain repository information in Control. Therefore, in this case, it is necessary to estimate the repository from other information. Further, some OSS projects do not use the GitHub repository. Thus, a suitable method is required to obtain information about such OSS projects.

6.4.3 Obtaining Dependencies

The dependencies included in the control can be obtained only based on the dependencies between the deb packages. Therefore, dependencies, such as the libraries included in the building process, must also be obtained. In future work, we intend to consider obtaining dependencies from other sources, such as software bills of materials (SBOM) [26].

6.5 Capability Limitations

6.5.1 Number of Target Computers

In this paper, we evaluated a single target computer. However, in practice, organizations involve multiple computers —thus, an appropriate visualization method is necessary. Therefore, as depicted in **Fig. 8**, computers and their alerts should be displayed in the form of table, and security administrators may choose to display directed graphs of different computers. The computational complexity of this process is $O(N)$ because each computer constructs the graph independently.

6.5.2 The Capability of Real-time Processing

The proposed method can visualize the collected information in a few seconds or tens of seconds. Although it takes time to collect repository information, this is not an issue currently because the development score does not change significantly over this time period. However, a calculation method is used for the development score that causes significant score fluctuations in a short time, this issue could become problematic. Therefore, an appropriate method is required to minimize the time taken for the collection of repository information.

6.5.3 Capability of Finding Zero-day Vulnerabilities

The proposed method can also apply to zero-day vulnerabilities after a CVE has been issued. Zero-day vulnerabilities are displayed in the visualization results in the same manner as other CVE-issued vulnerabilities. However, if the visualization results can include the information that security patches have not yet been released, security administrators can determine the priority of response.

6.6 Visualization Limitations

6.6.1 Displayed OSS within Dependencies

The proposed method displays all OSS within dependencies. However, some OSS may have a large number of dependencies. Therefore, they should be visualized by displaying only the portions containing the alerts.

6.6.2 The Validity of the Visualization Format

The proposed method uses a directed graph for the purpose of visualization. This is considered to be a suitable format for displaying dependencies. On the other hand, it is not sufficient for visualization of time-series changes, such as that of the development score. In such cases, other formats should be used for visualization. For example, as illustrated in Fig. 8, functions display fluctuations in the development score in time series by clicking on the development score in the directed graph.

6.7 Evaluation of Operational Efficiency

The user study and quantitative evaluation are important to visualization approach. These evaluations of the proposed method can indicate improvements of the operational cost of a security incident response for each OSS package's vulnerability. Moreover, these evaluations of the proposed method with related works and human analysis can highlight the efficiency of the proposed method.

To understand the feasibility of the proposed method, user studies should be conducted to compare the efficiency of executing each scenario with and without the proposed method on multiple users. Evaluation via user studies remains a challenge.

7. Related Work

Vulnerability information: Several types of studies have been conducted on vulnerabilities, e.g., on the analysis [4] and visualization [27] of vulnerability information stored on databases. Additionally, determining the severity of vulnerabilities is important to adopt appropriate vulnerability responses. The Common Vulnerability Scoring System (CVSS) [28] has been widely used as an indicator of vulnerability severity [29]. Improved scoring methods [5] and frameworks that include decision-making [30] have also been proposed. Further, studies have been conducted to estimate the priority of vulnerability responses [6].

Developmental information: Information about repositories is essential to address the security risks of OSS. However, in general, evaluation indicators for OSS repositories are related to the maturity [31] and importance [14] of OSS, and are used for their selection and evaluation. Some evaluation indicators related to security risk have been proposed, e.g., the OSS Scorecard [7], which is calculated in terms of software security from the perspective of OSS repositories. In addition, a study was conducted on the potential risks of OSS vulnerable to supply chain attacks on npm packages [32]. We intend to consider this in the scoring index in future works.

Table 7 Comparison with related work.
✓: Supported

	Vulnerability risks	Development risks	Dependency
EPSS [5]	✓		
OpenSSF Scorecard [7]		✓	
Dependency-Check [33]	✓		✓
Our proposed method	✓	✓	✓

Dependency information: Several tools are available for analyzing software dependencies [8]. Additionally, tools such as dependency-check [33] and Snyk [34] are available to analyze vulnerabilities in dependencies and improvement methods [35]. Further, vulnerabilities in dependencies have been analyzed on a large scale [36]. In the proposed method, these dependencies are described using package management systems.

7.1 Comparison with Related Work

Table 7 compares the related works, [5], [7], and [33]. The Exploit Prediction Scoring System (EPSS) [5] was designed to evaluate and predict the likelihood of vulnerabilities being exploited. It assigns scores based on factors such as severity of vulnerabilities, the availability of the exploiting code, and the ease of exploitation. OpenSSF Scorecard [7] is a framework for assessing the security of open-source projects. It is based on elements such as security best practices, secure development, risk management, community participation, security documentation, and software composition analysis. A dependency check [33] is a software tool that assists the identification of vulnerabilities in the dependencies of applications or projects. It scans dependencies, such as libraries and frameworks, and compares them to a vulnerability database.

However, EPSS, OpenSSF Scorecard, and the dependency check do not cover all aspects of vulnerability risks, developmental risks, and dependencies. Each factor of OSS security risk was individually considered, and the main focus was on the accuracy of each factor. Although individual indicators of vulnerability, development status, and dependencies can be obtained, it is difficult to automatically link them due to differences in OSS names and versions. While the proposed method is limited to Debian, all information is linked based on data managed by the Debian project.

The novelty of the proposed method is that it enables the understanding of vulnerability risk and developmental risk, considering their dependencies, which have been separately considered in related work, and enables rapid and wider understanding of the security risk of OSS. Additionally, we evaluated validity through visualization based on scenarios.

On the other hand, it is difficult to compare the cost of analysis times because each method differs in terms of the range of support and format of the output, such as numerical values and visualizations. The computational complexity of the OSS analysis should be calculated to promote fair comparisons. EPSS is a score mainly related to software vulnerability risk, OpenSSF Scorecard is a score related to OSS repositories, Dependency-Check is the result of an analysis of software dependencies and vulnerabilities, and the proposed method is a visualization of vulnerability and developmental risks, considering their dependencies on OSS. Since the output of each method is different, it is necessary to calculate the time complexity of each output for each software. In this case, the time complexity of all methods is $O(N)$. However, as shown in Section 6.3, the proposed method is capable of processing within a practical time.

8. Conclusion

In this paper, we proposed a method to link and visualize vulnerability, developmental, and dependency information of OSS. The proposed method extracts and links vulnerability, developmental status, and dependency information of OSS based on vulnerability databases, GitHub repositories, and package management systems. Finally, it visualizes them using directed graphs. This enables security administrators to check the visualization results, identify OSS with security risks, and formulate appropriate countermeasures. In future works, research should be conducted on obtaining dependencies from SBOM, supporting for other OS, and improving the method in terms of calculating scores that represent development status.

Acknowledgments This work was partially supported by the Japan Society for the Promotion of Science (JSPS) KAKENHI Grant Number JP23K16882, and ROIS NII Open Collaborative Research 2022 (22S0302)/2023 (23S0301).

References

[1] Yano, T. and Kuzuno, H.: Security Risk Visualization for Open-Source Software based on Vulnerabilities, Repositories, and Dependencies, *The 11th International Symposium on Computing and Networking* (*CANDAR 2023*) (2023).

[2] Synopsys, Inc.: Open source Security and Risk Analysis Report (2023).

[3] National Institute of Standards and Technology: CVE-2021-44228 (online), available from ⟨https://nvd.nist.gov/vuln/detail/CVE-2021-44228⟩ (accessed 2023-12-04).

[4] Williams, M.A., Dey, S., Barranco, R.C., Naim, S.M., Hossain, M.S. and Akbar, M.: Analyzing evolving trends of vulnerabilities in national vulnerability database, *2018 IEEE International Conference on Big Data* (*Big Data*), pp.3011–3020 (online), DOI:10.1109/BigData.2018.8622299 (2018).

[5] Jacobs, J., Romanosky, S., Edwards, B., Adjerid, I. and Roytman, M.: Exploit prediction scoring system (EPSS), *Digital Threats: Research and Practice*, Vol.2, No.3, pp.1–17 (online), DOI:10.1145/3436242 (2021).

[6] Walkowski, M., Krakowiak, M., Jaroszewski, M., Oko, J. and Sujecki, S.: Automatic CVSS-based vulnerability prioritization and response with context information, *2021 International Conference on Software, Telecommunications and Computer Networks* (*SoftCOM*), pp.1–6 (online), DOI:10.23919/SoftCOM52868.2021.9559094 (2021).

[7] Open Source Security Foundation (OpenSSF): Security Scorecards (online), available from ⟨https://github.com/ossf/scorecard⟩ (accessed 2023-12-04).

[8] Trail of Bits: It-Depends (online), available from ⟨https://github.com/trailofbits/it-depends⟩ (accessed 2023-12-04).

[9] de Zafra, D.E., Pitcher, S.I., Tressler, J.D. and Ippolito, J.B.: Information technology security training requirements: A role-and performance-based model, *NIST Special Publication*, vol.800, No.16, pp.800–816 (online), DOI:10.6028/NIST.SP.800-16 (1998).

[10] Mitre Corporation: CVE (online), available from ⟨https://cve.mitre.org/⟩ (accessed 2023-12-04).
[11] National Institute of Standards and Technology: NATIONAL VULNERABILITY DATABASE (online), available from ⟨https://nvd.nist.gov/⟩ (accessed 2023-12-04).
[12] Debian Project: Security Bug Tracker (online), available from ⟨https://security-tracker.debian.org/tracker/⟩ (accessed 2023-12-04).
[13] GitHub, Inc.: GitHub REST API (online), available from ⟨https://docs.github.com/rest⟩ (accessed 2023-12-04).
[14] The Open Source Security Foundation: Open Source Project Criticality Score (Beta) (online), available from ⟨https://github.com/ossf/criticality_score⟩ (accessed 2023-12-04).
[15] Python Software Foundation: PyPI - The Python Package Index (online), available from ⟨https://pypi.org/⟩ (accessed 2023-12-04).
[16] rubygems.org: RubyGems.org — your community gem host (online), available from ⟨https://rubygems.org/⟩ (accessed 2023-12-04).
[17] dpkg.org: Dpkg — Debian Package Manager (online), available from ⟨https://www.dpkg.org/⟩ (accessed 2023-12-04).
[18] RPM.ORG: RPM Package Manager (online), available from ⟨https://rpm.org/⟩ (accessed 2023-12-04).
[19] Debian Project: 5. Control files and their fields (online), available from ⟨https://www.debian.org/doc/debian-policy/ch-controlfields.html⟩ (accessed 2023-12-04).
[20] National Institute of Standards and Technology: CVE-2022-42919 (online), available from ⟨https://nvd.nist.gov/vuln/detail/CVE-2022-42919⟩ (accessed 2024-04-01).
[21] National Institute of Standards and Technology: CVE-2022-3602 (online), available from ⟨https://nvd.nist.gov/vuln/detail/CVE-2022-3602⟩ (accessed 2024-04-01).
[22] National Institute of Standards and Technology: CVE-2023-38545 (online), available from ⟨https://nvd.nist.gov/vuln/detail/CVE-2023-38545⟩ (accessed 2024-04-01).
[23] Debian Project: Package: python3.9 (online), available from ⟨https://packages.debian.org/en/bullseye/python3.9⟩ (accessed 2024-04-01).
[24] Debian Project: Package: openssl (online), available from ⟨https://packages.debian.org/en/bullseye/openssl⟩ (accessed 2024-04-01).
[25] Debian Project: Package: curl (online), available from ⟨https://packages.debian.org/en/bullseye/curl⟩ (accessed 2024-04-01).
[26] National Telecommunications and Information Administration: Software Bill of Materials (online), available from ⟨https://www.ntia.gov/sbom⟩ (accessed 2023-12-04).
[27] Pham, V. and Dang, T.: Cvexplorer: Multidimensional visualization for common vulnerabilities and exposures, *2018 IEEE International Conference on Big Data* (*Big Data*), pp.1296–1301 (online), DOI:10.1109/BigData.2018.8622092 (2018).
[28] Mell, P., Scarfone, K. and Romanosky, S.: Common vulnerability scoring system, *IEEE Security & Privacy*, Vol.4, No.6, pp.85–89 (online), DOI:10.1109/MSP.2006.145 (2006).
[29] Johnson, P., Lagerström, R., Ekstedt, M. and Franke, U.: Can the common vulnerability scoring system be trusted? a bayesian analysis, *IEEE Trans. Dependable and Secure Computing*, Vol.15, No.6, pp.1002–1015 (online), DOI: 10.1109/TDSC.2016.2644614 (2016).
[30] Spring, J.M., Householder, A., Hatleback, E., Manion, A.: Prioritizing Vulnerability Response: A Stakeholder-Specific Vulnerability Categorization (Version 2.0), CARNEGIE-MELLON UNIV PITTSBURGH PA (2021).
[31] Lenarduzzi, V., Taibi, D., Tosi, D., Lavazza, L. and Morasca, S.: Open source software evaluation, selection, and adoption: A systematic literature review, *2020 46th Euromicro Conference on Software Engineering and Advanced Applications* (*SEAA*), pp.437–444 (online), DOI:10.1109/SEAA51224.2020.00076 (2020).
[32] Zahan, N., Zimmermann, T., Godefroid, P., Murphy, B., Maddila, C. and Williams, L.: What are weak links in the npm supply chain?, *Proc. 44th International Conference on Software Engineering: Software Engineering in Practice*, pp.331–340 (online), DOI:10.1145/3510457.3513044, (2022).
[33] OWASP Foundation: OWASP Dependency-Check (online), available from ⟨https://owasp.org/www-project-dependency-check/⟩ (accessed 2023-12-04).
[34] Snyk Ltd.: Snyk (online), available from ⟨https://snyk.io/⟩ (accessed 2023-12-04).
[35] Ponta, S.E., Plate, H. and Sabetta, A.: Detection, assessment and mitigation of vulnerabilities in open source dependencies, *Empirical Software Engineering*, Vol.25, No.5, pp.3175–3215 (online), DOI:10.1007/s10664-020-09830-x (2020).
[36] Liu, C., Chen, S., Fan, L., Chen, B., Liu, Y. and Peng, X.: Demystifying the vulnerability propagation and its evolution via dependency trees in the npm ecosystem, *Proc. 44th International Conference on Software Engineering*, pp.672–684 (online), DOI:10.1145/3510003.3510142 (2022).

Tomohiko Yano received an M.E. degree in Informatics from the Kyoto University, Kyoto, Japan in 2015. Since joining SECOM in 2015, he has been engaged in research field of cyber security. He is now the researcher of the Cyber-Physical Systems Security Group at the SECOM Intelligent Systems Laboratory. He is a member of IEICE.

Hiroki Kuzuno received an M.E. degree in Information Science from Nara Institute of Science and Technology, Japan, in 2007, and a Ph.D. in Computer Science from Okayama University, Japan in 2020. In 2007 he became a Research Engineer at SECOM Intelligent Systems laboratory. Currently, he is an Assistant Professor at Kobe University, Japan. His research focuses on computer security specifically on operating systems and networks. He is a member of IEICE and IPSJ.