****

**APT REPORT ON**

**SECURITY GUIDELINE FOR OPEN SOURCE SOFTWARE**

**Edition: June 2021**

**The 33rd APT Standardization Program Forum (ASTAP-33)**

**7 – 15 June 2021**

**Virtual/Online Meeting**

***(Source: ASTAP-33/OUT-19)***

**No. APT/ASTAP/REPT-49**

**security guideline for Open Source Software**

1. **Introduction**

Open Source Software (OSS) has been used in industries, academia, and many other fields. However, security concerns about OSS have been issued to introduce OSS in one’s software.

This document describes how to handle security issues when using OSS in one’s software. Part I suggests a new vulnerability check model of the software when using OSS based on software resource identifier in Software Bill of Material. Part II provides guidelines to avoid OSS license violence when fixing exposed or known vulnerabilities in one’s software.

1. **Scope**

This document provides guidelines for those who develop or use software built with OSS. The purpose of this document is to support a comprehensive understanding of security issues with OSS.

The scope of this document is as follows:

* Vulnerability check model of the software when using OSS based on software resource identifier in Software Bill of Material
* Integrated software licensing and vulnerability management guide in the software development lifecycle
1. **Terms and definitions**

OSS, F/OSS:

 (Free and) Open Source Software

SOFTWARE RESOURCE IDENTIFIER:

 The identifier indicates the virtual resource file of the software. This identifier is declared in the Software Bill of Material.

Software Bill of Material (SBoM):

 *Bill of material* about software. SBoM describes the metadata of each software resource and dependent package

OSS License:

 Open Source Software license.
Almost every OSS license forces including the copyright notice (includes the name of an OSS) in all copies or substantial uses of the work. An SBoM describes this information for each Software Resource.

**Part I: Vulnerability check model of Open Source Software based on software resource identifying**

1. **Introduction**

Since there is software using Free and Open Source Software (F/OSS or OSS), the software developers and companies have to concern that their software product may have any vulnerability which is found after the release. However, the vulnerability is likely to be found afterward.

Also, OSS may be used in different software provided by numerous individuals or companies. Because each OSS is not developed by one team, they may have to subscribe to many vulnerability news sources.

Therefore, after finishing the development process and releasing the product, the software developers or companies may miss, forget, ignore, or lose their attention to check the vulnerability in their product.

To solve this situation, this document offers a novel method for software users to check vulnerability by themselves with software resource identifiers on provided SBoM. When the vulnerability is checked or recognized by the proposed method and protocol, it could ensure that the product contains the potentially vulnerable code in software even those function or code is unused or disable. This proposal can be interpreted as an emergency signal announcement and suggests options to a user before the vendor of identified software does any actions.

1. **Summary**

This model requires a system administrator for requesting SBoM from software vendors. Also, using a SOFTWARE RESOURCE IDENTIFIER on the SBoM to check vulnerabilities of their operating system and its components. When the software and its SBoM are provided to a system administrator properly, the administrator could identify known vulnerability issues which affect their software immediately and precisely.

1. **A virtual model of response model based on SOFTWARE RESOURCE IDENTIFIER**
	1. Overview
		1. Existing response model



In the existing response model, every individual software developer and company should check vulnerability information about their used OSS by themselves. This information is usually provided after the vulnerability is announced publicly as a form of CVE(Common Vulnerability and Errors). However, even though they realized the vulnerability of their developed software, no one can force the developers to inform the software vulnerability to their customers.

Moreover, when a developer compiles the software with a package manager, the developer needs to check and identify which version of the nested OSS library is used in the software. However, OSS, package managers, and modern compiler allow a different and broad version of the library for software compilation. Therefore, even if the developers select the same version of the OSS library by themselves, no one could guarantee which version of nested OSS libraries is used.

Understandably, this existing model may require source code analysis to identify vulnerability. It means that the software user is usually not allowed to check the vulnerability by themselves. Also, this situation comes from that almost every user is not a direct user of OSS because the user is insensitive to the OSS vulnerability issues.

* + 1. Response model in this proposal

In this proposal, the vulnerability information provided by security researchers and companies is broadcasted to a user directly. The transmitter is not required to recognize what kind of software the receivers have.

In this method, a software user(customer) should register SOFTWARE RESOURCE IDENTIFIER information about their using software as a form of the Bill of Material provided by the developer or vendor in the one’s receiver.

* + - 1. Summary of flow

Transmitter side: security analyst, researcher

* Vulnerability found
* Vulnerability mapping (Vulnerability DB ↔ SOFTWARE RESOURCE IDENTIFIER DB)
* Mapping with public/private vulnerability DB and unique SOFTWARE RESOURCE IDENTIFIER
* Transmitting SOFTWARE RESOURCE IDENTIFIER (& Vulnerability information) signal

Receiver side: Software user (administrator)

* Before receiving
	+ Register SOFTWARE RESOURCE IDENTIFIER of each software
	+ Setup Policies for each situation or vulnerability Score
* Receiving a signal (SOFTWARE RESOURCE IDENTIFIER)
* Detecting Vulnerable component based on a received signal
* Informing vulnerability to manager/administrator
* (Responding to vulnerable components like stopping service/blocking it/doing nothing)
	+ - 1. Request software package information

At first, a software user requests the SOFTWARE RESOURCE IDENTIFIER data. For example, the Software Bill of Material could contain the SOFTWARE RESOURCE IDENTIFIER and many other metadata that may be a clue to identify the vulnerable software resource. This information covers the essential features for checking vulnerability. For example, using multiple hash values and filename could help to avoid a SOFTWARE RESOURCE IDENTIFIER collision.

* + - 1. Mapping SOFTWARE RESOURCE IDENTIFIER to known vulnerability DB

Once all the necessary vulnerability information and affected SOFTWARE RESOURCE IDENTIFIER information of OSS is given, information mapping can be implemented. Vulnerability DB would try to match with the SOFTWARE RESOURCE IDENTIFIER DB, which contains the unique identifier of the software resource instead of its source code itself. These features have an advantage against efficiency and trade secret protection. In the past, checking vulnerability is comparing source code and the vulnerability DB or analyzing its properties, and it takes a very long time to complete the process. However, matching vulnerable OSS source code and its SOFTWARE RESOURCE IDENTIFIER is not a delicate nor heavy task when the vulnerability is recognized already by security researchers. Mapping can be done with both public and private vulnerability DB. Before this process can be done, registration of SOFTWARE RESOURCE IDENTIFIER should be done for all software by its developers. They are responsible for setting up policies for every situation.

* + - 1. Emergency vulnerability alert announces to the software end-user

Responder(software end-user/system administrator) receives vulnerability signal and inquires to registered SOFTWARE RESOURCE IDENTIFIER to detect potentially affected software by the received vulnerability.

After checking the vulnerability in the product, a receiver could generate a report automatically. The report contains the vulnerability and potentially affected software and could explain the details of the vulnerability to software users directly. The users could read the report in detail and take actions like stopping service, blocking it, or no behavior.

Since informed vulnerability may not be found or neglected by the software developer or vendor, some security problems would be caused. Therefore, an emergency alert announces the software user and suggests that they shut down or stop the software to avoid hackers or some cyber-attack before the developer updates to fix the vulnerability.

* + - 1. Inform to the software developer

At the same time, vulnerability detail and SOFTWARE RESOURCE IDENTIFIER data would also be transmitted to who is responsible for fixing or deal with the vulnerability. The vulnerability information can follow not only the form of the Software Bill of Material but also another format if needed, like CYBEX.

1. **Difference between the proposal and existing standards**
	1. CYBEX (ITU.T Recommendation X.1500 series)

This standard is for exchanging cybersecurity information between different stakeholders like researchers, software developers/vendors, and service providers/administrators for security research and fixing. However, in this standard, vulnerability information is provided by a security researcher/organization to an administrator or a user just for alerting potential vulnerability.

* + 1. SCAP (Security Content Automation Protocol)

SCAP is one of the similar standards which shares the identical purpose with the proposal. Moreover, this covers more vulnerability handling areas than the proposal. However, SCAP and this proposal have different stances about how to provide the information. SCAP finds vulnerability confirmed by its developer, but the proposal is standing as any other trustable 3rd party could provide an emergency signal.

The proposal (because vulnerability identifying process is done on end-user’s asset) once the information mapping process finishes because this mapping process does not use string searching or static/dynamic security analysis but uses only single or multiple values for each affected software resource to identify vulnerability, the result is issued at most immediately. SCAP also have some identifying method when the software follows defined rules by SCAP, but the rule is not generic enough to apply to every software. Therefore, end-users could not know security integrity in real-time with SCAP.

More specifically, the process of the proposal can be sure that the software includes vulnerable code in the software. However, some of the vulnerable code may not affect when using the software because it is not enabled, reachable, nor used.

* 1. Vulnerability handling processes (ISO/IEC 30111) & Vulnerability disclosure (ISO/IEC 29147)

These two standards provide guidelines on vulnerability handling and disclosure. Based on these two standards, software should be confirmed that there is an attraction of the identified vulnerability. Also, it should try to lower the risk as much as it can. These similar concepts are applied in the suggesting model. When a vulnerability is checked, an alert and temporary suggestion would be provided to a user or infected computer, which can minimize the risk from the vulnerability. Also, the information of the vulnerability would disclose to the developer and user. Besides that, ISO/IEC 30111 also provides guidelines for a vendor to handle the vulnerability to solve the problem. In the suggesting model, a report of vulnerability is sent to the developer of the product, and they should take further actions. Before the update is published, end-users could be advised to stop using the software.

* 1. A vendor-neutral framework for automatic notification of security related information and dissemination of updates (ITU-T Recommendation X.1206)

This standard and the proposal have very similar structures and concepts: Security related information is automatically updated to subscribers(end-users). The standard was provided in April 2008. At that time, the OSS software and its development/compilation process were not familiar to every software vendor. The unique feature of the OSS software couldn’t be reflected in the standard.

There are two main differences: 1. This standard manages the vulnerability by software name and version information method, and 2. (Preemptive) potential security information reporting process, which shares the same issue with CYBEX/SCAP.

The standard identifies the security related information by affected software’s *Version\_Info*, consisting of product name, product version, etc. However, in the development environment in this day, only the software version could not describe the version information of internal/nested (OSS) libraries. This situation leads that the end-user/developer could not realize that the vulnerability is affected in their software (binary).

4.4 SPDX Specification (ISO/IEC 5962) and OpenChain (ISO/IEC 5230:2020)

SPDX(Software Package Data eXchange) is the developing international standard data format of the Software Bill of Material.

SPDX data format has a vulnerability information field as an option. However, the area is not for post vulnerability checks but for providing information during development or releasing the software product.
Also, the SPDX data format has a filename field and optional multiple file hash fields. These fields could be treated as SOFTWARE RESOURCE IDENTIFIER of the proposal.

OpenChain is the international standard for open source license compliance which supports the software supply chain management process. The standards started with SPDX, and it diverged from SPDX to focus on software supply chain management.

These standards provide the information and guidelines about how to provide a Software Bill of materials in the software supply chains but lack in how to handle vulnerability found after the release of the software.

**Part II: Open Source Software License Compliance guide when using OSS in the software product**

1. **Introduction**

Since there exists software using Free and Open Source Software (F/OSS or OSS) and the software developers and companies may miss, forget, ignore, or lose their attention to check the vulnerability in their software. An OSS is used in different software provided by different individuals or companies. Because OSS discloses its source code, anyone could find the vulnerability, and the vulnerability information could be shared publicly.

However, when one can fix the vulnerability by oneself, one must be concerned about the OSS license’s obligation not to violate them. For example, the OSS licenses may force developers or vendors to disclose their software source code when fixing vulnerability was not done with proper step.

This document describes how to fix OSS vulnerability in software with not violating the licenses.

These days, software vulnerability should have corresponded immediately. Many developers and organizations involved vulnerability checking and patching process in their software development lifecycle. This kind of approach becomes common.

However, when using OSS in the product, developers or vendors have to concern about their development process.

OSS is mainly provided with an OSS license.

When using OSS for its software product, the user and programmer must obligate the clauses of the OSS license.

However, some OSS licenses have some strict clauses. For example, viral licenses such as GPL 2.0 and 3.0 require source code disclosing of entire software when linking or including the source code of the software in the software. Some of the licenses, such as the LGPL family, allow just linkto the software without source code disclose, but when including the source code on the software directly, the source code would be disclosed. Therefore, how and when the hotfix is done to the software is very important with OSS licenses.

Unfortunately, this guideline cannot give credible and reliable advice about OSS and its license because the interpretations of many OSS licenses are not clear. Also, there are not so many ruled cases about OSS license involved cases. Moreover, many cases depend on de jure, especially on the clauses of copyright for each de jure, etc. Therefore, this guideline is very narrow and limited for applying developer, vendor, or user region and cases.

However, applying a patch to fix a vulnerability should be done immediately. Hesitating to fix vulnerability because of license violence may not the excuse for the customers.

1. **Avoid using OSS which provided under viral licenses or introducing an OSS component tracking system into the software development process**

Avoid using OSS, which provided under viral licenses security-critical software when wanting to keep the source code of software closed without careful OSS license management and software development management.

The range of source code disclose caused by source code editing can be various for each OSS license and how OSS in the software product are introduced. This range depends on when the source code is edited, how the OSS is linked to their software, and many other factors based on the OSS license clauses.

If a developer or vendor wants to keep the software closed, the software development process should be designed by avoiding OSS license violations with their licenses. Design the process to manage licenses carefully. Then the monitor could find where the hotfix is applied and which components links with the applied component. Many of the current software development processes still do not monitor this kind of form.

When the developer or vendor cannot improve the software development process, OSS should be excluded from the products.

If the product needs immediate corresponding to a security problem, this kind of monitoring should be done automatically. (ex: Checking the linking form, its relationships, and its license interpretation)

1. **OSS isolating design and reordering hotfix patching step**

When the developer or vendor cannot avoid using viral licensed OSS, 1) design the software to isolate from OSS component and the software carefully, or 2) Make sure patching step is not violated disclosure clauses.

1. When the viral licensed OSS component is isolated from the software, many cases do not violate the licenses.

2. However, when the OSS component cannot be isolated from the software (such as embedded software, etc.), the order of applying the hotfix should make sure.

When following the proper step to apply the hotfix, it may not lead to license violation without disclosing the whole of software under many of the OSS licenses, but not every license.

1. **Check the nested library’s detailed version**

When a developer or vendor compiles the software with a package manager, one needs to check and identify which version and license of the nested OSS library are used.

OSS and modern compiler and package managers now allow a different and broad version of the library for software compilation. For example, when a nested library was updated because of a vulnerability issue, the version of the outer library may not be changed because the version is mainly numbered for compatibility but not for vulnerability. Therefore, even indicating a specific OSS library by its name and its version; the software may consist of different versions of the OSS library. This issue is not only about vulnerability issues but also about its license in rare cases. To solve this issue, checking vulnerability about the nested library by libraries’ name and version may not enough. Software Resource Identifier which is provided by SBoM should be checked for vulnerability validation.

Some of the packages may include unlisted OSS software in the package description, like assembly/C/C++ embedded code. Therefore, be careful when wanting to keep the software code disclosed.

**Bibliography**

* ISO/IEC 5962, *Information Technology — SPDX® Specification V2.2.1*
* ISO/IEC 5230:2020, *Information technology — OpenChain Specificati*on
* ITU-T Recommendation X.1500 (04/11), *Overview of cybersecurity information exchange*
* ITU-T Recommendation X.1206 (04/08), *A vendor-neutral framework for automatic notification of security related information and dissemination of updates*
* ISO/IEC 30111:2019, *Information technology — Security techniques — Vulnerability handling processes*
* ISO/IEC 29147:2018, *Information technology — Security techniques — Vulnerability disclosure*