



Open Source Software: The Ultimate in Reuse or a Risk Not Worth Taking?

Nancy R. Mead¹, Carnegie Mellon University

Carol Woody² and **Scott Hissam**, Software Engineering Institute

Open source software components are widely used to improve software development cost and schedule. However, in order to have confidence in open source software, more work is needed to adequately measure the cybersecurity risks associated with use of open software components. This article discusses the measurement challenges.

Digital Object Identifier 10.1109/MC.2024.3423908
Date of current version: 29 January 2025

For decades, software developers and managers have tried to find ways to make their jobs easier. One idea was to share small programs that produced commonplace functionality, whether commonplace math functions, sort routines, reuse on a small scale, and even more ambitious goals for reuse on a large scale. Sometimes, software was developed with reuse in mind but without an understanding of who would be using it. As a consequence, it often takes more effort to do the reuse than to do new development!

With the massive software systems being developed today, it certainly seems like open source software provides a convenient and inexpensive answer. Pieces of open source software are reviewed both by teams of developers and by many users. When it appears that a new update is available, open source software is often automatically downloaded and used.

From a cybersecurity perspective, it was argued that open source



FROM THE EDITOR

Welcome back! In this month's instance of the "Open Source" column, Nancy Mead and colleagues from Carnegie Mellon University's Software Engineering Institute review open source from a security perspective. Security, by now, is a recurring topic in this column for its importance in an ever more complex world of software. In this article, Mead et al.'s vantage point is unique in that their system-of-systems and large contracts perspective is different from the fast-paced and sometimes haphazard world of commercial development. With that, happy reading and keep on hacking!—Dirk Riehle

was *more* secure because the source code could be inspected so readily. However, cybersecurity threats in open source can be obfuscated by malicious actors. These bad actors could include individuals, collectives, or nation-states. They may be internal or external to the development team.

Some known goals of hackers include

- › near-term financial goals, such as resulting from ransomware attacks
- › data collection for future use or to perform pattern analysis
- › destroying or damaging critical systems
- › intelligence gathering
- › industrial espionage.

Some users and organizations inadvertently enable successful attacks on open source software by

- › using programs with typos in the program name
- › using programs with updated fictitious version numbers
- › falling victim to phishing attacks
- › lacking awareness of published warnings about compromised software and continuing to download compromised versions.

In addition to obvious near-term financial goals, such as ransomware,

quiet tampering, with some unknown future attack in mind, can also take place. Reports on ransomware and quiet tampering appear in the media daily. Sometimes, the goal is to collect data from users of the open source software and do pattern analysis. There are times when it is unknown whether the malicious actor is an individual, a team, or a nation-state. What's worse, even when a successful attack is detected and documented, some users continue to download the compromised software! Given that this is the case, it seems obvious that open source can no longer be automatically trusted. It's not just a matter of coding errors but cybersecurity risk. Here are some examples of compromised open source as well as efforts underway to measure the trustworthiness of a given open source package.

OPEN SOURCE CYBERSECURITY COMPROMISES

A few examples of compromises to widely used open source software can be found in Mead et al.,⁵ which provides the basis for much of this column and is available for download by those readers who want to dig a little deeper.

Log4j vulnerability

The Cyber Safety Review Board found that the Log4j vulnerability is too

widespread over Internet-connected systems to be completely contained.² First disclosed in December 2021, the Log4j vulnerability is a critical security flaw in a popular piece of Java logging software. It has been in circulation since 2012 and is embedded in millions of software packages, and additional downloads of the software occur daily. Clearly, a unified effort across organizations is needed to eliminate this vulnerability.⁴

Unfortunately, such a unified effort is far from reality. Although a patched version of Log4j is available, quoting from the Sonatype report⁷

“As of September 2023, downloads vulnerable to the infamous Log4Shell vulnerability still account for nearly a quarter of all new **downloads of Log4j**. It should be highlighted, that almost two years after the initial finding of this vulnerability, we're seeing this pace continue every week—that a quarter of all **net new downloads are of the vulnerable version of Log4j**. This is only part of the story. The reality is, nearly 1/3 of all Log4j downloads, ever, are of the vulnerable version.”

This bit of information, combined with the fact that 39% of organizations surveyed by Sonatype take more than a week to mitigate vulnerabilities, paints a grim picture of the current state of open source cybersecurity.

xz Utils Backdoor

The details of this attack are best described at <https://arstechnica.com/security/2024/04/what-we-know-about-the-xz-utils-backdoor-that-almost-infected-the-world/>. Quoting from just part of the report

“On Friday, a lone Microsoft developer rocked the world when he revealed a backdoor had been intentionally planted in xz Utils, an open source data compression utility available on almost all installations of Linux and other Unix-like operating systems. The person or people behind this project likely spent years on it. They were likely very close to seeing the backdoor update merged into Debian and Red Hat, the two biggest distributions of Linux, when an eagle-eyed software developer spotted something fishy.”

The person or persons responsible spent years establishing credibility to become one of the primary maintainers of the widely used open source software. This occurred in part because an excessive number of requests for new features, possibly from the same collective group, put the primary maintainer into overload. He then asked the newer “trusted” developer to help out. The backdoor would have allowed the attacker to inject malicious code during Secure Shell (SSH) operations, thereby overcoming

standard security measures. The footprint of the persona has all but disappeared, and speculation is that it could have been the work of a nation-state—in one of several possible geographic areas. However, that’s all it is, just speculation.

CURRENT TECHNIQUES FOR CYBERSECURITY MEASUREMENT OF OPEN SOURCE

Several attempts have been and are being made to assess whether a particular piece of open source software is secure. Some of these efforts attempt to define a way of measuring how secure a particular piece of open source software might be. This, of course, is highly desirable from a risk assessment perspective. A couple of examples of these measurement approaches are described here. The field evolves rapidly, so there may be many other approaches available.

The Open Source Security Foundation (OSSF) Scorecard is a tool available for free download that incorporates a set of metrics that can be applied to an open source software project. The idea is that those project attributes that OSSF believes contribute to a more secure open source application

are then reported using a weighted approach that leads to a score. There are around 20 project attributes that are evaluated by default. Each attribute can be rated from zero to 10, with 10 being the best score, and the risk level per attribute is set as low, medium, high, or critical to produce a weighted average considering both score and risk level. Scorecard is run weekly against 1 million open source projects deemed critical, and the results are publicly available. A snippet of the checks run by default is shown in Table 1 (see <https://github.com/ossf/scorecard?tab=readme-ov-file#scorecard-checks>).

From a metrics perspective, there are limitations to this approach.

- › The open source community is driving and evolving which items to measure and, therefore, build into the tool. Also, it is not clear how those factors were determined, whether the set of factors is complete, or what is intended for the long-term road map (that is, insufficient transparency).
- › The relative importance of each factor is also built into the tool, which makes it difficult (but

TABLE 1. Partial set of default checks run by scorecard. (Source: Adapted from <https://github.com/ossf/scorecard?tab=readme-ov-file#scorecard-checks>.)

Name	Description	Risk level	Token required	GitLab support	Note
Binary-Artifacts	Is the project free of checked-in binaries?	High	PAT, GITHUB_TOKEN	Supported	
Branch-Protection	Does the project use Branch Protection?	High	PAT (repo or repo > public_repo), GITHUB_TOKEN	Supported (see notes)	Certain settings are only supported with a maintainer PAT
CI-Tests	Does the project run tests in CI, for example, GitHub Actions, Prow?	Low	PAT, GITHUB_TOKEN	Supported	
CII-Best-Practices	Has the project earned an OpenSSF (formerly CII) Best Practices Badge at the passing, silver, or gold level?	Low	PAT, GITHUB_TOKEN	Validating	

not impossible) to tailor the results to specific and custom end-user needs.¹¹

- › Many of the items measured in the tool appear to be self-reported by the developer(s) versus validated by a third party, but this is a common “attribute” of open source projects.

Other tools, such as MITRE’s Hipcheck, are also available.⁶ Hipcheck analyzes repositories, and its documentation says it can answer questions such as the following:

- › Does this project practice code review?
- › When was this project last updated?
- › Are there concerning contributors to this project?
- › Are there potential malicious contributions to review?
- › Are there potential typosquatting attacks present?
- › Where are the highest-risk parts of the codebase?

For pull requests, its documentation similarly says it provides answers to questions such as the following:

- › What parts of the code are in the greatest need of review?
- › Is this pull request especially concerning?
- › Is this contributor new to this part of the code?

It’s not surprising that Hipcheck has some of the same limitations as Scorecard.

For an OSSF project, it is possible to get a score for the project using Scorecard along with scores for the individual dependency projects, but questions arise from this approach. How do those individual scores roll up into the overall score? Does the user pick the lowest score across all the dependencies or apply some sort of weighted average of scores? This area needs exploration and elaboration.

Furthermore, a recent research paper¹⁰ indicated cases where open source projects that score highly by Scorecard might, in fact, produce packages that have more reported vulnerabilities. From a research perspective, it is unknown whether this

while keeping in mind that the goal is to reduce vulnerabilities.

Measures that are effective more generally in supply chain risk management can also be applied to open source software. Documentation and examples of how to define these mea-

Even when a successful attack is detected and documented, some users continue to download the compromised software!

occurs because the application has received more reviews (and therefore more vulnerabilities were identified) or whether attacks on a popular application have exposed it to more vulnerabilities. Needless to say, the results of Zahan et al.¹⁰ are useful only for those open source projects evaluated by the tool, which is applied exclusively to GitHub, and those are only a fraction of the total number of open source applications available. All these issues indicate that further study is needed.

EXAMPLES OF POTENTIALLY USEFUL MEASURES

An extensive three-year study of security testing and analysis revealed that 92% of tests discovered vulnerabilities in the applications being tested.⁸ Despite showing improvement year over year, the numbers still indicate an unacceptable state of affairs. In addition, 27% of tests identified high-severity vulnerabilities, and 6.2% identified critical-severity vulnerabilities. In the Synopsys study, improvements in open source software appeared to link to improved development processes, including inspection and testing. However, older open source software that is no longer maintained still exists in some libraries, and it can be downloaded without those corresponding improvements. This study and others indicate that the community has started making progress in this area by defining measures that go beyond identifying vulnerabilities in open source software

and collect and analyze relevant data in various phases of the software assurance lifecycle already exist.⁹ The report discusses how the Software Assurance Framework (SAF) illustrates promising metrics for specific activities. SAF examines practices in the categories of process management, project management, engineering, and support. For each category, multiple areas of practice as well as specific practices are specified.

This is demonstrated in Table 2, which pertains to SAF “Practice Area 2.4 Program Risk Management” and addresses the question, “Does the program manage program-level cybersecurity risks?” Table 2, in addition to itemizing specific practices and their outputs, postulates candidate metrics. This work illustrates what can be done, although much more work is needed to flesh out all the desired candidate metrics and to assess their effectiveness.

WHAT ELSE IS NEEDED?

Once all the metrics needed to predict cybersecurity in open source software are understood, and this problem has not yet been solved, standards will be needed that make it easier to apply these metrics. Standards organizations, such as NIST and ISO, and cybersecurity-specific standards organizations, such as CISQ, could take on these challenges. Providers could consider including software products that come with metrics that help users

TABLE 2. SAF Practice Area 2.4 Program Risk Management: Does the program manage program-level cybersecurity risks?

Activities/practices	Outputs	Candidate metrics
Ensure that project strategies and plans address project-level cybersecurity risks (for example, program risks related to cybersecurity resources and funding)	Program Plan Technology Development Strategy (TDS) Analysis of Alternatives (AoA)	Percentage of program managers receiving cybersecurity risk training Percentage of programs with cybersecurity-related risk management plans
Identify and manage project-level cybersecurity risks (for example, program risks related to cybersecurity resources and funding)	Risk Management Plan Risk Repository	Percentage of programs with cybersecurity-related risks Number of cybersecurity-related risks tracked per month

understand the product’s cybersecurity posture.

As an example, at the operational level, Vulnerability Exploitability eXchange (VEX) helps users understand whether or not a particular product is affected by a specific vulnerability.¹ A VEX document is “a machine-readable collection of information conveying the status of products or components with respect to a vulnerability.” The description of

analyzing data is that when they are collected, they may not be collected or documented in a standard way. Reports are often written in unstructured prose that is not amenable to analysis, even when the reports are scanned, searched for keywords and phrases, and analyzed in a standard way. When reports are written in a nonstandard way, analyzing the content to achieve consistent results is challenging.

2024. [Online]. Available: [https://www.cisa.gov/sites/default/files/2023-04/33’minimum-requirements-for-vex-508c.pdf](https://www.cisa.gov/sites/default/files/2023-04/33%27minimum-requirements-for-vex-508c.pdf)


2. Cyber Safety Review Board (CSRB). “Review of the December 2021 Log4j Event. Cybersecurity and Infrastructure Security Agency (CISA).” Accessed: Jul. 18, 2024. [Online]. Available: https://www.cisa.gov/sites/default/files/publications/CSRB-Report-on-Log4-July-11-2022_508.pdf
3. European Union Agency for Cybersecurity (ENISA). “Good practices for supply chain cybersecurity.” ENISA Website. Accessed: Jul. 18, 2024. [Online]. Available: <https://www.enisa.europa.eu/publications/good-practices-for-supply-chain-cybersecurity>
4. S. Ikeda. “New cyber safety review board report: Log4j vulnerability is “endemic,” expect it to be exploited into the 2030s.” CPO Magazine. Accessed: Jul. 18, 2024. [Online]. Available: <https://www.cpomagazine.com/cyber-security/new-cyber-safety-review-board-report-log4j-vulnerability-is-endemic-expect-it-to-be-exploited-into-the-2030s/>
5. N. R. Mead, C. Woody, and S. Hissam. “The measurement challenges in software assurance and supply chain risk management.” SEI Website. Accessed: Dec. 22, 2023. [Online]. Available: <https://insights.sei.cmu.edu/library/measurement-challenges-in-sw-assurance-and-scrm-white-paper/>

Improvements in open source software appeared to link to improved development processes, including inspection and testing.

VEX further states: “VEX is designed to integrate with SBOM, vulnerability databases, and security advisories, but does not require any of these. VEX documents can be authored by the supplier of the software or by a third party.”

Such publicly available information can help users make choices about open source and other products in the supply chain. Of course, this is just one example of how data might be collected and used, and it focuses on vulnerabilities in existing software.

Similar standard ways of documenting and reporting cybersecurity risk are needed throughout the software product development process. One of the challenges in

In the meantime, until the ideal state is reached and standard trusted methods for measuring the cybersecurity of open source software exist, practitioners and managers need to take advantage of the measurement tools and reports that are currently available, such as those available from CISA and ENISA.³ Open source software that has *already* been reported as compromised should not continue to be downloaded and used, as happens so often at present. 

REFERENCES

1. Cybersecurity & Infrastructure Security Agency (CISA). “Minimum requirements for vulnerability exploitability eXchange (VEX).” CISA (.gov). Accessed: Jul. 18,

6. MITRE Corporation. "Hipcheck." GitHub Website. Accessed: Dec. 19, 2023. [Online]. Available: <https://github.com/mitre/hipcheck>
7. Sonatype Incorporated. "Sonatype 9th annual state of the software supply Chain report." Sonatype Website. Accessed: Jul. 18, 2024. [Online]. Available: <https://www.sonatype.com/state-of-the-software-supply-chain/introduction>
8. Synopsys. "2023 open source security and risk analysis report." Synopsys Website. Accessed: Dec. 18, 2023. [Online]. Available: <https://www.synopsys.com/software-integrity/engage/ossra/rep-ossra-2023-pdf>
9. C. Woody, R. Ellison, and C. Ryan, *Exploring the Use of Metrics for Software Assurance*. CMU/SEI-2018-TN-004. Pittsburgh, PA, USA: Software Engineering Institute, Carnegie Mellon Univ., 2019.
10. N. Zahan, S. Shohan, D. Harris, and L. Williams, "Do software security practices yield fewer vulnerabilities?" in *Proc. IEEE/ACM 45th Int. Conf. Softw. Eng., Softw. Eng. Pract. (ICSE-SEIP)*, Piscataway, NJ, USA: IEEE Press, 2023, pp. 292–303, doi: [10.1109/ICSE-SEIP58684.2023.00032](https://doi.org/10.1109/ICSE-SEIP58684.2023.00032).
11. "OpenSSF scorecard – Security health metrics for open source." GitHub. Accessed: Jul. 18, 2024. [Online]. Available: <https://github.com/ossf/scorecard>

NANCY R. MEAD is a fellow at the Software Engineering Institute and an adjunct professor of software engineering at Carnegie Mellon University, Pittsburgh, PA 15213 USA. Contact her at nm00@andrew.cmu.edu.

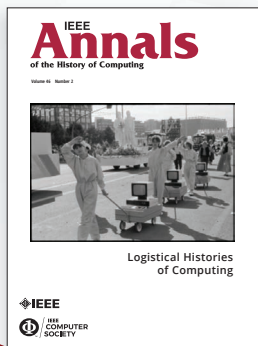
CAROL WOODY has been a senior member of the technical staff at the Software Engineering Institute,

Carnegie Mellon University, Pittsburgh, PA 15213 USA, since 2001. Currently, she is the technical manager for the Cyber Security Engineering team. Contact her at cwoody@cert.org.

SCOTT HISSAM is with the Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA 15213 USA. Contact him at shissam@sei.cmu.edu.

IEEE Annals

of the History of Computing



IEEE Annals of the History of Computing publishes work covering the broad history of computer technology, including technical, economic, political, social, cultural, institutional, and material aspects of computing. Featuring scholarly articles by historians, computer scientists, and interdisciplinary scholars in fields such as media studies and science and technology studies, as well as firsthand accounts, *Annals* is the primary scholarly publication for recording, analyzing, and debating the history of computing.



www.computer.org/annals

