Carnegie Mellon University
Software Engineering Institute

THE MEASUREMENT CHALLENGES IN SOFTWARE ASSURANCE AND SUPPLY CHAIN RISK MANAGEMENT

Nancy R. Mead (Carnegie Mellon University), Carol Woody, Scott A. Hissam

December 2023

Challenges

Software supply chain risk has increased exponentially since 2009 when the Heartland Payments System breach [Lewis 2015] made the issue newsworthy. The perpetrators reaped 100 million debit and credit card numbers. At the time, this was the largest data breach in recorded history, but it would not remain so. Recent events in 2020 and 2021, such as SolarWinds and Log4j, a popular logging package for Java [Liu 2021], show that the scale of disruption from a third-party software supplier can be massive as organizations grow their dependence on software-reliant technology.

The reuse of software has enabled faster fielding of systems since common components can be sourced externally, but all software comes with vulnerabilities, and attackers have expanded their capabilities to exploit them in products that have broad use. A recent report by SecurityScorecard [Townsend 2023] found that 98 percent of their sampled 230,000 organizations have had third-party software components breached within the prior two years.

Organizations shifting to cloud services to eliminate on-premise risk have frequently been surprised by supply chain risks inherited from service providers, which resulted from misconfigurations, unauthorized access, insecure application programming interfaces (APIs), etc. [Check Point 2023]. To identify and manage this growing risk landscape, organizations must increase collaboration across the range of participants involved in the selection, installation, and monitoring of third-party software to identify and manage potential risks.

Earlier research addressed by the Software Engineering Institute (SEI) assembled practices critical to meeting this need in the Acquisition Security Framework (ASF) [Alberts 2022]. However, each organization has a unique technology environment, and there are no widely accepted measures for evaluating their accepted risk.

Diving into supply chain risk a little further, supply chain attacks initially appeared in third-party software, which was either developed through custom contracts or purchased as commercial software. It is a bit easier to tailor a custom contract to protect against these risks, and the guidance available for supply chain risk management (SCRM) can go a long way towards developing a plan. Establishing and implementing a SCRM plan, however, is an expense that many organizations will not sign up for.

A recent European Union (EU) study [ENISA 2023] included 1,081 organizations in 27 member states. In this report, the researchers noted that only 47 percent of the surveyed organizations in the EU have an allocated budget for information and communication technology and operational technology (ICT/OT) for supply chain cybersecurity. Researchers further observed that 52 percent of the organizations surveyed have a rigid patching policy that includes 80 percent or more of their assets. On the other hand, only 13.5 percent of the surveyed organizations have visibility into patching for less than half their assets, meaning that patching may be done by a third-party organization or not at all.

The situation is no better for users of open source software, which is the latest target of attackers in the software supply chain. For years, many considered open source software to be more secure because its code was visible and developed by "trusted" individuals. An early study [Hissam 2002] provided insight into a widespread attack against both proprietary and open source software operating systems—the TearDrop and NewTear attacks—that were actually informed by open source software before vulnerability disclosure processes were widely used. Contrary to popular belief, open source software has never really been secure; whether or not a piece of open source software is secure is in the eye of the beholder—be they altruistic or nefarious.

A Sonatype survey of 621 practitioners indicated that only 28 percent of their organizations become aware of new open source vulnerabilities within a day of disclosure, 39 percent discover them within one to seven days, and 29 percent take more than a week to become aware of them [Sonatype 2023]. The same survey found that 39 percent of the respondents' organizations take more than a week to mitigate vulnerabilities.

Examples

In this section, we discuss two examples of open source vulnerabilities: Log4j and Black Hat Europe.

Log4j Vulnerability. The Cyber Safety Review Board found that the Log4j vulnerability is too widespread over Internet-connected systems to be completely contained [CSRB 2022]. 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, 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 [Ikeda 2022].

Unfortunately, such a unified effort is far from reality. Although a patched version of Log4j is available, quoting from the Sonatype report [Sonatype 2023]:

> *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 data, combined with the fact that 39 percent of organizations surveyed by Sonatype take more than a week to mitigate vulnerabilities, paints a grim picture for the current state of open source cybersecurity.

Despite a 96 percent chance of there being a problem-free version available, 3.97 billion of the 37.8 billion monthly downloads from Maven Central contain a vulnerability. Only 1.8 billion of these downloads are of items that have no available fix, leaving over 2.1 billion installations of vulnerabilities each month that could be easily avoided.

Black Hat Europe. In another example of open source vulnerabilities, the following was reported in advance of 2019 Black Hat Europe[Nelson 2023]:

> *Researchers have discovered 21 vulnerabilities in a popular brand of industrial router. Seven of the newly discovered vulnerabilities lie in internal components of the routers.* ***Fourteen of them derive from open source components****, specifically, a captive portal for Wi-Fi networks and an XML processing library. The nature of the vulnerabilities run the gamut: cross-site scripting (XSS), denial of service (DoS), remote code execution (RCE), unauthorized access, and authentication bypass.*

Cost

Intellias summarizes the estimated cost of supply chain attacks quite well [Intellias 2023]. In 2023, the MOVEit vulnerability cost businesses over $9.9 billion, with more than 1,000 businesses and over 60 million individuals affected. Furthermore, the estimated cost from just seven high-profile supply chain attacks, starting with SolarWinds, is around $60 billion. This figure does not include the impact of government-imposed fines and legal actions related to privacy laws on both the affected businesses and companies that rely on them. In this regard, there is little distinction between supply chain attacks resulting from compromised proprietary software versus compromised open source software.

Any successful supply chain attack can result in substantial financial loss, loss of reputation, lawsuits, and investigations. In recent articles [Birsan 2023, O'Neill 2023], it was revealed that the information of 1.6 million patients was compromised by a successful MOVEit hack. Even though the vulnerability was documented in May 2023, that particular hack was not discovered until October 2023. The patch to the vulnerability had not yet been applied.

In a report on the recovery from the successful attack on Metro-Goldwyn-Mayer (MGM) casinos [Jones 2023], interviews with management indicated a loss of $100 million at its Las Vegas properties. The loss is covered by cyber insurance; however, the cost of that insurance has doubled or, in some cases, quadrupled in recent years. Of course, this loss estimate includes only the immediate loss of revenue and does not consider the result of class action lawsuits, some of which have already been filed. As is typically the case, the organization plans to invest heavily in information technology (IT) **after** the successful attack.

Measurement

We have seen challenges for addressing software assurance across the lifecycle that are directly impacted by the limitations in measuring software assurance.

Suppliers should be proactive. Yet leadership across the supply chain continues to underinvest in software assurance, especially early in the lifecycle. This lack of investment leads to design decisions that lock in weaknesses because there is no means to characterize and measure the risk they are accepting. Suppliers rush to deliver new features to motivate buyers at the expense of analyzing the code to remove potential vulnerabilities, and buyers have limited means to evaluate the risk in products they acquire.

Even if a supplier addresses an identified vulnerability quickly and issues a patch, it is incumbent on the users of that software to apply the fix. Software supply chains are many levels deep, and too frequently the patches apply to products buried deep within a chain. Each layer must apply the patch and send an update up the chain. This can be a slow and faulty process since knowledge of where each specific product has been used is limited for those higher in the chain. Recent mandates to create software bills of materials (SBOMs) [White House 2022] support an attempt to improve visibility, but the fix still needs to be addressed by each of the many layers that contain the vulnerability.

The Open Source Security Foundation (OSSF) Scorecard is a tool 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.

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

1. 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 roadmap (i.e., insufficient transparency).
2. The relative importance of each factor is also built into the tool, which makes it difficult (but not impossible) to tailor the results to specific, custom, end-user needs [OSSF 2023].
3. 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, have the same limitations [MITRE 2023]. 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? Do you pick the lowest score across all the dependencies, or do you apply some sort of weighted average of scores? This area needs exploration and elaboration.

Furthermore, a recent research paper [Zahan 2023] 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 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, Zahan's results 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.

How Do We Measure Software Cybersecurity Risk?

State of the Practice

Currently, it is possible to collect vast amounts of data related to cybersecurity in general. We can measure financial loss due to a successful attack and loss of confidence in a particular company as measured by the loss of customers or impacts on stock values. We can measure the elapsed time from the beginning of a successful attack until its discovery and the elapsed time from discovery until mitigation and recovery.

We can also measure specific product characteristics related to cybersecurity. Commercial companies, nonprofits, and government entities offer data collection tools. What is not always clear is the cause-and-effect relationship between the data, vulnerabilities, and successful attacks. Also, much of the data collected reflects the results of an attack, whether attempted or successful. Data on earlier security lifecycle activities often reflects the development processes used. Although needed, it is not diligently collected, nor is it analyzed as thoroughly as in later points of the lifecycle.

As software engineers, we believe that improved software practices and processes will result in a more robust and secure product. However, which specific practices and processes actually result in a more secure product? There can be quite a bit of elapsed time between the implementation of improved processes and practices and the subsequent deployment of the product. If the product is not successfully attacked, does it mean that it's more secure? Or does it just mean that it's a less interesting target from an attacker perspective? Zahan concludes the following [Zahan 2023]:

> *Security metrics are a hard problem, especially in predicting vulnerabilities or assessing the effectiveness of counter measures [Cheng 2014, Scala 2019]. We should consider that the software security field has an inherently greater amount of unexpected variation.*

Consider the recently updated Cyber Security & Information Systems Information Analysis Center (CSIAC) *The DoD Cybersecurity Policy Chart* [CSIAC 2023]. Although the authors have made a valiant effort to categorize and classify the subject policies, how can project managers be expected to sift through hundreds if not thousands of pages of documentation to find out what policy applies to them?

Certainly, government contractors have a profit motive that justifies meeting the cybersecurity policy requirements that apply to them, but do they know how to measure the cybersecurity risk of their products? And how would they know whether it has improved sufficiently? For open source software, when developers are not compensated, what would motivate them to do this? Why would they even care whether a particular organization—be it academic, industry, or government—is motivated to use their product?

Currently Available Metrics

The SEI led a research effort to identify the metrics currently available within the lifecycle [Woody, 2018] that could be used to provide indicators of potential cybersecurity risk. From an acquisition lifecycle perspective, there are two critical information needs:

- Is the acquisition headed in the right direction as it is engineered and built (predictive)?
- Is the implementation sustaining an acceptable level of operational assurance (reactive)?

Each step in the lifecycle can produce useful information, but the lifecycle requires structuring the processes and practices to gather the data in a form that can be analyzed and enforcing the creation of appropriate measures as each process or practice is performed. Trends in current approaches show an increased reliance on tools selected to perform lifecycle steps, but there is limited consideration of how the available data will be integrated for useful analysis.

For code, the work of Capers Jones in collecting and tracking defects to evaluate quality [Jones 2014] provides an opportunity for setting expectations of code quality based on the level of defects. SEI research leveraged this data to project an expectation of vulnerabilities based on defect rates [Woody 2014]. As development shifts further into Agile increments, many of which include third-party and open source components, different tools and definitions are applied to collecting defects so that the meaning of this metric in predicting risk becomes obscured.

Highly vulnerable components that are implemented using effective and well-managed zero trust principles can deliver acceptable operational risk. In a similar vein, well-constructed, high-quality components with weak interfaces can be highly prone to successful attacks. Operational context is critical to the risk exposure. A simple evaluation of each potential vulnerability using something like a Common Vulnerability Scoring System (CVSS) score can be extremely misleading since the score without the context has limited value in determining actual risk.

The lack of visibility into the development processes and methods used to develop third-party software—particularly open source software—means that measures related to the processes used and the errors found prior to deployment, if they exist, do not add to the useful information about the product. This lack of visibility into product resilience as it relates to the process used to develop it means that we do not have a full picture of the risks, nor do we know whether the processes used to develop the product have been effective. It's difficult, if not impossible, to measure what is not visible.

Consider a recent blog post by Eric Goldstein from CISA [Goldstein 2023]. It announces a *Secure by Design Alert Series*, and states the following:

> *Insecure technology products are not an issue of academic concern: they are directly harming critical infrastructure, small businesses, local communities, and American families.*

This, of course, is not new information. In addition to its implications for large companies producing software products, the same concerns apply to software in the supply chain **and** open source software. One might imagine that large companies will respond in some fashion, but how will the open source community respond? Are open source developers concerned with using specific languages, such as RUST, that are supportive of security goals? Are they concerned with using default settings that are inherently more secure?

More important, from our viewpoint, is that these improvements result from decisions that do not cover the myriad decisions in the development process. Additionally, as before, although it may be easier to measure the security improvements resulting from decisions such as these, many other decisions are made during the development process for which the corresponding improvement (or lack thereof) in security may be unknown. This could occur because we do not have good ways to measure security improvement or because we do not have evidence to support the choice of those measurement techniques.

Lessons Learned from Outsourcing

One might wonder whether there are lessons learned from outsourcing that can be applied to open source software. Outsourced software is acquired from third parties, and open source software is also acquired from third parties. Carrying this a little further, one might also consider reused software to be similar to open source, except, in that case, the third party is another in-house project, but the source code is typically available to the new project. Let us take a look at lessons learned from outsourcing, and how one might apply measurement.

Dr. Howard Rubin noted the following [Rubin 2000]:

> *Lessons learned through outsourcing successes and failures of the past indicate that a 'critical failure factor' of an outsourcing agreement is the selection of a set of performance measures to manage the agreement that*
>
> 1. *do not align with the business needs and objectives of all parties*
> 2. *are so inflexible that they cannot be adapted to changing needs*
> 3. *present unrealistic performance targets*
> 4. *act as lagging and not 'leading' indicators in that they report results reactively instead of providing a 'look ahead' view of performance*
> 5. *do not cover the complete scope of work covered by the agreement*
> 6. *do not consider the natural performance evolution of an agreement from transition to maturity/steady-state evolution.*
> 7. *In short, the measures selected to guide an outsourcing agreement essentially play the role of 'performance engineering targets'*

How should these lessons be adjusted for open source software? Many of the ideas apply, but it may indeed be the case that there is no agreement between the user and the creator of open source software. There are instances of organizations paying developers of open source software to provide software that is specific to their project. Such software may or may not be covered by an agreement that would account for the risks that Rubin itemized. If there is no agreement between the user and the supplier, that leaves the user with sole responsibility for defining the performance measurements and determining whether or not they are being met. For cybersecurity of open source software, by and large, the indicators are lagging (e.g., vulnerabilities, successful attacks) rather than leading.

As noted above, when the processes used for product development are not visible, it is impossible to determine whether they have been effective in terms of thwarting future attacks. How does one

improve a process when the process is invisible and the measurement that should be taking place is also invisible? More to the point, how can we get leading indicators for open source cybersecurity when the process is invisible?

What Are Useful Measures for Process and Product?

Measurement Frameworks Applied to Cybersecurity

Let us consider the history of software measurement. Initially, software measurement was basically concerned with tracking tangible items that provided immediate feedback [Scheff 1979, Curtis 1979]. Project cost estimation and completion tracking tended to be based on lines of code or function points, and many different ways of measuring code size were developed.

Then code quality measures started to be considered. Complexity measures, such as those developed by McCabe and Halstead, were used as ways of predicting code quality. Other measures, such as bug counts in trouble reports, errors found during inspection, and mean time between failures drove some measurement efforts [McCabe 1976, Wikipedia 2023]. Evidence surfaced that supported the fact that it was less costly to locate and correct errors early in the software process rather than later. However, convincing development managers to spend more money upfront for cost reduction across the project's lifecycle was a tough sell given that their performance evaluations heavily relied on containing development costs. Therefore, adopting improved development processes often depends on getting buy-in from executive management.

A few dedicated researchers tracked the measurement results over a long period of time. Basili and Rombach's seminal work in measurement resulted in the Goal-Question-Metric (GQM) method for helping managers of software projects decide what measurement data would be useful to them [Basili 1984, Basili 1988]. Experiments run in the Goddard Software Engineering Lab provided support for this approach.

Building on this seminal work, the SEI added a step that linked the questions to the measurement data collected, resulting in the Goal, Question, Indicator, Metric (GQIM) method. The indicators identify information needed to answer each question. Then, in turn, metrics are identified that use the indicators to answer the question. This additional step reminds stakeholders of the practical aspects of data collection and provides a way of ensuring that the needed data is collected for the selected metrics [Boyd 2002]. This method has already been applied by both civilian and military stakeholders [Gray 2016].

Now, let us consider the situation we face with cybersecurity. Similar data has been collected, and shows that it is less costly to correct errors that might lead to vulnerabilities early in the lifecycle, rather than later when software is operational. Studies of real projects have illustrated that it can be up to 100 times cheaper to find and fix cybersecurity errors at requirements time compared to when software is operational.

The results of those studies help answer questions about development cost and reinforce the importance of using good development processes. In that regard, those results support our intuition. For open source software, if there is no visibility into the development process, we do not have that information. Furthermore, even when we know something about the development process, the total cost associated with a vulnerability after software is operational can range from zero (if it is never found and exploited) to millions of dollars.

Over the history of software engineering, we have learned that software metrics for both the process and the product are needed. This is no different in the case of the cybersecurity of open source software. We must be able to measure the processes for developing and using software and how those measurement results affect the product's cybersecurity. It is insufficient to measure only operational code, its vulnerabilities, and the attendant risk of successful hacks. In addition, success hinges on a collaborative, unbiased effort that allows multiple organizations to participate under a suitable umbrella.

Primary Buyers vs. Third-Party Buyers

Consider the cases that apply when software is acquired rather than developed in house [Mani 2014]:

- In the first case, acquirers of custom contract software can require that the contractor provide visibility into both their development practices and their SCRM plan. When the supplier is developing the software product, the product and processes are visible and can be measured at each step.
- In the second case, acquirers can specify the requirements, but the development process is not visible to the buyer. Hence, they are not measurable by the buyer.
- In the third case, the software product already exists, and the buyer is typically just purchasing a license. The code for the product may or may not be visible, further limiting what can be measured. The product could also, in turn, contain code developed further down in the supply chain, thus complicating the measurement process.

Open source software resembles the third case. The code is visible, but the process used to develop it is invisible unless the developers choose to describe it. The value of having this description depends on the acquirer's ability to determine what is good versus poor quality code.

Today, many U.S. Government contracts require the supplier to have an acceptable SCRM plan, the effectiveness of which can presumably be measured. Nevertheless, a deep supply chain—with many levels of buyers **and** dependencies—clearly is concerning. First, you have to know what is in the chain, then you have to have a way of measuring each component, and finally you need trustworthy algorithms to produce a bottom line set of measurements for the ultimate product constructed from a chain of products.

Measuring the risks associated with the attack surface of the ultimate product is helpful but only if you can determine what the attack surface is. With open source, if the build picks up the latest version of the product, the measurement process should be revisited to ensure you still have a valid bottom line number. However, this approach presents a number of questions:

1. Is measurement being done?
2. How effective is the measurement process and its results?
3. Is measurement repeated every time a component in the product/build changes?
4. Do you even know when a component in the product/build changes?

Examples of Potentially Useful Measures

An extensive three-year study of security testing and analysis reveals that 92 percent of tests discovered vulnerabilities in the applications being tested [Synopsys 2023]. Despite showing improvement year over year, the numbers still present a grim picture of the current state of affairs. In addition, 27 percent of tests identified high-severity vulnerabilities, and 6.2 percent 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 while keeping in mind that the goal is to reduce vulnerabilities. Measures that are effective in SCRM are relevant to open source software. Documentation and examples of how to define these measures and collect and analyze relevant data in various phases of the software assurance lifecycle already exist [Woody 2018]. In their report, Woody, Ellison, and Ryan discuss how the Software Assurance Framework (SAF) illustrates promising metrics for specific activities. They demonstrate this in the table below, which pertains to SAF *Practice Area 2.4 Program Risk Management* and addresses the question, "Does the program manage program-level cybersecurity risks?"

Activities/Practices	Outputs	Candidate Metrics
Ensure that project strategies and plans address project-level cybersecurity risks (e.g., program risks related to cybersecurity resources and funding).	Program Plan Technology Development Strategy (TDS) Analysis of Alternatives (AoA)	% program managers receiving cybersecurity risk training % programs with cybersecurity related risk management plans
Identify and manage project-level cybersecurity risks (e.g., program risks related to cybersecurity resources and funding).	Risk Management Plan Risk Repository	% programs with cybersecurity related risks # cybersecurity related risks tracked per month

Conclusion and Recommendations

Once we understand all the metrics needed to predict cybersecurity in open source software, we will need standards that make it easier to apply these metrics to open source and other software in the supply chain. Providers could consider including software products that come with metrics that help users 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 [CISA 2023]. 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 that we have faced in analyzing data is that when it is collected, it 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 key words and phrases, and analyzed in a standard way. When reports are written in a non-standard way, analyzing the content to achieve consistent results is challenging.

We have provided some examples of potentially useful metrics, but data collection and analysis will be needed to validate that they are, in fact, useful in the supply chains that include open source software. This validation requires standards that support data collection and analysis methods and evidence that affirms the usefulness of a specific method. Such evidence may start with case studies, but these need to be reinforced over time with numerous examples that clearly demonstrate the utility of the metrics in terms of fewer hacks, reduced expenditure of time and money over the life of a product, enhanced organizational reputation, and other measures of value.

New metrics that have not yet been postulated must also be developed. Some research papers may describe novel metrics along with a case study or two. However, the massive amount of data collection and analysis needed to truly have confidence in these metrics seldom happens. New metrics either "fall by the wayside" or are adopted willy-nilly because renowned researchers and influential organizations endorse them, whether or not there is sufficient evidence to support their use. We believe that defining metrics, collecting and analyzing data to illustrate their utility, and using standard methods requires unbiased collaborative work to take place for the desired results to come to fruition.

Acknowledgments

The authors acknowledge the support from Sayeed Choudhury and the Carnegie Mellon University (CMU) Libraries. We also appreciate the helpful comments from Sayeed Choudhury, and from Andrew Lilley Brinker and his colleagues at MITRE. Our editors, Sandy Shrum and Barbara White, contributed greatly to the readability of this paper.

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM23-2391

References

[Alberts 2022]
Alberts, Christopher J.; Bandor, Michael S.; Wallen, Charles M.; & Woody, Carol. *Acquisition Security Framework (ASF): An Acquisition and Supplier Perspective on Managing Software-Intensive Systems' Cybersecurity Risk*. Software Engineering Institute, Carnegie Mellon University. October 4, 2022. https://insights.sei.cmu.edu/library/acquisition-security-framework-asf-an-acquisition-and-supplier-perspective-on-managing-software-intensive-systems-cybersecurity-risk/

[Basili 1984]
Basili, Victor R. & Weiss, David M. A Methodology for Collecting Valid Software Engineering Data. *IEEE Transactions on Software Engineering*. Volume 10. Number 3. November 1984. Pages 728-738. https://ieeexplore.ieee.org/document/5010301

[Basili 1988]
Basili, Victor R. & Rombach, H. Dieter. The TAME Project: Towards Improvement-Oriented Software Environments. *IEEE Transactions on Software Engineering*. Volume 14. Article 6. June 1988. Pages 758–773. https://ieeexplore.ieee.org/document/6156

[Birsan 2023]
Birsan, Alex. Dependency Confusion: How I Hacked Into Apple, Microsoft and Dozens of Other Companies [blog post]. *Medium Blog*. February 9, 2021. https://medium.com/@alex.birsan/dependency-confusion-4a5d60fec610

[Boyd 2002]
Boyd, Andrew. The Goals, Questions, Indicators, Measures (GQIM) Approach to the Measurement of Customer Satisfaction with E-Commerce Web Sites. Pages 177-187. In *Aslib Proceedings*. Volume 54. Number 3. June 2002. http://dx.doi.org/10.1108/00012530210441728

[Check Point 2023]
Check Point Software Technologies Ltd. Main Cloud Security Issues and Threats in 2023. *Check Point Website*. December 19, 2023 [accessed]. https://www.checkpoint.com/cyber-hub/cloud-security/what-is-cloud-security/top-cloud-security-issues-threats-and-concerns/

[Cheng 2014]
Cheng, Yi; Deng, Julia; Li, Jason; DeLoach, Scott A.; Singhal, Anoop; & Ou, Xinming. Metrics of Security. *Cyber Defense and Situational Awareness*. Springer. 2014. 263–295. https://doi.org/10.1007/978-3-319-11391-3_13

[CISA 2023]
Cybersecurity & Infrastructure Security Agency (CISA) Minimum Requirements for Vulnerability Exploitability eXchange (VEX). April 2023. https://www.cisa.gov/sites/default/files/2023-04/minimum-requirements-for-vex-508c.pdf

[CSIAC 2023]
Cybersecurity & Information Systems Information Analysis Center (CSIAC). The DoD Cybersecurity Policy Chart. *CSIAC Website*. December 18, 2023 [accessed]. https://csiac.org/resources/the-dod-cybersecurity-policy-chart/

[CSRB 2022]
Cyber Safety Review Board (CSRB). *Review of the December 2021 Log4j Event*. Cybersecurity and Infrastructure Security Agency (CISA). July 11, 2022. https://www.cisa.gov/sites/default/files/publications/CSRB-Report-on-Log4-July-11-2022_508.pdf

[Curtis 1979]
Curtis, Bill; Sheppard, Sylvia B.; & Kuresi, Elizabeth. *Evaluation of Software Life Cycle Data from the PAVE PAWS Project*. RADC-TR-80-28. Rome Air Development Center, Air Force Systems Command. March 1980. https://apps.dtic.mil/sti/citations/ADA085323

[ENISA 2021]
European Union Agency for Cybersecurity (ENISA). Threat Landscape For Supply Chain Attacks. *ENISA Website*. July 29, 2021. https://www.enisa.europa.eu/publications/threat-landscape-for-supply-chain-attacks

[ENISA 2023]
European Union Agency for Cybersecurity (ENISA). Good Practices for Supply Chain Cybersecurity. *ENISA Website*. June 13, 2023. https://www.enisa.europa.eu/publications/good-practices-for-supply-chain-cybersecurity

[Goldstein 2023]
Goldstein, Eric. CISA Announces Secure by Design Alert Series: How Vendor Decisions Can Reduce Harm at a Global Scale [blog post]. *CISA Blog*. November 29, 2023. https://www.cisa.gov/news-events/news/cisa-announces-secure-design-alert-series-how-vendor-decisions-can-reduce-harm-global-scale

[Gray 2016]
Gray, Douglas. *Applying the Goal-Question-Indicator Metric (GQIM) Method to Perform Military Situational Analysis*. CMU/SEI-2016-TN-003. Software Engineering Institute, Carnegie Mellon University. May 23, 2016. https://insights.sei.cmu.edu/library/applying-the-goal-question-indicator-metric-gqim-method-to-perform-military-situational-analysis/

[Hissam 2002]
Hissam, S.A.; Plakosh, D.; & Weinstock, C. Trust and Vulnerability in Open Source Software. *IEE Proceedings—Software*. Volume 149. Issue 1. Pages 47–51. February 2002. https://doi.org/10.1049/ip-sen:20020208

[Hubbard 2023]
Hubbard, Douglas W. & Seiersen, Richard. *How to Measure Anything in Cybersecurity Risk*. John Wiley & Sons, Inc. April 2023.

[Ikeda 2022]
Ikeda, Scott. New Cyber Safety Review Board Report: Log4j Vulnerability Is "Endemic," Expect It to Be Exploited into the 2030s. *Chief Privacy Officer (CPO) Magazine*. July 21, 2022. https://www.cpomagazine.com/cyber-security/new-cyber-safety-review-board-report-log4j-vulnerability-is-endemic-expect-it-to-be-exploited-into-the-2030s/

[Intellias 2023]
Intellias. What We've Learned from Recent 2023 Supply Chain Attacks: Are You Prepared? [blog post]. *Intellias Blog*. October 4, 2023. https://intellias.com/supply-chain-attacks/

[Jones 2014]
Jones, Capers. *Evaluating Software Metrics and Software Measurement Practices*. Version 4. Namcook Analytics LLC. March 2014. https://docplayer.net/20369543-Evaluating-software-metrics-and-software-measurment-practices-version-4-0-march-14-2014-capers-jones-vp-and-cto-namcook-analytics-llc.html

[Jones 2023]
Jones, David. MGM Resorts Anticipates No Further Disruptions from September Cyberattack. *Cybersecurity Dive Website*. November 9, 2023. https://www.cybersecuritydive.com/news/mgm-resorts-no-disruptions-cyberattack/699320/?mod=djemCybersecruityPro&tpl=cy

[Lewis 2015]
Lewis, Dave. Heartland Payment Systems Suffers Data Breach. *Forbes*. May 31, 2015. https://www.forbes.com/sites/davelewis/2015/05/31/heartland-payment-systems-suffers-data-breach/

[Liu 2021]
Liu, Nancy. SolarWinds to Log4j: More Risk Management Wake-Up Calls. *sdx central.* December 21, 2021. https://www.sdxcentral.com/articles/news/solarwinds-to-log4j-more-risk-management-wake-up-calls/2021/12/

[Mani 2014]
Mani, Sidhartha & Mead, Nancy. *An Evaluation of A-SQUARE for COTS Acquisition*. CMU/SEI-2014-TN-003. Software Engineering Institute, Carnegie Mellon University. May 13, 2014. https://insights.sei.cmu.edu/library/an-evaluation-of-a-square-for-cots-acquisition/

[McCabe 1976]
McCabe, Thomas J. A Complexity Measure. *IEEE Transactions on Software Engineering.* Volume SE-2. Number 4. December 1976. https://www.cs.mtsu.edu/~untch/6050/private/McCabe1976.pdf

[Mead 2022]
Mead, Nancy R. Critical Infrastructure Protection and Supply Chain Risk Management. Pages 215–218. In *2022 IEEE 30th International Requirements Engineering Conference Workshops (REW)*. 2022. https://doi.org/10.1109/REW56159.2022.00047

[MITRE 2023]
MITRE Corporation. Hipcheck. *GitHub Website*. December 19, 2023 [accessed]. https://github.com/mitre/hipcheck

[Nelson 2023]
Nelson, Nate. 21 Vulnerabilities Discovered in Crucial IT-OT Connective Routers. *DarkReading Website*. November 13, 2023. https://www.darkreading.com/vulnerabilities-threats/21-vulnerabilities-discovered-crucial-it-ot-connective-routers

[O'Neill 2023]
O'Neill, Ian. *Data Breach Notifications*. Office of the Maine Attorney General. December 18, 2023 [accessed]. https://apps.web.maine.gov/online/aeviewer/ME/40/78e177b2-14ba-4d97-a8cf-eeaa3f1ba0b8.shtml?mod=djemCybersecruityPro&tpl=cy

[OSSF 2023]
Open Source Security Foundation (OSSF). Scorecard. *GitHub Website*. December 19, 2023 [accessed]. https://github.com/ossf/scorecard/issues/2296

[Rubin 2000]
Rubin, Howard A. Managing Outsourcing through Metrics. In *Winning the Outsourcing Game*. Auerbach Publications. June 2000. https://www.taylorfrancis.com/chapters/edit/10.1201/9780203997482-24/managing-outsourcing-metrics-howard-rubin

[Scala 2019]
Scala, Natalie M.; Reilly, Allison C.; Goethals, Paul L.; & Cukier, Michel. Risk and the Five Hard Problems of Cybersecurity. *Risk Analysis*. Volume 39. Number 10. Pages 2119–2126. 2019. https://pubmed.ncbi.nlm.nih.gov/30925207

[Scheff 1979]
Scheff, Benson H.; Vodges, W. B.; & Hall, N. R. *PAVE PAWS Modern Programming Data Collection System*. RADC-TR-79-137. Rome Air Development Center, Air Force Systems Command. June 1979. https://apps.dtic.mil/sti/pdfs/ADA073357.pdf

[Schwartz 2021]
Schwartz, Samantha. Log4j and the Problem with Trusting Open Source. *Cybersecurity Dive Website*. Dec. 20, 2021. https://www.cybersecuritydive.com/news/log4j-open-source-vulnerability/611784/

[Sonatype 2023]
Sonatype Incorporated. Sonatype 9th Annual State of the Software Supply Chain Report. *Sonatype Website*. 2023. https://www.sonatype.com/state-of-the-software-supply-chain/introduction

[Souppaya 2022]
Souppaya, Murugiah; Scarfone, Karen; & Dodson, Donna. *Secure Software Development Framework (SSDF)* Version *1.1*. NIST SP 800-218. National Institute of Standards and Technology (NIST). February 2022. https://csrc.nist.gov/pubs/sp/800/218/final

[Synopsys 2023]
Synopsys. 2023 Open Source Security and Risk Analysis Report. *Synopsys Website*. December 18, 2023 [accessed]. https://www.synopsys.com/software-integrity/engage/ossra/rep-ossra-2023-pdf

[Townsend 2023]
Townsend, Kevin. 98% of Firms Have a Supply Chain Relationship That Has Been Breached: Analysis. *Security Week*. February 1, 2023. https://www.securityweek.com/98-of-firms-have-a-supply-chain-relationship-that-has-been-breached-analysis/

[Tschacher 2016]
Tschacher, Nikolai Philipp. *Typosquatting in Programming Language Package Managers*. Department of Informatics, University of Hamburg. March 2016. https://incolumitas.com/data/thesis.pdf

[White House 2022]
United States White House, Executive Office of the President. *Enhancing the Security of the Software Supply Chain through Secure Software Development Practices*. OMB M-22-18. Office of Management and Budget (OMB). September 14, 2022. https://www.whitehouse.gov/wp-content/uploads/2022/09/M-22-18.pdf

[Wikipedia 2023]
Wikipedia. *Halstead Complexity Measures*. December 20, 2023 [accessed]. https://en.wikipedia.org/wiki/Halstead_complexity_measures

[Woody 2014]
Woody, Carol; Ellison, Robert; & Nichols, William. *Predicting Software Assurance Using Quality and Reliability Measures*. CMU/SEI-2014-TN-026. Software Engineering Institute, Carnegie Mellon University. 2014. https://insights.sei.cmu.edu/library/predicting-software-assurance-using-quality-and-reliability-measures-2/

[Woody 2018]
Woody, Carol; Ellison, Robert; & Ryan, Charles. *Exploring the Use of Metrics for Software Assurance.* CMU/SEI-2018-TN-004. Software Engineering Institute, Carnegie Mellon University. 2019. https://insights.sei.cmu.edu/library/exploring-the-use-of-metrics-for-software-assurance/

[Zahan 2023]
Zahan, Nusrat; Shohan, Shohanuzzaman; Harris, Dan; & Williams, Laurie. Do Software Security Practices Yield Fewer Vulnerabilities? Pages 292–303. In *IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP).* IEEE. https://doi.org/10.1109/ICSE-SEIP58684.2023.00032

Contact Us

Software Engineering Institute
4500 Fifth Avenue, Pittsburgh, PA 15213-2612

Phone: 412/268.5800 | 888.201.4479
Web: www.sei.cmu.edu
Email: info@sei.cmu.edu