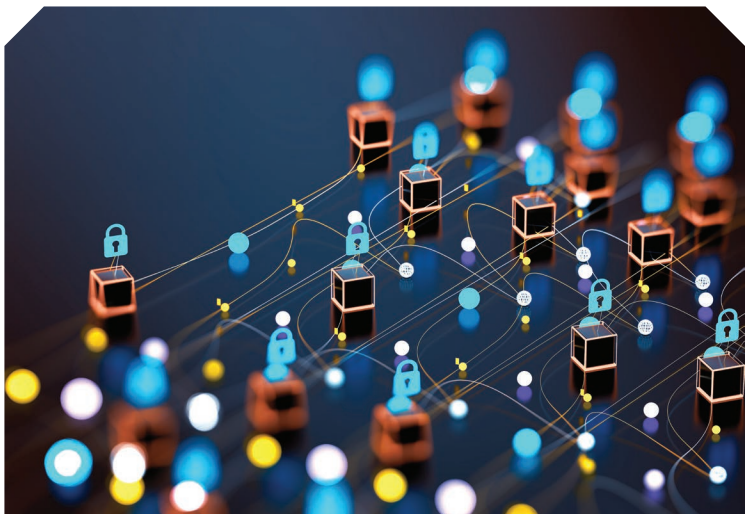
Editor: Sean Peisert, sppeisert@lbl.gov

Software Bill of Materials as a Proactive Defense

Hamed Okhravi and **Nathan Burow** | Massachusetts Institute of Technology Lincoln Laboratory
Fred B. Schneider | Cornell University

The recently mandated software bill of materials (SBOM) is intended to help mitigate software supply-chain risk. We discuss extensions that would enable an SBOM to serve as a basis for making trust assessments thus also serving as a proactive defense.

In 2020, an attacker accessing the build system of the Texas-based company SolarWinds was able to cause distribution of a malicious update to the tens of thousands of machines running the Orion end-host monitoring tool.[1] Once installed, the malicious update gave the attacker access to that customer's machines. Microsoft, Intel, and Cisco, as well as multiple U.S. government departments and agencies, including Treasury, Justice, Energy, and the Pentagon, were among the victims. The 2024 XZ Utils backdoor attack[2] in which malicious code was included in the ubiquitous XZ compression library is another prominent attack similar to SolarWinds in its vector.



©SHUTTERSTOCK.COM/CARLOS CASTILLA

The attacks on SolarWinds and XZ were classic examples of a software supply-chain attack.[3] Such an attack allows "the adversary to utilize implants or other vulnerabilities inserted prior to installation in order to infiltrate data, or manipulate information technology hardware, software, operating systems, peripherals (information technology products) or services at any point during the life cycle."[4] Software supply-chain attacks are attractive because an attacker's investments into gaining access to modify a single system are highly leveraged. Such attacks are not new or novel; a particularly insidious one was outlined 40 years ago in Ken Thompson's 1984 Turing Award lecture.[5] A modern example of this attack, the "Xcode Ghost" attack,[6] was used to compromise software in Apple's App Store in 2015. However, with so many other ways to penetrate today's software, the software supply chain had been widely considered an unlikely attack vector. Effort was more sensibly directed at blocking other kinds of attacks.

The success and scope of the SolarWinds attack changed that view, at least for U.S. government systems. Among the mitigations outlined in the subsequently issued Presidential Executive Order 14028[7] were a mandate that a software bill of materials (SBOM) would be required for any software product supplied to the U.S. government. This is not a United States-only initiative; the Cyber Resilience Act (CRA), a set of rules for security standards in the European Union, indeed has similar SBOM requirements.[8]

Digital Object Identifier 10.1109/MSEC.2025.3528535
Date of current version: 17 March 2025

As the name suggests, an SBOM for a software system describes all of the components that are part of that software system. So an SBOM for a software system provides information that enables predictions about whether attacks on the associated system would be facilitated by compromising a given supplier.

Defenders can consult SBOMs to prevent the spread of recently seen attacks (the so-called *one-day exploits*). The SBOM is thus serving as a reactive defense, coming into play only after an attack has been detected someplace. As another example of this reactive usage of the SBOM, consider how it could have helped assess the extent of exposure to the Log4Shell attack against Log4j, the popular Java logging framework back in 2021. In addition, an SBOM provides the information necessary for determining whether different software systems have components in common and, thus, could be susceptible to common-mode failures or attacks. Here the SBOM is serving as a proactive defense by providing insight into a system's resilience to attacks that have not yet been detected and perhaps not even conceived.

An SBOM—with some extensions—could become even more effective as a proactive defense if the SBOM could be used by software developers for assessing the trustworthiness of open source and third-party software components. However, the information that the Executive Order and the European Union's CRA today mandate an SBOM to include is too modest for making credible assessments about trustworthiness of this software. They require an SBOM to name the organizations that were involved in creating the components comprising a software system. That information enables some judgments about the trustworthiness of those components. But an SBOM is not required today to specify the build pipeline used to produce the software; assessments of trustworthiness would require that an SBOM incorporated (and perhaps extended) ideas from the reproducible builds movement.[9] This movement focuses on approaches that can determine whether a generated binary corresponds to the original source code or not. Reproducible builds require addressing various challenges in the small and in the large. The challenges in the large include proper recording of build information and root cause analysis for build differences, while the challenges in the small include timestamp and path differences. An SBOM today also is not required to give other information about methods (e.g., test coverage criteria or vulnerability discovery tools used) that a developer used for assurance.

Software supply-chain attacks are attractive because an attacker's investments into gaining access to modify a single system are highly leveraged.

Today's SBOM requirements have been likened by some[10] to giving a list of ingredients for a meal, in contrast to giving a recipe for cooking that meal or giving a description of how the meal should taste. By listing only the ingredients, today's SBOMs ignore the complexities of software vulnerabilities.[11] In particular, a larger software system that contains a vulnerable module might be unaffected by that vulnerability because the vulnerability might not be exploitable in the context of the larger system. Exploitability could be captured in today's SBOMs, however, using extensions such as the vulnerability exploitability exchange (VEX). VEX specifies the impact of each vulnerability (CVE) on a given product by identifying it as `affected`, `not affected`, `fixed`, or `under investigation`.

Given what information an SBOM is today mandated to provide, a natural question is: What additional information should SBOMs contain to help realize their potential as proactive defenses? That is the subject of this article. In the "Today: A Reactive Defense" section, we summarize how today's SBOMs are intended to be used as a reactive defense. Then, the "Tomorrow: A Proactive Defense" section discusses properties that additions to SBOMs should have to be useful in making trust assessments for the system an SBOM labels. Finally, the "Measures of Success" section discusses the costs and other criteria for understanding when making those additions are a good idea.

Today: A Reactive Defense

The SBOM for a software system *S* is expected to be a machine-readable record that gives the list of components and libraries included in *S*, describing certain baseline attributes for each[12]: supplier name, component name, version string, unique identifier, dependency relationship, author of SBOM, and a timestamp. The supplier name and version string are useful both for uniquely identifying the component and for deciding whether some form of mitigation is required when a supply-chain attack has been discovered. Other recommended, albeit not required, attributes may also be present in an SBOM. For example, if the SBOM includes a hash for some component, then it becomes possible to check whether the bits that will be executed are the same as the sequence of bits that the developer intended.

Abstractly, an SBOM defines a directed acyclic graph in which an edge in the graph from node *n* to node *m* indicates that the subsystem being represented by *n* is a component of the subsystem being represented by *m*. To determine the impact of a software supply-chain attack, the entire graph for a system will often be generated from the SBOMs for its subsystems. Modern tools are available to generate this transitive closure from the component relation that is documented in the SBOM for each subsystem.

The SBOM is not a new concept. The SAFECode Forum formed in 2007 by multiple "big-name" software development companies explicitly mentions the need for "maintaining a list of third-party components" in its "Managing Security Risks Inherent in the Use of Thirdparty Components" white paper, circa 2017.[13] Other guidance documents have also alluded to SBOM-like ideas. For example, the Microsoft Security Development Lifecycle Process Guidance, circa 2012[14] includes the requirement that "all code developed outside the project team (third-party components) must be listed by filename, version, and source."

What is new with Executive Order 14028 is the mandate for SBOMs to become available to end customers. Reactions have been mixed.[15] There never was any debate that an SBOM would be useful for developers who must eliminate a vulnerability or must assess the scope of the products impacted by that vulnerability. In response to the new mandate, however, some have expressed skepticism about the usefulness of SBOMs for end customers.[11] Yet, an end customer with proper resources (e.g., a medium to large company) could use an SBOM in mitigating the spread of a one-day exploit. For example, particular network ports could be closed on the enterprise firewall or certain features could be disabled to prevent immediate exploitation before proper patches are developed. Other concerns voiced about Executive Order 14028 focus on the timeline, absence of infrastructure for storing SBOMs, and missing specifics that allow different government agencies considerable flexibility in their interpretations of the new mandate.[16] The European Union initiative is not far enough along for these issues to be considered/debated.

An SBOM for a software system provides information that enables predictions about whether attacks on the associated system would be facilitated by compromising a given supplier.

SBOM generation and management tools that exist today provide a wide-range of features. This article is not intended to provide a comprehensive treatment of these tools, but for completeness we briefly mention a few. BlackDuck is a software composition analysis tool that extracts dependencies and builds SBOMs from source code, binaries, and even snippets of code. It also integrates with software development tools to automatically generate SBOMs. Synk is another developer-focused tool that performs software composition analysis on open source code. Checkmarx One, FOSSA, Syft, and the Microsoft SBOM Tool are some other prominent SBOM generation tools.

Tomorrow: A Proactive Defense

By adding information to what is today mandated for the SBOM of a system *S*, we can create transparency into the practices used to develop *S*. This transparency would enable trust assessments about the binary executable for *S*. The transparency would also provide justification for requests that a developer undertake additional testing and/or code analysis.

Consider the case of SolarWinds, wherein the build system was compromised, causing malicious binaries to be distributed without directly effecting the source code. The form of SBOM mandated today does not contain the information necessary for detecting this attack. Only if an SBOM contained additional information could the build process be replicated to create a binary executable that, if checked against an authoritative hash, would reveal that an executable is corrupted. This checking, however, presumes the use of reproducible builds,[9] as advocated in the free and open source software community. So making an SBOM into a proactive defense because it enables trustworthiness assessment not only requires incorporating additional information into an SBOM, but also adopting certain development practices.

A step beyond requiring reproducible builds would be to require reproducible tests. Taking that step would involve incorporating into an SBOM the information necessary for a developer's tests to be repeated on a delivered system. Even when those tests are not actually being repeated, having the SBOM provide this information would make it possible to assess the quality of the testing that a developer had performed. For example, having test information in the SBOM would enable the determination of the code coverage achieved by the test suite and it would allow the determination of whether

checks had been made for various common weakness enumeration categories. More generally, SBOM extensions can be a vehicle for making claims—ideally, claims that can be independently verified—about the DevSecOps practices employed in producing a software system.

Closed source software is potentially problematic for this vision of SBOM extensions, though. But such problems have been addressed in the regulatory community before. Possible solutions include the use of trusted third parties, cooperative inspections, and sampling. All of these solutions are seen today in practice by agencies in the United States, such as the Food and Drug Administration for agriculture products, Federal Aviation Administration for aircraft production, and Customs and Border Protection inspections. Other agencies in the European Union and other jurisdictions perform similar regulatory and inspection functions. Moreover, additional approaches may be possible for software by using infrastructure-as-a-service and virtualization, where continuous integration and continuous deployment pipelines, testing infrastructure, and so on could be easily shared with a trusted third party for validation.

Expanding SBOMs for Trust Assessment

The ultimate goal of using an SBOM for proactive security is to have the SBOM facilitate trust assessments of systems. A trust assessment for some system *S* will be based on the attributes of a system *S*, and these attributes can be put into different categories.

- *Intrinsic:* These are attributes that can be verified by analyzing the codebase for *S*. What programming language was used? Does the code incorporate checks for sanity of inputs, buffer overflows, etc.? Does the code comply with specific style guides or standards?
- *Developmental:* These are attributes of the environment and the process used to develop *S* and to establish assurance about its functionality. What compiler was used in creating the codebase for *S*? What hardware and software was used in the development environment? How was *S* tested (and what characterizes the test cases that were used)? What evaluation methods were employed (and what properties were verified)?
- *Reputational:* These are attributes believed to be correlated with the development of trustworthy systems. Is the company doing the development under financial stress? How trustworthy are the software producer's other products? What education or certifications do the developers have? In what country was the software produced?

What is new with Executive Order 14028 is the mandate for SBOMs to become available to end customers.

Intrinsic attributes have been widely studied in the security community. The vast literature on software testing and sanitization is, in essence, about establishing trust based on intrinsic attributes. Developmental attributes are covered by the reproducible build, DevSecOps, and software engineering communities. Reputational factors venture into socioeconomic factors that are the domain of the business and political communities. One of the challenges for SBOMs is to distill the vast amounts of work in these spaces into a readily stored and verified artifact, with results that are difficult to game.

The enumeration just given for the possible contents of a future SBOM is not intended to be exhaustive. Reasoning about the real-world security of a system is a multifaceted, complex, open problem. There is not ever likely to be a one-size-fits-all answer to the question: What information should inform a decision to trust a given system? This is because the relevance of various attributes depends on the system being evaluated, as well as on how that system will be used. So we advocate that the SBOM be seen as a flexible framework for conveying information. Certain fields might be mandated, but the SBOM also should evolve and convey other information that might be useful for making trust assessments. The community, in turn, must see the requirement to provide an SBOM as an opportunity to incentivize higher assurance for systems by creating transparency about practices in software development and analysis.

One sensible guiding principle for selecting the attributes that an SBOM conveys is to give facts, not analysis results. What analyses we can perform is likely to change with additional research; an SBOM that reports facts would not become obsolete by such research. There is, of course, a risk of having the size of an SBOM rival the size of the system, because the SBOM as mandated incorporates so much detail. That is not the intention. Rather, an initial goal might be for the SBOM to contain sufficient detail about a system so that one could determine whether an existing or new CVE should prompt a deeper dive into the code.

It is worth mentioning that a recent Defense Advanced Research Projects Agency program, Enhanced SBOM for Optimized Software Sustainment (E-BOSS), aims to enhance SBOMs

with new types of metadata and develop new cyber-reasoning capabilities. What metadata will be incorporated into the SBOM and how it is used to perform proactive defense is yet to be seen.

Measures of Success

As with any security mandate, it will be vital to understand whether an SBOM mandate is achieving worthwhile goals as opposed to being "security theater" without a meaningful impact. We have seen that the goals for an SBOM mandate can range from enabling the rapid discovery of known vulnerabilities to serving as a basis for making trust decisions. After there is a consensus on the goals for an SBOM mandate, then incentives can be identified for ensuring that all participants have a reason to work toward not only the letter of a mandate, but also the spirit.

It is impossible to conduct double-blind randomized control trials that would gauge the impact of an SBOM mandate on the obvious concerns: number of cyberattacks, amount of data leaked, economic damage, etc. Nonetheless, indications of success would include a reduction in the exploitable window for known attacks or a reduction in the total number of unknown or zero-day, attacks. Insurance companies requiring SBOMs when writing cyber policies would be another indicator of impact, and premium reductions for using SBOMs could not only incentivize the use of SBOMs but would help quantify their economic value.

A longer-term possible consequence of an SBOM mandate is to increase code quality, generally. What gets measured gets managed. To the extent that SBOMs enable the measurement of code attributes that correlate with quality, having an SBOM would enable developers and users to differentiate between better and worse code. A simple first step might be for an SBOM to report the number and recency of updates to a component, so that developers can easily check how well maintained a library is before using it.

It is important not to ignore potential abuses of SBOMs. There is a long trail of systems that are patched slowly, if ever. Consequently, an SBOM repository offers attackers a road map of known vulnerabilities to exploit. This risk can be mitigated to some extent by limiting users to accessing subsets of information relevant to them in the repository. However, history shows that access control is difficult to implement well in practice. Drawing on Sun Tzu, we believe it is critical to know yourself, and that any advantage gained by attackers is more than outweighed by the additional transparency for defenders.

Voltaire is credited with observing that "the best is the enemy of the good." Viewed through that lens, some of the suggestions we are making about uses and extensions to SBOMs can be seen as being the enemy. Critics of our suggestions will argue—rightly—that SBOMs are a clear improvement over the status quo, so the current mandate should suffice! By enumerating a system's components, an SBOM provides the information that a system owner needs to take action when one of those components has been successfully attacked elsewhere, so with SBOMs we are better off. But SBOMs deliver the "good" only with an infrastructure to manage and distribute SBOMs. Insufficient attention is being devoted to that piece of the SBOM picture; technical as well as policy challenges must be addressed.

Security measures bring overhead and inconvenience. They are best deployed when we have some adversary in mind. The U.S. government's mandate for SBOMs was introduced following a supply-chain attack and, thus, the mandate might be seen as a defense against adversaries who launch supply-chain attacks. Supply-chain attacks are expensive to perpetrate, so those adversaries will be well-resourced. However, we should also expect a well-resourced adversary to be capable of subverting compilers and other elements of a build tool chain. An SBOM can be subverted unless it has a verifiable link to the software artifact (with all its components) and to the tool chain used to build that artifact. The currently mandated SBOM does not incorporate that information. So the current mandate exhibits a fundamental weakness relative to its goals.

Finally, we have noted that SBOMs offer enormous potential beyond their use in alerting a system's owner when a successful attack elsewhere should be worrisome. In addition to its role in alerting a system owner of new vulnerabilities, an SBOM could include information needed by system owners who are making trust decisions. Moving toward support for such extensions is not in the critical path for the widespread deployment of SBOMs, so contemplating these extensions isn't an impediment to the "good." But the chances of making that progress will be much improved if they are part of our thinking today. ■

Acknowledgment

Steve Lipner and the *IEEE Security & Privacy* reviewers provided helpful feedback on an earlier version of this article. This material is based upon work supported by the Under Secretary of Defense for Research and Engineering under Air Force Contract FA8702-15-D-0001, and Air Force Office of Scientific Research under Award FA9550-23-1-0435, as well as by funding from the Massachusetts Institute of Technology Lincoln Laboratory, Amazon, and Google. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of these organizations.

References

1. S. Peisert et al., "Perspectives on the SolarWinds incident," *IEEE Security Privacy*, vol. 19, no. 2, pp. 7–13, Mar./Apr. 2021, doi: 10.1109/MSEC.2021.3051235.
2. "XZ/Utils/backdoor." Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/XZ_Utils_backdoor
3. F. Massacci and L. Williams, "Software supply chain security [Guest Editors' Introduction]," *IEEE Security Privacy*, vol. 21, no. 6, pp. 8–10, Nov./Dec. 2023, doi: 10.1109/MSEC.2023.3321189.
4. "NIST glossary." National Institute of Standards and Technology (.gov). [Online]. Available: https://csrc.nist.gov/glossary/term/supply_chain_attack
5. K. Thompson, "Reflections on trusting trust," *Commun. ACM*, vol. 27, no. 8, pp. 761–763, 1984, doi: 10.1145/358198.358210.
6. D. Goodin, "Apple scrambles after 40 malicious "XcodeGhost" apps haunt app store," *Ars Technica*, Sep. 21, 2015. [Online]. Available: https://arstechnica.com/information-technology/2015/09/apple-scrambles-after-40-malicious-xcodeghost-apps-haunt-app-store/
7. "Executive order 14028: Improving the nation's cybersecurity," *The White House*, May 12, 2021. [Online]. Available: https://www.whitehouse.gov/briefing-room/presidential-actions/2021/05/12/executive-order-on-improving-the-nations-cybersecurity/
8. T. Turner, "SBOM requirements in the EU's CRA (cyber resilience act)," *FOSSA*, Sep. 9, 2024. [Online]. Available: https://fossa.com/blog/sbom-requirements-cra-cyber-resilience-act/
9. C. Lamb and S. Zacchiroli, "Reproducible builds: Increasing the integrity of software supply chains," *IEEE Softw.*, vol. 39, no. 2, pp. 62–70, Mar./Apr. 2022, doi: 10.1109/MS.2021.3073045.
10. National Telecommunications and Information Administration. *SBOM Explainer: What is SBOM?* (Dec. 22, 2020). [Online Video]. https://www.youtube.com/watch?v=6yljBKKl8Vo&t=105s
11. S. Lipner. "Software products aren't cookies." csoonline.com. [Online]. Available: https://www.csoonline.com/article/566091/software-products-aren-t-cookies.html
12. United States Department of Commerce. "The minimum elements for a software bill of materials (SBOM)." National Telecommunications and Information Administration (.gov). [Online]. Available: https://www.ntia.doc.gov/files/ntia/publications/sbom_minimum_elements_report.pdf
13. SAFECode, "Managing security risks inherent in the use of thirdparty components," Tech. Rep., 2017. [Online]. Available: https://safecode.org/wp-content/uploads/2017/05/SAFECode_TPC_Whitepaper.pdf
14. Microsoft, "SDL process guidance version 5.2," Tech. Rep., May 2012. [Online]. Available: legacyupdate.net/download-center/download/29884/microsoft-security-development-lifecycle-sdl-process-guidance-version-5
15. W. Enck and L. Williams, "Top five challenges in software supply chain security: Observations from 30 industry and government organizations," *IEEE Security Privacy*, vol. 20, no. 2, pp. 96–100, Mar./Apr. 2022, doi: 10.1109/MSEC.2022.3142338.
16. G. Bitko, "ITI Letter to office of management and budget," Information Technology Industry Council, Washington, DC, USA, Nov. 2022. [Online]. Available: https://www.itic.org/documents/public-sector/ITILettertoOMBreM-22-18.pdf

Hamed Okhravi is a senior researcher at the Massachusetts Institute of Technology (MIT) Lincoln Laboratory, Lexington, MA 02421 USA. His research interests include systems security, security evaluation, and operating systems. Okhravi received a Ph.D. in electrical and computer engineering from the University of Illinois at Urbana-Champaign. He is the recipient of two Best Paper Awards, three R&D 100 Awards, the Federal Laboratory Consortium for Technology Transfer Excellence in Technology Transfer Award, MIT Lincoln Laboratory's Best Invention and Early Career Technical Achievement Awards, and the National Security Agency's Best Scientific Cybersecurity Paper Award. He is a Senior Member of IEEE. Contact him at hamed.okhravi@ll.mit.edu.

Nathan Burow is a technical staff member at the Massachusetts Institute of Technology Lincoln Laboratory, Lexington, MA 02421 USA. His research interests include systems security, operating systems, and binary analysis. Burow received a Ph.D. in computer science from Purdue University. His work has received two best paper awards. He is a Member of IEEE. Contact him at nathan.burow@ll.mit.edu.

Fred B. Schneider is the Samuel B. Eckert Professor of Computer Science at Cornell University, Ithaca, NY 14853 USA. His research interests include aspects of trustworthy systems, such as those that will perform as expected, despite failures and attacks. He is a fellow of the American Association for the Advancement of Sciences and the Association for Computing Machinery, and has been elected to membership of the National Academy of Engineering and the Norwegian Academy (NTV). He is a Fellow of IEEE. Contact him at fbs@cs.cornell.edu.