Editors: William Enck, whenk@ncsu.edu | Samuel King, kingst@ucdavis.edu | Angelos Stavrou, astavrou@gmu.edu

Top Five Challenges in Software Supply Chain Security: Observations From 30 Industry and Government Organizations

William Enck and Laurie Williams | North Carolina State University

Software is complex, not only due to the code within a given project, but also due to the vast ecosystem of dependencies and transitive dependencies upon which each project relies. Recent years have observed a sharp uptick of attacks on the software supply chain spurring invigorated interest by industry and government alike. We held three summits with a diverse set of organizations and report on the top five challenges in software supply chain security.

Major security incidents disrupted what were to be relaxing holiday breaks for software organizations in both 2020 and 2021. In 2020, the build process for SolarWinds's network management tool, Orion, which is used to manage routers and switches inside corporate networks, was maliciously subverted to distribute malware to create backdoors on victim's networks. This malware enabled spying on at least 100 companies and nine U.S. government agencies, including the Centers for Disease Control and Prevention, U.S. Department of Homeland Security, U.S. Justice Department, Pentagon, and U.S. State Department.

In 2021, the popular logging library log4j, used by more than 35,000 Java packages, allowed an attacker to perform remote code execution by exploiting an accidentally injected insecure Java Naming and Directory Interface lookup feature, which is enabled by default in many versions of the library. Both the SolarWinds and log4j events were driven by the software supply chain, whereby software products include "upstream" components as well as dependencies, which may be maliciously or accidentally vulnerable.

Sonatype[1] reports a 650% year-over-year increase in detected supply chain attacks (on top of a 430% increase in 2020) targeted toward upstream open source repositories. The U.S. government is so concerned by software supply chain security deficiencies that a whole section of Executive Order 14028,[2] "Improving the Nation's Cybersecurity," issued 12 May 2021, is focused on new compliance requirements

Digital Object Identifier 10.1109/MSEC.2022.3142338
Date of current version: 21 March 2022

 Copublished by the IEEE Computer and Reliability Societies

for government vendors to enhance supply chain security.

Given that software supply chain security needs "hair on fire" attention, industry and government agencies have jumped into action with both tactical and industry-wide collaborative efforts and sizable financial investments.[3] We conducted two industry and one government software supply chain security summits. The goal of these events was to enable sharing among industry practitioners having practical experiences and challenges with software supply chain security.

We intentionally kept attendance relatively small (a total of 30 organizations across the three summits) and utilized the Chatham House rule,[4] whereby participants are free to use the information received, but neither the identity nor the affiliation of the speaker(s), nor those of any other participants, may be revealed to encourage honest, intimate sharing in a trusted environment. As a result, we cannot identify the organizations in this column. By design, the summit participants were from diverse domains, company sizes, geographies, and company maturities—all from the United States—including prominent organizations who are leading industry-wide software supply chain security efforts.

In this column, we share the top five challenges in software supply chain security that we identified through running these summits. We share these to aid organizations in formulating their action plans for dealing with these issues:

1. updating vulnerable dependencies
2. leveraging the software bill of materials (SBoM) for security
3. choosing trusted supply chain dependencies
4. securing the build process
5. getting industry-wide participation.

To Update or Not To Update?— Is That Even the Question?

Challenge 1: Updating Vulnerable Dependencies

GitHub's Dependabot service notifies developers when they reference a fixed version of a dependency with a known vulnerability. Historically, developers and security experts have disagreed on whether or not fixed dependencies are a good idea.

The goal of these events was to enable sharing among industry practitioners having practical experiences and challenges with software supply chain security.

Developers like fixed dependencies: they prevent changes from breaking their project. In contrast, security experts have long touted the mantra of automatic updates, even for software dependencies. They argue that the widespread adoption of a more agile "move fast and break things" approach to software development can tolerate changes in dependencies, and it is better to have the latest version of a dependency in case there was an unannounced security fix.

SolarWinds was a wake-up call that changed the conversation around fixed dependencies. It reminded security experts that quickly updating to the latest version of a dependency might also introduce malicious code. One summit participant gave the advice that you do not want to be the first or last to update a dependency. Ideally, you want enough other people to update to the new version of the dependency to make sure it is okay. Simultaneously, you do not want to be the last because it might actually be fixing a vulnerability. You need to develop a policy that strikes this balance.

Another participant indicated that their organization is adopting stronger controls to prevent the inclusion of vulnerable dependencies. The build process now requires project maintainers to take an action within a set number of days (e.g., update the dependency or mark it as not exploitable). If an action is not taken in time, the continuous integration/continuous deployment (CI/CD) system will break the build. This policy marks a distinct change in the approach by leadership, for whom developer time has long been seen as the most important business optimization.

In contrast, another participant took a drastically different position, stating that doing "hand-to-hand combat" with individual vulnerabilities is the wrong approach. There simply are not enough human resources to make the model sustainable in the long term. The participant gave the analogy of fire management, stating that, if you spend all of your time fighting fires, you will not spend any time building and deploying fire prevention techniques. It is no coincidence that the network firewall gets its name from fire prevention.

Instead of reacting to known vulnerabilities in dependencies, we should be focusing more effort on isolation techniques that ensure a vulnerable dependency has limited impact when it is exploited. For example, Firefox has recently started deploying RLBox to do exactly this. Ultimately, we need to create a meaningful concept of "zero trust" for software dependencies.

The SBoM: What Is It Good For?

Challenge 2: Leveraging the SBoM for Security

The executive order brought the SBoM into the limelight in a big way. The SBoM is actually an old concept that is being brought to the

forefront. Over the last 10 years, a number of industrial efforts, such as SPDX, CycloneDX, and SWID,[5] have sought to standardize machine-readable formats of the SBoM for modern environments. The U.S. Cyber Supply Chain Management and Transparency Act of 2014[6] called for a bill of materials of each binary component that is used in the software, firmware, or product. Conceptually, an SBoM is just like what it sounds, a list of all of the code and build dependencies (and, ideally, version information) that went into creating a software product. A key aspect of an SBoM is to provide transparency. Assuming the SBoM is automatically created during the build process, a software consumer can remove trust in the organization providing the software.

Summit participants had widely divergent opinions on the usefulness of SBoMs. On one end of the spectrum, the requirement of sharing SBoMs among companies was considered harmful. While the concept of an SBoM is nice, the devil is in the details. Software is not always consumed in atomic ways. Developers often pull in only specific files or functions. This information must be tracked internally but is less useful to share among companies. Vulnerabilities are also context specific. Just because you use a vulnerable version of a dependency does not mean you are actually vulnerable. Ultimately, SBoMs are indirectly getting at the question of whether or not a software product has a vulnerability. Why not just require accurate and timely vulnerability information?

On the other end of the spectrum, some participants felt strongly that widespread use of SBoMs is necessary. They argued that the current software supply chain is invisible and that a lot of it only comes in during the build process (e.g., deeply transitive dependencies). It is not clear who is evaluating and reporting on this information. In contrast, SBoMs provide a way to move toward a zero-trust approach for supply chain, confidence in the unknown, and contract negotiations for risk management.

In the middle of the spectrum was the sentiment that SBoMs could be great. They are currently just a list of ingredients. However, they could contain additional evidence to trust the environment that built the software. They could allow for hash validation of all components to be compared to values in the manifest. Antivirus software could potentially use SBoMs to determine if the ingredients could meet known malware.

Instead of reacting to known vulnerabilities in dependencies, we should be focusing more effort on isolation techniques that ensure a vulnerable dependency has limited impact when it is exploited.

We left the discussion with the conclusion that, while current SBoMs are largely a compliance exercise, efforts at establishing standards and requirements for SBoMs have the potential to lay the groundwork for innovative security enhancements that leverage the SBoM. Once in place, we need to create and automate metrics that are verifiable, meaningful, nongameable, and attestationable, with the ability to demonstrate adherence to security policies.

Separating the Wheat From the Chaff: What Can Be Trusted?

Challenge 3: Choosing Trusted Supply Chain Dependencies

Ken Thompson's 1984 Turing Award talk about trusting trust[7] was brought up repeatably in one summit—"To what extent should one trust a statement that a program is free of Trojan horses? Perhaps it is more important to trust the people who wrote the software." The software supply chain is affected most at trust boundaries, for example, bringing in dependencies. By definition, every dependency is outside the trust boundary.

Readers may be familiar with the XKCD comic about software dependencies,[10] which depicts "a project some random person in Nebraska has been thanklessly maintaining since 2003" as a foundation for modern digital infrastructure. There are many takeaways from this comic. Apt to our discussion is how to establish trust with the people developing your dependencies. Can you trust the maintainers of a library over time? What about the integrity of the library's build environment or the compiler? Will an organization sell or turn over their library to someone malicious? What if a library is deleted and someone takes the name? Can the accuracy of the SBoM be trusted?

Package managers and researchers are exploring logic- and machine learning-based mechanisms for separating the wheat from the chaff. For example, tools are identifying typosquatting, so the rogue packages can be removed from repositories. Additionally, we have begun research aimed at identifying malicious packages based on package metadata, such as the presence of install scripts, maintainer accounts associated with an expired email domain, and inactive packages with inactive maintainers.[8] Currently, these and other machine learning-based sorting approaches to identify bad hygiene have a low signal-to-noise ratio and present technical challenges. Additionally, launched in August 2020,

the Open Source Security Foundation (OpenSSF), sponsored by the Linux Foundation and with governing board members from Microsoft (chair), Intel, IBM, Google, and GitHub, has several working groups and products that can aid in chaff separation and mitigation.

The principles of trust and what you can count on are not consistent across the board. Some attendees called for a science of trust.

It Takes More Than Two to SLSA

Challenge 4: Securing the Build Process

Build specifications and environments have been largely overlooked by security analysis efforts. The recent widespread adoption of popular (CI/CD) tools, such as Jenkins, Travis CI, Tekton, and GitHub Actions, provides a useful foundation for establishing documented and attestable build environments. However, they also open the attack surface for injecting malicious code during the build process. For example, a large community has developed around providing reusable GitHub actions to perform common CI/CD tasks. These GitHub actions do not always have strong access control and integrity protection.

The supply chain levels for software artifacts [SLSA (pronounced "salsa")] framework provides a checklist of standards for reasoning about the build process. SLSA is based on Google's internal processes and defines four levels, beginning with simply having a scripted build and recording provenance information and ending with using an ephemeral, isolated, parameterless, and hermetic build environment. Bonus points are given if the build is reproducible, i.e., two builds produce bit-for-bit identical output.

The summit participants were largely positive on SLSA but noted that secure build environments are a huge open-ended challenge. One participant suggested that figuring out how to secure the build environment is where the most interesting new work in the field is going to happen in the next 10 years. Essentially, where we are today with securing build environments is where we were with the secure software development lifecycle around Microsoft's Trustworthy Computing days (2002).

SLSA is a great starting place to think about the problem, but getting into all of the nooks and crannies is going to take a lot more time and effort. There are some fundamental problems, such as trusting the compiler. There is also little understanding of just how brittle systems are. For example, switching the order of including files could make something malicious or vulnerable. The documentation will also be immensely important and a potential source of attack (e.g., an action that will turn off security features). In the near term, focusing on auditability and reproducibility is a must.

Reproducible builds are a great way to know if someone is amok in your build system. There are a number of efforts on this front. For example, the Debian-initiated https://reproducible-builds.org effort has characterized and classified the many types of nondeterminism that can be introduced during the build process. It has also helped push upstream changes to compilers and tools to help in this effort. However, one participant noted that many languages just do not support the concept. They had not heard of the ability for Objective C, JavaScript, Rust, or Kotlin.

Avoiding the Tragedy of the Commons

Challenge 5: Getting Industry-Wide Participation

The big tech giants are acutely aware of the software supply chain risk and have been for some time. Some of them have created short-term solutions, such as repositories of "verified" dependencies that their developers may select from. Not only are these efforts manual intensive, but they help only that company, which may eventually incorporate an external project that was not subject to their controls. As such, efforts that contribute to the common good are needed to secure the software supply chain in the long term.

Fortunately, the major players in the industry are already coming together through the form of a number of projects. One participant noted that, if you want to know how industry is addressing the security of the software supply chain, look at the projects managed under the Linux Foundation. These include the OpenSSF (mentioned earlier), sigstore, and in-toto,[9] a joint industry–academic project that helps shed light on code-to-binary provenance. These efforts will lay the foundation for all companies to contribute to the larger need of software supply chain security.

However, it is not enough for these collaborative efforts to simply exist. They need to be adopted and used by the large majority of the software industry. This transition will not be easy, and it will take time. It is unrealistic to expect vendors to incorporate best practices overnight. Rather, it is useful to have a comparison of a given vendor to the industry as a whole. Similar efforts occurred for secure software development processes. Of note is the building security in maturity model (BSIMM), which has become the de facto standard for assessing a company's software security practices and providing an industry-wide picture of practice adoption. Similar to BSIMM, the Open Web Application Security Project (OWASP) provides the software assurance maturity model (SAMM), which can be used by organizations to assess their software

security practices and develop a plan for improving their software security posture.

Multiple summit participants called for a "BSIMM for supply chain" that can help them understand the software development and build practices they should adopt to improve supply chain security. Fortunately, based upon experiences with SAMM, an OWASP working group has proposed the software component verification standard (SCVS), a framework for identifying activities, controls, and best practices, which can help in identifying and reducing risk in a software supply chain. Organizations can obtain "But what should we be doing?" guidance from both the SLSA and SCVS frameworks. Opportunities exist for automating data collection for both of these. However, ensuring metrics are meaningful and nongameable requires significant attention.

All Hands on Deck

Several summit participants indicated that the executive order was going to force industry into adopting security practices that should have been done 20 years earlier. Sometimes, only through challenges do we make progress.

We conclude this column with a call to action based upon the summit discussions. Software development organizations can take this opportunity to improve their software development and their build processes. While the implications of the executive order may be compliance requirements for vendors, the summit attendees shared a joint desire for actually making the supply chain more secure, not just attaining compliance. They indicated a weariness toward compliance but an energy about making the supply chain more secure. They also indicated a desire for measuring whether the security of the supply chain is actually more secure: compliance measures may be leading indicators, but more desired are lagging indicators that represent actual security.

This leads to the need for more research in software supply chain security, which needs to focus on measurement, such as security measures; philosophy, including the science of trust; and technical challenges, for example, attacking the top five challenges discussed in this column. Finally, summit attendees resounded in their sentiment that educators had to teach students about software supply chain security, in particular, secure build processes. ■

References

1. "2021 state of the software supply chain," Sonatype, Jul. 2021. https://www.sonatype.com/resources/state-of-the-software-supply-chain-2021
2. "Improving the nation's cybersecurity," National Archives and Records Administration, College Park, MD, USA, Executive order 14028, May 12, 2021. [Online]. Available: https://www.federalregister.gov/documents/2021/05/17/2021-10460/improving-the-nations-cybersecurity
3. M. Kelly, "Google and Microsoft promise billions to help bolster us cybersecurity," The Verge, Aug 25, 2021. https://www.theverge.com/2021/8/25/22642054/apple-amazon-google-microsoft-cybersecurity-billions
4. "Chatham house rule," Chatham House, London, U.K., 2021. [Online.] Available: https://www.chathamhouse.org/about-us/chatham-house-rule
5. "Survey of existing SBOM formats and standards," National Telecommunications and Information Administration, Washington, DC, USA, Oct. 25, 2019.
6. "H.R.5793–Cyber Supply Chain Management and Transparency Act of 2014," Congress, Oct. 25, 2019. [Online.] Available: https://www.congress.gov/bill/113th-congress/house-bill/5793/text
7. K. Thompson, "Reflections on trusting trust," *Commun. ACM*, vol. 27, no. 8, pp. 761–763, Aug. 1984, doi: 10.1145/358198.358210.
8. N. Zahan, L. A. Williams, T. Zimmermann, P. Godefroid, B. Murphy, and C. S. Maddila, "What are weak links in the NPM supply chain?" in *Proc. Int. Conf. Softw. Eng., Softw. Eng. Pract.*, 2022.
9. S. Torres-Arias, H. Afzali, T. K. Kuppusamy, R. Curtmola, and J. Cappos, "In-toto: Providing farm-to-table guarantees for bits and bytes," in *Proc. USENIX Security Symp.*, Santa Clara, CA, USA: USENIX Association, Aug. 2019, pp. 1393–1410.
10. R. Munroe, "Dependency," XKCD, [Online.] Available: https://xkcd.com/2347

William Enck is a professor in the Department of Computer Science and codirector of the Secure Computing Institute at North Carolina State University, Raleigh, North Carolina, 27695, USA. His research interests include systems security. Enck received a Ph.D. in computer science and engineering from Penn State University. Contact him at whenck@ncsu.edu.

Laurie Williams is a distinguished university professor and codirector of the Secure Computing Institute at North Carolina State University, Raleigh, North Carolina, 27695, USA. Her research interests include software security. Williams received a Ph.D. in computer science from the University of Utah. Contact her at lawilli3@ncsu.edu.