To Whom It May Concern,

In light of the growing number of reported software supply chain attacks and to aid NIST in evaluating and improving its Cybersecurity Framework, related documents and initiatives, this comment makes three suggestions.

Suggestion #1: Emphasize an "enlightened ecosystem" approach over a "single organization" approach to software supply chain security. The enlightened ecosystem approach calls for an organization to not only understand its software dependencies, especially its open source software dependencies, but to use its financial resources and staff to actively contribute to upstream open source projects and proactively manage security at the ecosystem level.

The enlightened ecosystem approach calls on organizations to reinforce the digital infrastructure being upheld by, to quote the popular web comic *xkcd*, "some random person in Nebraska." [xkcd 2020]]The Cybersecurity Framework understandably emphasizes the perspective of one single organization such as a company or government agency. This perspective, like all perspectives, highlights some aspects of reality while omitting others. It's our view that the "single organization" approach has grown increasingly inadequate and can mislead an organization into believing that simply inventorying its software dependencies and remediating vulnerabilities is an optimal approach to software supply chain security. Calling out the benefits of an ecosystem approach is more likely to help those parties trying to make the case to their organization's leadership that they should support such efforts. It will therefore be easier for those concerned with software supply chain security efforts to join and contribute to cross-industry groups such as the Open Source Security Foundation and open source software security projects such as Sigstore or SLSA. [sigstore; SLSA]

Suggestion #2: Emphasize software integrity as a foundational prerequisite for software security. In line with the recommendations of NIST's Secure Software Development Framework, protecting software from tampering logically precedes reducing and remediating vulnerabilities.

Identifying and remediating unintentional vulnerabilities is essential work. Any vision of a secure software supply chain demands it. But no amount of hunting for unintentional vulnerabilities will protect the world's software consumers and producers from the harms of a software supply chain that lacks integrity. Should malicious parties be able to tamper with software during the development, build, or publishing process, severe harm to consumers and producers can still result. The Secure Software Development Framework version 1.1., especially its "Protect the Software" group, recognizes this

reality. Unfortunately, there have been and still exist many impediments to enforcing integrity in the software supply chain, issues sometimes overlooked because of the current focus on creating an inventory of dependencies and remediating known unintentional vulnerabilities. This concern over software integrity isn't theoretical. According to at least one dataset of software supply chain compromises, there have been, depending on the methodology, somewhere between 62 and 137 known malicious compromises of software integrity (i.e. attacks on source, build, and publishing infrastructure). [Geer et al, 2020] Projects like Sigstore, which make code signing and attestation easier, are examples of methods for improving the integrity of the software supply chain.

Suggestion #3: Promote market competition in software security by, at a minimum, calling out the harmful effect of anti-competitive "DeWitt clauses" that stifle benchmarking. Some software security product providers include terms in the license that prevent others from publishing the results of benchmarking without the tool creator's permission, harming society and international security. (Wheeler, 2021)

For instance, our organization has had to undertake its own benchmarking activities of commercial malware detection software given the lack of published results comparing effectiveness, the intended result of these anti-competitive clauses. Similarly, when it comes to software supply chain security, including both malicious compromise and unintentional vulnerabilities, there is a noticeable lack of publicly-released benchmarks comparing, for instance, commercial security static analysis or dynamic analysis tools. This lack is the inevitable result of DeWitt clauses. And this lack hurts the security of the software supply chain by slowing the spread of helpful tools and practices.

References

Geer, Dan, Bentz Tozer, and John Speed Meyers, "Counting Broken Links: A Quant's View of Software Supply Chain Compromises," *USENIX ;login:*, December 2020.

Munroe, Randall, xkcd #2347, August 2020, available at https://xkcd.com/2347/, accessed March 16, 2022.

sigstore, project website, https://www.sigstore.dev/, accessed March 16, 2022.

SLSA, project website, https://slsa.dev/, accessed March 16, 2022.

Wheeler, David A., “The DeWitt Clause’s Censorship Should be Illegal,” Personal Blog, 2021, available at https://dwheeler.com/essays/dewitt-clause.html, accessed March 16, 2022.