Comments on 86 FR 29568: SBOM Elements and Considerations

We respectfully request your consideration of Garmin International, Inc.'s comments in response to NTIA's Request for Comments within Notice 86 FR 29568 [1].

Below, we provide a direct hyperlink to relevant paragraphs under the Request for Comments within the Notice [1], followed by our corresponding comment. Additional citations within our comments are numbered in order of appearance, within square brackets (consistent with IEEE citations in technical writing).

3.d. Integrity and authenticity https://www.federalregister.gov/d/2021-11592/p-48

We advocate for making integrity hashes a recommended rather than required element of the SBOM. The hash does not add value in cases where the recipient does not have the corresponding source files, which is likely to be common for proprietary components. It also does not add value for use within an organization when a version control system such as git is used to identify the revision of the component used. We also recommend that careful attention be given to precisely specify the method to generate any integrity hash, especially in cases where the component consists of multiple files. Cryptographic hash algorithms will generate different hashes when input ordering changes. This fact complicates automated SBOM generation by any build system where compilation ordering is not completely deterministic. See also our comment below on 3.e. about limitations on trust.

3.e. Threat model

The questions posed under 3.e. are more generally about establishing trust in "any code you didn't write yourself" [2]. Establishing trust and confidence in a target software assembly (and/or the tooling used to build it) is orthogonal the primary SBOM concern of identifying the so-called "ingredients list" in said software.

Apart from the question about minimal supporting contents in the SBOM, require the build system to have its own SBOM as practicable. Build-time code generation from interface files as well as the output of compilers and assemblers are existing recognized hurdles concerning feasible depth vs. limits to establishing trust.

Apart from the minimum SBOM specification, separately work to make build systems immutable as feasible. Feasibility is likely to vary greatly by method of deployment: IoT device, desktop application, cloud SaaS, etc. As noted by [3], "Software security risks are posed by vulnerabilities rather than by method of deployment".

Keep concerns separated and focus on simplicity of the SBOM itself as the so called "ingredients list" for a target software deliverable. Keeping concerns separated will better facilitate SBOM reuse as a building block across new contexts and methods of deployment.

- SBOM for component A may serve as a dependency in a higher-level software assembly (with own SBOM B)
- Modern methods of deployment in distributed applications blur the lines between the target deliverable, its runtime configuration, and its build system.

- Overspecification would hinder SBOM usability for its primary use cases, across all methods of deployment.

3.i. Vulnerabilities

To facilitate meaningful reporting of known, unmitigated vulnerabilities within individual components

- The SBOM should identify all vulnerabilities that have been patched
- Consider allowing vulnerability dismissal by providing date of analysis

Vulnerability dismissal is a stop-gap for VEX [3] -- a longer-term effort. Near-term, dismissal should be allowed if analysis can provide assurance (better yet, evidence) that a known vulnerability within an individual component is not exploitable within a higher-level assembly. Dismissal would minimally require date of analysis.

[1] Software Bill of Materials Elements and Considerations. 86 Fed. Reg. 29568, 29571 (NTIA Notice and Request for Comment, June 2, 2001). [Online]. Available: https://www.federalregister.gov/documents/2021/06/02/2021-11592/software-bill-of-materials-elements-and-considerations [Accessed: June 9, 2021].

[2] Thompson, Ken. "Reflections on trusting trust." *ACM Turing award lectures*. 2007. 1983. https://dl.acm.org/doi/pdf/10.1145/1283920.1283940 [Accessed: June 9, 2021].

[3] J. Spring, "CERT/CC Comments on Standards and Guidelines to Enhance Software Supply Chain Security," Carnegie Mellon University's Software Engineering Institute Blog, June 1, 2021. [Online]. Available: http://insights.sei.cmu.edu/blog/certcc-comments-on-standards-and-guidelines-to-enhance-software-supply-chain-security/ [Accessed: June 9, 2021].

[4] “VEX”, DRAFT - Requirements for Sharing of Vulnerability Status Information. NTIA, October 10, 2020. [Online]. Available: https://www.ntia.doc.gov/files/ntia/publications/draft_requirements_for_sharing_of_vulnerability_status_information_-_vex.pdf [Accessed: June 9, 2021].