Cyber Supply Chain Software Integrity

Abdul Rahman
Hume Center for National Security and Technology
and Commonwealth Cyber Initiative
Virginia Polytechnic University
Arlington, Virginia 22203
abdul@vt.edu

Sachin Shetty
Virginia Modeling, Analysis and Simulation Center
Old Dominion University
Norfolk, Virginia 23529
sshetty@odu.edu

1. Introduction

A software supply chain (SSC) is the series of steps performed when writing, testing, packaging, and distributing software. Most SSCs consist of a set of integrated (i.e., chained together) steps with different purposes. In some cases, these steps serve to verify, in other cases they serve to compile. Both steps facilitate driving the project toward a completed build state in the form of a usable software product. SSC security is vital to enterprise software builds. An adversary with the ability to influence or control SSC build steps can inject malware into a final product (e.g., backdoors and / or vulnerabilities). Due to the multiple step nature of building software via SSCs, breaches are an effective way for attackers to impact many users at the same time. According to the update framework (TUF, 2021): “We can think of a software update system as ‘secure’ if: a) it knows about the latest available updates in a timely manner, b) any files it downloads are the correct files, and c) no harm results from checking or downloading files.” Securing SSCs involves implementing strategies that are preventive in nature against a volume and variety of possible attacks. Current frameworks with these characteristics claim to provide “last mile” security but may simply be enabling vulnerabilities previously embedded within products built via a compromised SSC. According to TUF, “it is possible that, by the time the package makes it to a software update repository, it has already been compromised. (TUF, 2021)”

In this position paper, we discuss the update framework (TUF) and the in-toto framework as viable methods for securing software supply chains. The proposed methods address the area **Guidelines for software integrity chains and provenance.** We will also discuss a common set of attacks being implemented today against software supply chains and how these frameworks can be utilized to prevent this. Assurance of software supply chains will also be discussed in the context of these attacks and approaches to innovate using these two frameworks as foundational elements.

2. Framework Overview

In this section, we provide an overview of two frameworks that have the potential to address the software integrity chain and provenance requirements, such as, a) Providing artifacts that describe how suppliers and vendors are confirming to software security processes, b) Provenance of software code that would allow attestation of integrity and verifiability of software code development and build processes. c) Continuous update of software processes to compute metrics that would allow evaluation of trustworthiness of vendors and suppliers.

In-toto - In-toto, is a system for securing the way in which software is developed, built, tested, and packaged (i.e., the software supply chain). In-toto attests to the integrity and verifiability of every action performed: writing code, compiling, testing, and deploying. The framework is transparent to the user regarding the steps performed (order and user). According to In-toto (2021), “the framework allows the user to verify if a step in the supply chain was intended to be performed, if the step was performed by the right actor, and attests those materials (e.g., source code) were not tampered with between steps. (in-toto, 2021)”

***TUF* -** The Update Framework (**TUF**) provides developers with the ability to protect update systems against repository compromises and attack vectors that focus on signing keys. TUF is an elegant way to issue trust information about software along with providing other meta-information about these artifacts. A core goal is to authenticate the originating provider of the data stored within the repository. In addition, TUF can validate the freshness of the artifacts along with repository consistency as these are critical steps toward overall integrity and security for SSCs. A key objective in the application of TUF is to prevent nefarious behavior that may emanate from “attackers [that have the ability to] *mix and match* software artifacts in such a way that the sum of their parts is malicious. (TUF, 2021)”

Attacks and Weaknesses: A Path toward Assurance- Known attacks and corresponding weaknesses are provided below. These can be use cases for assurance or hardening activities within SSCs. Ensuring that an SSC can withstand an attack below can remediate critical gaps present in many unprotected SSCs. Furthermore, any SSC should take these into consideration to inform a proper and secure design of their SSC. The following attacks have been provided as examples from TUF (TUF, 2021):

- **Arbitrary software installation**. Attacker delivers arbitrary files in response to download requests and install anything he wants on the client system, yet none will be detected as illegitimate.

- **Rollback attacks**. An attacker presents files to a software update system that are older than those the client has already seen. With no way to tell it is an obsolete version that may contain vulnerabilities, the user installs the software. Later, the vulnerabilities can be exploited by attackers.
- **Fast-forward attacks**. An attacker arbitrarily increases the version numbers of project metadata files in the snapshot metadata well beyond the current value, thus tricking a software update system into thinking that any subsequent updates are trying to rollback the package to a previous, out-of-date version. In some situations, such as those where there is a maximum possible version number, the perpetrator could use a number so high that the system would never be able to match it with the one in the snapshot metadata, and thus new updates could never be downloaded.
- **Indefinite freeze attacks**. An attacker continues to present files to a software update system files that the client has already seen. As a result, the client is kept unaware of new files.
- **Endless data attacks**. An attacker responds to a file download request with an endless stream of data, causing harm to clients (e.g., a disk partition filling up or memory exhaustion).
- **Extraneous dependencies attacks**. An attacker indicates to clients that, to install the software, they want, they also need to install unrelated software. This extra software may be from a trusted source but could still have known vulnerabilities that are exploitable by the attacker.
- **Mix-and-match attacks**. An attacker presents clients with a view of a repository that includes files that did not exist there together at the same time. This can result in outdated versions of dependencies being installed, and other complications.
- **Wrong software installation**. An attacker provides a client with a trusted file that is just not the one the client wanted.
- **Malicious mirrors preventing updates**. An attacker controls one repository mirror and can use it to prevent clients from obtaining updates from other, non-malicious mirrors.
- **Vulnerability to key compromises**. An attacker who can compromise the one key in a single key system, or less than a given threshold of keys, can compromise clients. These attacks can occur whether the client relies on a single online key (if only being protected by SSL) or a single offline key (if protected by most software update systems that use key signing).

3. Proposed Solution for Cyber Software Supply Chain Integrity and Provenance Assurance

A secure ecosystem can consist of TUF that delivers updates and their corresponding changes as in-toto metadata. In this respect, the contents of a software repository and its integrity is critical. Knowing that repository artifacts are reliable and trusted are necessary steps to ensure all downstream software take dependencies safely. To do this, in-toto creates a cryptographic paper trail that's very akin to cryptographically enforce-able 'Bills of Materials', driving toward necessary cryptographically authenticated paper trails all the way to the raw materials that created it (e.g., think of source code, configuration files, etc.). This produces a very strong coalition of products, that ensures everything is very tightly sealed and packaged (TUF), and one that gives you cryptographic visibility on the process that produced final software products (in-toto). The basic idea is simple: SSCs can leverage in-toto in the pipeline to create the cryptographically authenticated paper trail, and then use TUF to store all the necessary components. This can be embodied within the following steps: ***(a)*** Initialize a TUF repository**,** ***(b)*** Create an in-toto layout and register it in a special place in the TUF repository**,** ***(c)*** Conduct normal pipeline activities but create in-toto attestations that are submitted to a TUF repository**.** Finally, we can leverage Blockchain in-conjunction with cryptographically authenticated in-toto metadata to record the provenance for software and firmware at all stages of a cyber-supply chain to ensure authenticity and quality control (Shetty, 2018; Liang, 2018).

References

1. The Update Framework (TUF). (2021) Retrieved from: https://theupdateframework.io.
2. In-Toto Framework. (2021). Retrieved from: https://in-toto.io.
3. Cloud Native Computing Foundation (CNCF). (2021). Retrieved from: https://www.cncf.io/.
4. Xueping Liang, Sachin Shetty, Deepak Tosh, Yafei Ji, Danyi Li. (2018) "Towards a Reliable and Accountable Cyber Supply Chain in Energy Delivery System using Blockchain", 14th EAI International Conference on Security and Privacy in Communication Networks (SecureComm).
5. Sachin Shetty (2018), Assured Cyber Supply Chain Provenance using Permissioned Blockchain, https://cred-c.org/researchactivity/assured-cyber-supply-chain-provenance-using-permissioned-blockchain.