An analysis of the current state of technologies used to mitigate software supply chain attacks

Name: Santiago Espinosa Mooser

UID: 3035557399

Project Background

Software nowadays is built in a variety of different ways, including using frameworks, from scratch, or even as an extension of another piece of software. However, the one thing that all software has in common is that it uses libraries –code that was written, bundled and published by a third party in order to be reused in other projects. This support from third-party coders through freely available, open-source libraries and tools has allowed companies and open-source software alike to thrive, since commonly used pieces of software are written once and commonly distributed for free, instead of having to be rewritten every single time they are used. This *modus operandi* has worked well in the past due to the availability of the source code of these libraries —the code the libraries are generated from, which allow anyone to check the safety of the code. However, more often than not, the person writing the library is an external, unknown third-party rather than someone within the group or organization using the library. This has led to a potentially unhealthy reliance and trust on third-parties to not insert malicious code into the software libraries that they freely provide. That said, there have been and there continue to be attacks on software through the use of compromised or malicious libraries. These type of attacks, called software supply chain attacks, have been more and more widespread over the last decade. In 2020 alone, there were 8 major supply chain attacks[1] that caused anywhere from USD $10 million to USD $18 million worth of direct damage to the affected parties[2]. In order to prevent these attacks, many companies have been working on improving the security of their software supply chain. In particular, they are most interested in improving the risk that comes with using open source software libraries. Companies like Google, Microsoft, Snyk, and GitLab have been developing tools and processes such as Supply chain Levels for Software Artifacts (SLSA)[3], dependency scanning tools, and reproducible builds to deal with the problems that come with using open source libraries. These tools allow developers and companies alike to improve the security of their software development process. However, these new technologies and tools are not widely adopted due to their recent development. Standards like SLSA and tools like Sigstore have only been publicly available since 2021, and code version control providers such as GitLab and Github have only just started to offer support for these standards in the last year (2022)[4,5]. This lack of adoption has led to a lack of consistency in the implementations of these standards among open-source software libraries and tools. This, in turn, has and continues to cause problems both for the developer that chooses to include the software

---

[1] Roberts, Paul. “A (Partial) History of Software Supply Chain Attacks.” *Reversing Labs*, 8 June 2022, https://blog.reversinglabs.com/blog/a-partial-history-of-software-supply-chain-attacks.

[2] Satter, Raphael. “SolarWinds Says Dealing with Hack Fallout Cost at Least $18 Million.” *Reuters*, 13 Apr. 2021, https://www.reuters.com/technology/solarwinds-says-dealing-with-hack-fallout-cost-least-18-million-2021-04-13/.

[3] Ingram , Dustin. “Securing the Software Development Lifecycle with Cloud Build and SLSA.” *Google Blog*, Google, 29 July 2021, https://cloud.google.com/blog/products/application-development/google-introduces-slsa-framework.

[4] Hershkovitch, Dov. “Gitlab 15.1 Released with SAML Group Sync and SLSA Level 2 Build Artifact Attestation.” *GitLab*, GitLab, 22 June 2021, https://about.gitlab.com/releases/2022/06/22/gitlab-15-1-released/#slsa-2-attestation-included-for-build-artifacts.

[5] “SLSA-Framework/Github-Actions-Demo: Proof-of-Concept SLSA Provenance Generator for Github Actions.” *GitHub*, Https://Slsa.dev, https://github.com/slsa-framework/github-actions-demo.

library in their software product, and the end users of the software product. Accordingly, this project will shed light on the current state of the technologies used to mitigate software supply chain attacks, with a specific focus on those used in open-source software spaces.

Project Objective

As technologies used to mitigate supply chain attacks have not been widely adopted, it is hard for software developers to quickly and reliably identify when a given library or tool may create an unacceptable amount of risk to the end software product. This project aims to create a report that assesses the state of current techniques used to mitigate software supply chain attacks, as well as give feedback on how these may be improved or better implemented. This is done in order to inform the overall industry about currently adopted mitigations, as well as encourage a more effective approach to mitigate these type of attacks. The objective is to improve the understanding of why mitigating these types of attacks is important, as well as provide a framework of how a software developer can implement these mitigations. The project introduces the main areas of concern in regards to the software supply chain, as well as the various tools currently available to the open-source community as a starting point for this analysis. Additionally, as per the objective of the project, the report will present statistics on the use of specific mitigations against supply chain attacks and evaluate their effectiveness. Particularly, on open-source community software. Finally, the report will provide the author's commentary, aimed at improving the average software engineer's software supply chain security practices.

Project Methodology

In order to assess the current state of the open-source community, it's necessary to run extensive studies on the most popular software packages for different programming languages. To do this, we will download the source code for the 20 most popular packages in a number of programming languages and run tests on them. The tests will include:

1. Checking release files for digital signatures to assess the publisher's or maintainer's ability to correctly use private cryptographic keys.
2. Checking git commits for digital signatures to assess the code contributors' ability to correctly use private cryptographic keys.

3. Scanning code dependencies with dependency scanning tools to assess the project maintainers' ability to keep up with tech debt and third-party vulnerabilities.
4. (potentially) Checking whether the project has reproducible builds.[6]

In order to simplify the overall complexity of the project, the project will be implemented in python and bash, with the data being stored in a postgreSQL database. The scripts will download, parse and analyze the top 20 code repositories in various programming languages, as well as save the results of the analyses in the postgreSQL database. Whether this is implemented as a simple web application in order to make visualization easier or as a command-line utility that outputs graphs in the form of image files is yet to be decided. The programming languages chosen for the report are:

1. Javascript (npm) due to its popularity in web applications.
2. Typescript (npm) due to its recent popularity as a static-type replacement for javascript.
3. Rust (Cargo) due to its recent inclusion in the Linux Kernel.
4. Python (PyPi) due to its popularity as a scripting language.
5. Java (Maven) due to its use in financial institutions.
6. C due to its legacy importance (popularity assessed through Github downloads[7]).
7. C++ due to its legacy importance (popularity is assessed through Github downloads[7]).
8. GoLang due to its recent rise as an alternative to high-performing, statically-typed languages such as C and C++ (popularity assessed through Github downloads[7]).
9. Ruby (Gems) due to personal interest.

Additionally, although strictly not a programming language, docker has become a popular way of bundling code and software products. Therefore, 20 or more of the most popular docker containers will also be tested for:

1. Sigstore digital signatures.
2. Source code repository commit signatures.
3. Dependency Vulnerabilities.

Once the analysis has been done, the paper will report on the different trends among the different programming languages, and the code bundling utility docker.

[6] This test would have to be done manually, as there is no standard way of specifying reproducible builds

[7] Because C, C++ and GoLang do not have central package repositories, the way that packages are assessed for popularity is done through the number of downloads on GitHub: https://stackoverflow.com/questions/4338358/github-can-i-see-the-number-of-downloads-for-a-repo.

Tentative Project Schedule and Milestones

The tentative schedule of the project is as follows:

October	Finish scripts that collect statistics on the most downloaded code bundles for all 10 types of code distribution systems. Start working on scripts to download and analyze code bundles (i.e. integrate scripts with APIs). Start working on website.
November	Finish scripts to download and analyze code repositories and bundled distribution artifacts. Start collecting metadata on repositories for later analysis. Have first draft of website ready by the start of the month.
December	Finish Data collection. Start analyzing data and re-assess scope of project. Start preparing for first presentation.
January	Give first presentation. Start writing report on collected data. Start plotting statistics and publishing code used for data collection.
March	Finish final report and start working on final presentation.
April	Give final presentation. Prepare for project exhibition. (Potentially) Prepare for competition.
May	(Potentially) Compete for best FYP.

References

1. Roberts, Paul. “A (Partial) History of Software Supply Chain Attacks.” Reversing Labs, 8 June 2022, https://blog.reversinglabs.com/blog/a-partial-history-of-software-supply-chain-attacks.
2. Satter, Raphael. “SolarWinds Says Dealing with Hack Fallout Cost at Least $18 Million.” Reuters, 13 Apr. 2021, https://www.reuters.com/technology/solarwinds-says-dealing-with-hack-fallout-cost-least-18-million-2021-04-13/.
3. Ingram , Dustin. “Securing the Software Development Lifecycle with Cloud Build and SLSA.” Google Blog, Google, 29 July 2021, https://cloud.google.com/blog/products/application-development/google-introduces-slsa-framework.
4. Hershkovitch, Dov. “Gitlab 15.1 Released with SAML Group Sync and SLSA Level 2 Build Artifact Attestation.” GitLab, GitLab, 22 June 2021, https://about.gitlab.com/releases/2022/06/22/gitlab-15-1-released/#slsa-2-attestation-included-for-build-artifacts.
5. “SLSA-Framework/Github-Actions-Demo: Proof-of-Concept SLSA Provenance Generator for Github Actions.” GitHub, Https://Slsa.dev, https://github.com/slsa-framework/github-actions-demo.