Improving Trust and Providing Provenance in Software Source Code Repositories

Dustin H. Hoffman
dustin.h.hoffman@gmail.com

Source code repositories provide an appealing target for malicious actors to compromise the software supply chain. Utilizing cryptographically signed commits to source code version control systems provides confidence that only trusted parties have contributed to the source code. The technology required to enable this exists, at least in the popular version control system git, it simply needs to be integrated into the development and delivery practices for providers of critical software.

Source Code Attacks

Several attacks on open-source code repositories are publicly known. A few recent examples include:

- March 2021, servers used by the PHP project are compromised and used to insert a remote code execution (RCE) vulnerability to the source code of the PHP project. The commit appeared to correct a typo and was spoofed to look like Rasmus Lerdorf, the original creator of PHP, submitted it. Luckily, this update was discovered during routine code analysis prior to being published by the project. (Sharma, 2021)
- March 2020, GitHub announced the Octopus Scanner malware was in multiple repositories hosted on the platform. The malware attacked the NetBeans development environment and installed backdoors within the source code. Once it infected a project, it published the infected version and attempted to prevent the legitimate authors from superseding it. (Mühlberg, 2020)
- November 2018, the widely used JavaScript package event-stream introduced a dependency on a malicious package called flat-stream. This resulted in software products dependent upon the event-stream package to include the malicious source code. (Sparling, 2018)

Reports of compromises of proprietary, closed source software repositories are less well known. The compromise of a private source repository could introduce generally available vulnerabilities, (e.g., RCE) but could also introduce targeted attacks on the victim. For example, RCE defects may be defeated by effective network defense or detected by static code analysis tools. However, the attacker could introduce code to degrade or halt a service at a specific time. Such an attack may be less likely to be discovered through static code analysis or testing due to its exploitation of the victim's problem domain.

Signed Commits

It is nearly universal practice to store software source code in a version control system. A version control system tracks each incremental change to the software. When a developer updates the source code, the change is committed to the version control system, along with the identity of the developer and when the change occurred. This allows developers to coordinate distributed, concurrent work on a project, while also providing a history of changes so that defects introduced during the development can be reverted.

Version control systems also employ access control features to ensure that only authorized users are able to make changes to the source code. For example, the access control features can be used to prevent a junior developer from making a change to a software release that has not been properly vetted. The access controls can also help prevent developers without a need to know from accessing an organization's unique intellectual property.

A user is able to inspect the commit history in order to determine who made each commit. However, the history relies on the security provided by the access control of the version control system. In other words, there is no independent method to verify who made a commit. How do we know that someone did not manually rewrite the commit history? Did someone compromise a developer's account and submit a commit? Was a man-in-the middle attack performed?

Signing each commit to the version control system with a cryptographic signature can help to prevent and detect changes made by malicious actors. In a closed, proprietary environment the cryptographic signatures can ensure that changes to the repository were actually performed by the person purported to be the author. In open source repositories, changes will likely come from a wide array of sources, and these will be signed by the developers. However, tooling can identify and flag for further scrutiny submissions made by users that are not frequent, well known contributors to the project.

Version control systems used in crucial software projects should use the following features:

- **Cryptographically Sign Every Change** Every change within the version control system must be signed using a strong cryptographic hash. Ensuring that each change is signed (modifying, deleting, and moving) provides assurance of the provenance of the source code.
- **Use Secure Key Stores** Best practices must be used to ensure the keys used to sign commits are only used by the purported developers. Secure key stores and smart cards can reduce the ability of malicious actors to obtain developer's credentials.
- **Dual Signed Commits** A version control system should provide the ability for commits to be signed by two (or more) developers. Dual signing could be used for code reviews, to ensure an equivalent of two person access for high trust applications, or on teams employing extreme programming paradigms.
- **Integrate Source Signature into Signed Deliverables** Executable signing is already used to increase trust in software supply chains. Calculating a cryptographic hash of all commits used in a release and including this within the signing of executables can be used to verify the source used to build a binary.

Weaknesses

- **Tooling Updates** Some version control systems include the ability to sign commits. For example, git allows users to sign commits, but the feature is not in common use. However, features like dual signed commits and including source signatures in executables would require developments.
- **Process Resistance** Resistance by organizations and developers to change practices can reduce uptake. The extra process of distributing keys to developers is solved, but requires a non-trivial level of effort.
- **Protection of Intellectual Property** Organizations are unlikely to provide access to version control systems for the public to verify. Trusted third parties could be employed by the organizations in order to verify or audit signatures, providing assurance to Government and commercial customers.
- **Key Compromise** As with all cryptographic procedures, especially those involving humans, a significant defect is key compromise. Signing of version control commits could reuse the protections developed to protect keys used for authentication or to sign executables.

References

Mühlberg, B. (2020, June 16). *Malware Attacks on GitHub Repositories a Disturbing Development for Open Source Projects.* Retrieved from CPO Magazine: https://www.cpomagazine.com/cyber-security/malware-attack-on-github-repositories-a-disturbing-development-for-open-source-projects/

Sharma, A. (2021, March 29). *PHPs Git server hacked to add backdoor to PHP source code.* Retrieved from bleepingcomputer.com: https://www.bleepingcomputer.com/news/security/phps-git-server-hacked-to-add-backdoors-to-php-source-code/

Sparling, A. (2018, November 20). *I don't know what to say. #116.* Retrieved from GitHub.com: https://github.com/dominictarr/event-stream/issues/116