Git Signing with SSH

2020/10

Damien Miller *djm@google.com*

Vision

Every object and line of code in a git repository is cryptographically attributable back to its author.

(Ideally) via a hardware root-of trust

Why

- Cryptographic signatures prevent gross tampering
- Developer attribution prevents authorship forgery
- Attribution allows us to draw perimeters around malice
    - What else has $bad_actor committed/pushed?
- Hardware roots of trust resist theft of keys
    - Attacker may steal *use* but must maintain presence on victim’s device
- Subsequent transforms (In-Toto, trusted builds, etc) only as trustworthy as source

What is missing now?

- Widespread use of cryptographic signatures in git
  - GPG signatures are supported, but use is relatively rare
  - GPG/PGP is famously difficult to use
  - No trivial way to link GPG identities with repository identities
  - Hardware support of GPG keys requires fiddly PKCS#11
- Improved lifecycle of signatures in practical use of git
  - Preserving attribution signatures across mutations
    - Rebase, rollup, etc.

My proposal: SSH signatures in git

Teach git to use SSH keys for signing as a peer to gpg

- SSH is ubiquitous, installed by default on ~every operating system
- SSH has a simple, flat trust model
- Most developers interact with git using SSH
- Most SSH users already maintain trusted SSH keys at repository hosts
- Repository hosts already map SSH keys to user identities
- SSH signatures can easily be tied to a hardware root of trust

Implementation: signatures in SSH

- Signature mode for OpenSSH "SSHSIG"
- Implemented in OpenSSH 8.1, released 2019-10
- Retains SSH's simple trust model: *authorized_keys*-like map between identities and keys
- Support for most SSH idioms:
  - Keys resident in hardware
  - Keys resident in *ssh-agent*
  - Certificate keys

Implementation: hardware-backed keys

- OpenSSH has supported hardware-backed keys via PKCS#11 for ~10 years
  - Unfortunately, PKCS#11 is fiddly and tokens are comparatively expensive
- OpenSSH recently added support for U2F/FIDO keys (2020-02)
  - Inexpensive (<USD$10)
  - Much simpler to use
  - Touch-per-signature requirement complicates theft-of-use
- Otherwise acts almost exactly like any other SSH key type
- Supported in OpenSSH, libssh, Golang x/crypto/ssh

Implementation: generalizing git signing

- Git’s cryptography code has fairly deep assumptions of a gpg backend
- Some work began in 2019 to pry these apart
  - Linux Foundation-sponsored intern project by Ibrahim El Rhezzali
  - Mentored by David Huseby dhuseby@linuxfoundation.org
- Abstracted signing and verification interfaces
- Pluggable signature backends
- Unfortunately internship finished without the work landing
- **Dust off and commit or reimplement?**

Implementation: SSH signatures in git

- Trivial if git supports pluggable sign/verify backends
- Wider story of tying SSH identities to repository identities
  - Trivial in case of commercial repository hosts (they already do this)
  - Not difficult for self-hosted git
  - git servers already link SSH keys to accounts at some level
  - E.g. system accounts/authorized_keys
- Some questions wrt multiple signing
  - Signature coexistence if an object is signed with both GPG and SSH

Wider story for cryptography in git

- End-to-end cryptographic attribution is currently broken by mutating operations
  - E.g. rollup commits, rebase and commit, etc.
- Need plan for how to handle these operations. E.g.
  - Somehow preserve original commits
  - Countersign mutation with repository key
  - Allow attribution to peek through mutations back to original commits
- Push vs commit signatures

Vision (again)

Every object and line of code in a git repository is cryptographically attributable back to its author.

(Ideally) via a hardware root-of trust

Thanks

Google™