University of Tartu

Department of Computer Science

KSI integration with Git

Seminar on blockchain technology [MTAT.03.323]

Kristjan Jansons
Markus Lippus

Tartu 2017

Introduction

Git is a massively popular version control system used by software developers globally. It enables one to search the history of changes made during the development process and see which user has changed parts of the code. To more securely verify the identity of the developer, previously GPG authentication for git has been developed, but verification for the integrity of the actual data is lacking.

Guardtime KSI technology enables users to sign documents as to not only verify the original creator of the data, but also the integrity of the signed document. The aim of this project has been to integrate this capability with git as to enable the signing and verification of git commits to ensure the integrity of the code at any point in the development process

Overview

We implemented the project with the help of Git hooks, Catena DB and KSI Java SDK. The process had two stages: first the stage for signing the files during a commit and second the stage for verifying the files during a checkout.

The first stage for signing the files during a commit:

1. We have configured a commit hook which is activated on every commit.
2. We determine which files need to be signed.
3. The files which need to be signed are hashed using SHA-256, encoded using base64 and sent to Catena DB for signing. This is done separately for every file.
4. For every signing request made to Catena we receive back ID-s corresponding to the signatures stored in Catena DB.
5. The ID-s received from Catena are written to the end of the commit message.

The second stage for verifying the files during a checkout:

1. We have configured a checkout hook which is activated on every checkout.
2. Signature ID-s are read from the end of last commit message.
3. Signatures corresponding to signature ID-s are retrieved from Catena DB.
4. Signature base64 string is converted to byte array which is converted to KSI signature.
5. KSI signatures are used to verify the corresponding documents. Verification is performed using calendar-based verification policy using KSI Java SDK. The code for it was largely taken from KSI Java examples.
6. The result of the verification is displayed in the terminal as successful or not successful.

Usage

The necessary files for this application can be found at the bitbucket repository here. A short guide is also in the readme file at the repository.
To use this integration the hooks and the verification application must be added to the corresponding project.

- Hooks should be added at .git/hooks of the corresponding repository.
- A “config.ini” file should be created in the root directory, containing the following:
    - Guardtime username
    - Guardtime password
    - Catena DB signing service url
    - Catena DB extending service url
    - Guardtime publications file url

- Java application should be kept in the root directory of the repository.

Now every time the user types “git commit” the commit hook activates, finds all files added to the current commit, signs them with KSI and stores the signature in the Catena DB. The corresponding ID provided by the Catena DB is stored at the end of the the commit message next to the name of the file.

To verify files, users should use “git checkout” as this enables verification at current HEAD position and also imposes mandatory verification when moving the HEAD. During verification the application reports success or failure for each file. If there were errors, a corresponding message is shown, otherwise success is reported to the user.

Lessons learned

Time was spent on getting to know the capabilities of KSI Blockchain technology, rather than on programming itself. KSI blockchain would have enabled us to tackle the task in multiple ways. For example, using only the KSI SDK without Catena DB. Also, for verification we could have used key-based verification and in the future also publication-based verification once the next publication becomes available.

Also, it was very interesting to get a sense of a different kind of approach to blockchain compared to Bitcoin’s approach. Although the precise implementation is not clear from the documentation there are probably a set of distributed trusted computing units (the Core) which maintain the Guardtime Blockchain. The part which is blockchain is the calendar blockchain which is distributed from the Core to downstream. Calendar blockchain itself is a special hash tree to where data is appended after every second. The root node of the calendar blockchain is published every month in newspapers to ensure long-term integrity. This sort of implementation seems to be efficient, reliable and scalable which seemed to be a problem with some of the implementations introduced during the course.

Git hooks turned out to be very useful for tackling this task easily. Instead of using git hooks, we could have modified the git code itself to make it work, but that seemed to be out of the scope of this project which was KSI technology.

Possible further improvements

In their current form the hooks must be separately implemented for each project. These hooks and the java application could be added to git templates to initialize every project with these capabilities.

Environment variables are inaccessible to git hooks as every hook gets its own environment at runtime. A reasonable system should be devised to access the Guardtime credentials centrally without duplicating these in the hooks.

There is also the inherent problem with git hooks in that they are not compulsory but rather a suggestion to the developer. There is of course the possibility to add web hooks that would refuse commits that are not signed as required, but this assumes that webhooks can be implemented at the remote repository, which is not an option in the case of GitHub, for example.