Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Bachelor thesis

Privacy implications of exposing Git meta data

presented by

Arne Beer

born on the 21th of December 1992 in Hadamar

Matriculation number: 6489196

Department of Computer science

submitted on December 17, 2017

Supervisor: Dipl.-Inf. Christian Burkert

Primary Referee: Prof. Dr.-Ing. Hannes Federrath

Secondary Referee: Prof. Dr. Dominik Herrmann

Abstract

Even if you're not doing anything wrong, you are being watched and recorded.

Edward Snowden

Contents

Acronyms

API Application programming interface

FS file system

SHA-1 Secure Hash Algorithm 1

UTC Coordinated Universal Time

VCS version control system

Chapter 1
Introduction

Git is a code version control system which is used by most programmers on a daily basis these days. According to the Eclipse Community Survey about 42.9% of professional software developers used git in 2014 with an upward tendency [1]. It is deployed in many if not most commercial and private projects and generally valued by its users. It allows quick jumps between different versions of a project's code base and to manage and merge code from different sources to one upstream.

Several million users send new commits to their Git repositories every day. On Github alone, the currently biggest open source platform, there exist about 25 million active repositories, a total of 67 million repositories and about 24 million users [2].

Some well known projects and organizations use Git, for example Linux[3], Google[4], Adobe[5] and Paypal[6]. Every repository contains the complete contribution history of every contributing user. Each commit contains the full directory structure, a link to a blob for every file, a timestamp, a commit message from the author and more additional metadata.

This raises the question how much information is hidden in the metadata of a Git repository and which attack vectors could be introduced by mining this information, regarding a contributer or the owner of the repository.

The newly gained knowledge could be utilized by employers to spy on their employees. It could be used by an unknown attacker who aims to obtain sensitive information about a company and its employees trough their open-source projects. It is even possible that a privat person wants to monitor another person that regularly contributes to open-source repositories.

As there have not been any papers published about this specific topic or at least no public paper and Git plays such a crucial role in todays information technology, I want to investigate and evaluate this potential threat.

[1] Ian Skerret. Exclipse Community Survey 2014 Results. `https://ianskerrett.wordpress.com/2014/06/23/eclipse-community-survey-2014-results/` Retrieved Oct. 25, 2017

[2] The State of the Octoverse 2017, Retrieved Oct. 25 2011, `https://octoverse.github.com/`

[3] `https://github.com/torvalds/linux`, Retrieved Nov. 24 2017

[4] `https://github.com/google`, Retrieved Nov. 24 2017

[5] `https://github.com/adobe`, Retrieved Nov. 24 2017

[6] `https://github.com/paypal`, Retrieved Nov. 24 2017

1.1 Motivation

1.2 Leading Questions and Goals

Chapter 2
Data Aggregation

The biggest initial task for this thesis was the acquisition of data. The data should be as extensive as possible, feature a high conjunction between contributers over several repositories to verify a possible connection between those and have realistic meta data. Two different solutions existed for these requirements.

The first approach was to design an algorithm for automatic local generation of repositories. The main problem with this solution is, that the visualization and data mining code might be highly optimized for this specific generation algorithm. Real world data is noisy and inconsistent. Thereby the developed solution might have worked on the generated data, but would have probably failed on real world data.

The second solution was to get real world data from somewhere. The most obvious choice was to mine data from open source projects. I chose Github for this purpose, as it hosts one of the biggest collection of open source projects and provides a great Application programming interface (API) for querying Github's meta data. A problem with this approach is that we don't have access to all important meta data, as for example the full list of members for organizations or the internal team structure of organizations. Another problem are old email addresses, which are not related to any account anymore. Even though some ground truth is missing, I decided to use this approach as it was still the most viable and promising way to gain as much ground truth and real world noise as possible.

2.1 Structure of the Data

Before we get to the data aggregator, I want to briefly explain the internal Git storage data structure and mechanisms, which are important for the purpose of this thesis [1].

Git, as most programmers know it, is a collection of high level abstraction tools to work with it's underlying file system (FS). The most basic structure in Git is an *blob* object. A *blob* object is a file which has been added to a Git FS and is compressed and saved in the `.git/objects` directory under the respective Secure Hash Algorithm 1 (SHA-1) hash of the uncompressed file. The probability of a SHA-1 collision is really low, roughly 10^{-45}, even though Google managed to force a collision in an controlled environment in 2017 [1].

[1] Announcing the first SHA1 collision: `https://security.googleblog.com/2017/02/announcing-first-sha1-collision.html` Retrieved Dec. 16, 2017

To represent a UNIX FS or to simply bundle multiple Git *blob* objects together, Git introduces the *tree* object. A *tree* object is a file which has a SHA-1 hash reference to all underlying *blob* and *tree* objects as well as their names and file permissions. If a *tree* holds a reference to another *tree* it could be interpreted as a subdirectory.

```
100644 blob 11d1ee77f9a23ffcb4afa860dd4b59187a9104e9        .gitignore
040000 tree ac0f5960d9c5f662f18697029eca67fcea09a58c        expose
100644 blob 61b5b2808cc2c8ab21bb9caa7d469e08f875277a        install.sh
040000 tree 8aaf336db307bdcab2f082bd710b31ddb5f9ebd4        thesis
```

Listing 1: *tree* file.

Now we come to the probably most important Git feature for this thesis; the *commit*. The commit is utilized to provide an exact representation of a state of the repository's files and directories.

```
tree cd7d001b696db430b898b75c633686067e6f0b76
parent c19b969705e5eae0ccca2cde1d8a98be1a1eab4d
author Arne Beer <arne@twobeer.de> 1513434723 +0100
committer Arne Beer <arne@twobeer.de> 1513434723 +0100
```

Listing 2: *commit* file.

As you can see in listing 2, the *commit* is just another kind of file utilized by Git, which contains some meta data about a repository version:

- The reference to a *tree* object. This is practically the root directory of the Git project
- A reference to one or multiple parent commits, to maintain a version history
- The name and email address of the author
- The name and email address of the committer
- The exact commit and publish Coordinated Universal Time (UTC) timestamp with timezone

Just as the *blob* object the *tree* and *commit* files are also stored in the `.git/objects` directory under their respective hash.

With these simple methods Git manages to create a robust version control system (VCS). Git also provides tools to easily switch between commits of a project (checkout), show the changes between two different commits (diff) and to resolve conflicts between two different commits and merge them together. There are a lot more features available, but those mentioned are the most important for this project.

2.2 The Aggregator

As earlier mentioned I decided to get data from Github and wanted to utilize their API. In 200lol Github introduced the *Github APIv2*. The API is publicly available and can be used by anyone who is registered on Github and has a rate limit of 5000 requests per user per hour.

The data aggregator is written in Python and combines querying Github with cloning repositories and scanning them locally.

2.3 Problems

List of Figures

List of Listings

List of Tables

Bibliography

[1] Scott Chacon and 2nd Edition Ben Straub. *Pro Git*. Apress, 2014. ISBN: 978-1484200773.

Eidesstattliche Erklärung

„Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit im Studiengang Informatik selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel – insbesondere keine im Quellenverzeichnis nicht benannten Internet-Quellen – benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Ich versichere weiterhin, dass ich die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht.“

Ort, Datum

Unterschrift