Redmine - Defect #35186

wrong user in version logs

2021-04-29 17:10 - Wim Bertels

Status:	Closed	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:		**% Done:**	0%
Category:	Security	**Estimated time:**	0.00 hour
Target version:			
Resolution:	Invalid	**Affected version:**	3.1.3

Description

In
https://www.redmine.org/projects/redmine/repository
we see the author.

As git is also a version control system in redmine:
the logs may show the wrong author.

How to reproduce

- A new redmine instance
- With git version enabled, (maybe the same problem exists with svn, using the .subversion/servers file)
- Create 2 users: joe and donald
- Create a new project with repositories (git, svn, ..)
  - assign joe to the project (donald assigned or not: both tests can be done, for now don't assign donald)
- Clone the git repo as joe (hence use the auth credentials of joe)
- commit and push something
- normally the author in the redmine website will say joe for this commit
- now run in the locally cloned repo: $git config user.name donald
- commit and push something
- probably the author in the redmine website will say donald for this commit, even though it was authenticated and done by joe..
- what is more: the user page of donald will say he made a commit (which of course joe did, donald did not)

as there are several users who have accounts on different project management servers/services,
this also occurs when joe has a global git config user.name donald set.

Version used: 3.3.1 Stable (did not test above this version)

possibly related to #5768 ?

(the names joe and donald were chosen at random)

History

#1 - 2021-05-03 18:34 - Holger Just

- Status changed from New to Closed

- Resolution set to Invalid

This is not a security issue but is exactly how distributes version control systems (such as git or mercurial) work. Here, there is no inherent central authority which decides upon access restrictions or metadata.

Instead, with got and mercurial, each commit is first done to a local repository. Each commit will have various metadata attached to it such as the author, comitter, a timestamp, ...) This meta data can be freely set when performing the commit. Operations such a rebasing, you can even edit this later (which will produce a different commit ID). With git and mercurial, it is common to assemble commits from various sources (such as patch files, external repositories, scripts, ...) and then to push these commits to e.g. a Redmine-controlled git repository. In this case, a user who is authorized to push to a repository can push commits by any author. The author on the commits is just another metadata field which is not checked (by default) anywhere and which is just used for display purposes rather than for access checks.

With subversion, you have a different, centralized model. Here, the server-based repository is the central single point of truth. The author of a commit is necessarily the same person who "sends" the commit. As such, it makes sense to enforce the author field of subversion commits to be the authorized user who performs the commit (although you can configure the subversion server to not enforce this or to allow changing this later).

With that being said, if you truely need to strongly identify the author of a git commit later, git offers the possibility to sign commits with a PGP key. Although this information is not specifically processed by Redmine currently, it allows you to identify responsible persons later by checking the actual git repository.

Using a custom pre-receive hook on your server-side git repositories, you may also enforce further arbitrary rules. Here you could e.g. enforce that each commit is authored by currently authenticated user or that it is signed by a known PGP key. This however is outside the currently scope of Redmine.

#2 - 2021-05-04 16:04 - Wim Bertels

Holger Just wrote:

> This is not a security issue but is exactly how distributes version control systems (such as git or mercurial) work. Here, there is no inherent central authority which decides upon access restrictions or metadata.
>
> Instead, with got and mercurial, each commit is first done to a local repository. Each commit will have various metadata attached to it such as the author, comitter, a timestamp, ...) This meta data can be freely set when performing the commit. Operations such a rebasing, you can even edit this later (which will produce a different commit ID). With git and mercurial, it is common to assemble commits from various sources (such as patch files, external repositories, scripts, ...) and then to push these commits to e.g. a Redmine-controlled git repository. In this case, a user who is authorized to push to a repository can push commits by any author. The author on the commits is just another metadata field which is not checked (by default) anywhere and which is just used for display purposes rather than for access checks.
>
> With subversion, you have a different, centralized model. Here, the server-based repository is the central single point of truth. The author of a commit is necessarily the same person who "sends" the commit. As such, it makes sense to enforce the author field of subversion commits to be the authorized user who performs the commit (although you can configure the subversion server to not enforce this or to allow changing this later).
>
> With that being said, if you truely need to strongly identify the author of a git commit later, git offers the possibility to sign commits with a PGP key. Although this information is not specifically processed by Redmine currently, it allows you to identify responsible persons later by checking the actual git repository.
>
> Using a custom pre-receive hook on your server-side git repositories, you may also enforce further arbitrary rules. Here you could e.g. enforce that each commit is authored by currently authenticated user or that it is signed by a known PGP key. This however is outside the currently scope of Redmine.

that is interesting,

what i also noticed is that redmine replaces the user of the log with the full name of the user based on that information

example, given:

userid:fullname
joe: *joe here*
donald: *donald there*
both existing redmine users and

donald: *donald athome*
as a non existing redmine user, another donald, not the same as before but present in the log of the git project,

so suppose now *joe* and *donald athome* are the only ones that have made commits (git),
then the log in the redmine site will show:
donald there instead of *donald athome*
as donald is looked up, and replaced by *donald there*