Proposal for GSoC 2018

Add two-factor authentication to rubygems.org and rubygems

Personal Information

Contact

Name: QIU Chaofan

Email: fwage73@gmail.com

GitHub: @ecnelises

Education

Tongji University, Shanghai China

Software Engineering, expected graduation July, 2019

Experience

- Familiar with Ruby language and Ruby on Rails, have 2 years experience in developing Rails apps.
- Worked at Strikingly as backend internship in the second half of 2017, mainly creating a useful posting tool named 'weitie'.
- Passionate to write clean and reusable code in methodology of TDD.

About Project

Project Name

Add two-factor authentication to rubygems.org and gem command line tool

Project Description

Ideas for RubyGems - add 2FA to rubygems

The link above describes basic concepts and requirements for such project. In general, two-factor authentication (2FA) is a way to improve security level of accounts using one-time password. And the password should be generated from both users' secret key stored in server and current time.

Currently, when a user login, push gem to server and take other important actions relating to the account, only email and password are needed. These cannot safely identify users for impostors may use the logged-in device or steal the password. Adding two-factor authentication to both rubygems.org site and gem command line tool will provide extra security. That is to say, if a user do the important actions above, an extra password is needed, which can be generated by apps like Authy or Google Authenticator.

Necessary glossary

- Two-Factor Authentication (2FA). This means an extra authentication procedure that needs another code besides normal ways like password or token.
- One-Time Password (OTP). Using a static password as 2FA code cannot improve security level. There are algorithms which calculates dynamic code by an initial secret key and other information. Nowadays many mobile/desktop apps supports OTP, like Google Authenticator.
- HMAC-based OTP (HOTP), an OTP algorithm that uses HMAC digest algorithm and a counter to generate auth code. (See RFC #4226)
- Time-based OTP (TOTP), an OTP algorithm that uses current time interval (usually each interval lasts for 30 seconds) to generate code. (See RFC #6238) TOTP is often considered more secure than HOTP because the code will quickly expire.

Project Outline

In this proposal plan, the work should be finished by serveral parts.

1. Design levels and authentication workflow

   Not all actions a logged in user do should require extra authentication. A good example, npm, gives user two choices on 2FA level: **auth-only** and **auth-and-writes**. The former means extra authentication is only required when logging in and removing 2FA. The latter also requires authentication when user do everything that changes data, like publishing packages, add/remove tokens, change profile, etc.

   Here I plan to set three levels:

   1) **auth-only**, that is similar to the mode in npm. If user log in and remove 2FA, the one-time password is required.
   2) **auth-and-package**, besides occasions above, pushing new gems and change ownership of packages will also need the password.
   3) **auth-and-write**, besides occasions above, changing passwords, changing profile, resetting API key will also need the password.

   Workflow for enabling two-factor authentication should be:

   1) When logged in at rubygems.org, at profile setting page, there is link `add two-factor authentication`. Now the confirm link will be sent to registration email address. When confirmed, the secret key QR-Code and recovery code will be shown to user. Notice that the keys will not be shown twice unless user turns 2FA off. At the page, user can set 2FA authentication level.
   2) After user saves the secret key and recovery code, two-factor authentication can be used. When taking actions on the website, user will be redirected to one-time password input page if 2FA is needed.
   3) If user turns on two-factor authentication on packing publishing, when doing `gem push`, digital code is needed if logged in.

4) User can use the recovery code if lost two-factor secret key. However, the code will be regenerated after used once. (Like what GitLab does)

I almost agree with the workflow described at [RubyGems.org Issue #1252].

2. Prepare basic implementation and tests

   Contribution guideline of rubygems.org requires that every contribution should contain unit tests. And for implementing the authentication code generator, we may need external gem for it. There is some gems doing the stuff well, like 'google-authenticator' and 'two_factor_authentication'. Controller methods will be added for secret key pages and APIs, and user model also needs change. These requires respective unit tests.

   Both web pages and API need addition due to import of two-factor authentication. This stage will implement main features we listed above. And all commits should satisfy the tests.

3. Adding two-factor authentication support to gem command line tool

   People use gem command to push their gems. So the command should also be changed for new authentication APIs. This stage will test whether the command line tool works well with new API. And backward-compatibility should be maintained. That means, the new version command line tools can also works well with old gem servers that does not support two-factor authentication.

4. Summary and documents

   After all main works done, we need to write a summary file describing work during implementing this feature. And documents of rubygems.org and gem command should be added. At this stage, the code should pass CI tests and be ready to be deployed. If there is anything for more adjustment, we have time for iteration.

Final Deliverable

When the process is over, a code base that supports the two-factor authentication feature we wrote above will be merged into main branch of rubygems with all tests successfully passed.

And it will be nice to see the version with the new feature deployed when GSoC 2018 period ends.

Timeline

Period	Work
Before May 4	• Getting familiar with rubygems.org code base. • Dive into code by fixing issues. • Getting started with rubygems source code.

May 4 - May 20	• Keep communicating with the community on what form of 2FA should be delivered at the end. • Test and choose an appropriate package that does 2FA stuff or write own if needed.
May 20 - Jun 14	• Implement first prototype with tests. • Add changes to user profile controllers and models. • Add OTP code prompt to gem if server requires 2FA.
Jun 14 - Jul 13	• Keep iterating the prototype by comments from community. • Fix bugs of prototype and tests the workflow with Authenticator apps. • Write document and tutorials on 2FA usage.
Jul 13 - Aug 1	• Update the document. • Make test cases more complete.
Aug 1 - Aug 12	• Prepare to merge code into main branch. • Write final summary article on work product. • Buffer for unexpected project delay.

Milestones & Obstacle

Framework of tasks to be done is already explained at the 'project outline' section. To make it simple, I list some events which can be seen as milestone.

1. A mock site with workflow decided can be shown.
2. The site is able to enable 2FA and calculate & compare the OTP.
3. RubyGems command line tool responds to site well.
4. All tests are passed and documentation is ready.

Also, there stands some obstacles.

1. As described at Issue #1252, current token authentication method should be changed. JSON Web Token with expiration and more information is a better solution.
2. A good way of OTP generation should be chosen. It must be at least compatible with popular 2FA authentication apps, like Google Authenticator and Authy, for the HOTP and TOTP algorithm listed above. A more secure solution is named U2F based on asymmetric encryption, but few sites used it.
3. Maybe the new site should be backward-compatible, otherwise it will be strange error for users who enabled 2FA but still using old RubyGems distribution that doesn't support this feature.

Other information

About time

I live in Shanghai with timezone GMT+8. During the summer, I will be available in most weekdays and weekends, especially from late April to start of July. Before the application result announced, I can dive into the code. Although I have not read rubygems.org code before, I know how gem works and have experience at Ruby on Rails programming. So I believe the process won't be long.

For schedules at summer, I have no plan for vacation trips this year. All courses will end at June, with no examinations this semester. Currently, there is no internship for me. According to calendar of school, even the earliest internship program has to be begun at mid-July, when the main tasks of project are almost over.

Looking forward to good days participating in Ruby community and joining Google Summer of Code!

Reason for application

I encountered programming, Linux and philosophy of open-source when I was in junior high school. The story of how the 'strangers' met up on internet and built such huge software community vibrated me.

And I learnt Ruby at 2016, whose simplicity and power, from my point of view, really helps people who wants to achieve what they want. While Ruby or Ruby on Rails are not so popular around me, in proportion to its potential help to their programming experience. So doing contribution to RubyGems can be a good start for me to join the community and help others into Ruby world.

I have experience at Ruby and Rails programming. So I believe this project fits my coding level well. Although I have no commitment on open-source projects before, leak of such experience through the whole university period will be definitely disappointment to be. Whether the application be accepted or not, I will join the project as well :)

About me

Aside from coding, I enjoy music at my daily life. Not grasping a kind of musical instrument is a pity, even some people say the best instrument is human's voice. So I plan to start learning harmonica these days.

Translation and writing is my another important hobby. Translating articles, especially those on technology fields, helps spread views and knowledge further.

Considering health, my friends often ask me to do more exercise to keep fit. I once enjoyed volleyball at high school days. And I think it's time to repick it :)

Chances for underrepresented people

Actually, considering situations around me, it's hard to classify who certainly belongs to the 'underrepresented people in open-source world'. Attitude towards open-source software, can be described as three levels:

1. **Use open-source**. Currently, students now need more tools for data analysis, daily work automation, etc. A large user base is a must for

development of any open-source community. For feasible action, we can advocate open-source tools, like Ruby Mechanism or LaTeX, to people around us.

2. **Understand open-source**. It’s not easy for those who didn’t encounter the community before to do contribution by themselves. Explaining how the open-source runs and how they value newcomers’ work are a big point to them. We should let them know contributing to open-source world is not like serious employment, any user can become developer.
3. **Contribute to open-source**. A senior told me, ‘the one you contribute, the hundred community returns back’. At my university, rare students share about their experience contributing to open-source projects. This makes few people realize that this is another cool choice besides looking for an internship day by day or playing alone with their own gadgets. I hope I can be the one making a good start :)

Also, we can encourage newcomers start at work outside coding, like documentation and translation.

Actions listed above can also be carried on to help more women involved in open-source projects. But, because of social environment and education experience, women at university lack confidence in programming. So, changing the minds by holding events like Rails-Girls really helps. And correcting such justice on women’s ability on technology fields needs our united effort.