CommonRuby - Feature #15344

Being proactive about Ruby security

11/27/2018 01:10 AM - ioquatix (Samuel Williams)

Status:	Feedback	
Priority:	Normal	
Assignee:		
Target version:		

Description

I would like to start a discussion relating to https://github.com/rubygems/rubygems/issues/2496

I don't know what has been done here already. I know from a computability POV, it's impossible to solve this problem.

That being said, it seems like we could do more.

Some parts of this problem relate to tooling (like RubyGems above).

Other parts relate to isolation within the interpreter.

As code bases get bigger, security is more of a concern, both in terms of how users interact with a system, and what they can do.

Dropping all but the necessary privileges is a great way to provide security at the interpreter level.

In BSD, they recently adopted this kind of model: https://www.openbsd.org/papers/BeckPledgeUnveilBSDCan2018.pdf

It might be something which we can incorporate into Guilds.

User can create guild, pledge which APIs should be available, and then load 3rd party code.

Some code does not function well when it is isolated, Ruby often encourages monkey patching. That being said, in many large Ruby applications, business logic could be isolated. Some benefits might include reduced memory usage, improved security, etc. Apple adopted a similar policy with XPC (https://developer.apple.com/library/archive/documentation/MacOSX/Conceptual/BPSystemStartup/Chapters/DesigningDaemons.html).

For example, it might look like:

```
Guild.new do
  Guild.drop(File)
  Guild.drop(Socket)

  require 'unstrusted/3rd/party/code`
  Untrusted.all_your_base(are_belong_to_us) # Raise error or SIGABRT.
end
```

It would bring more purpose to Guild, not just for scalability, but also reliability, and security.

This is not the only way to solve this problem, I welcome any and all discussion.

Related issues:

Related to Ruby master - Feature #8468: Remove $SAFE	**Closed**	**06/01/2013**

History

#1 - 11/27/2018 03:06 AM - shyouhei (Shyouhei Urabe)

- Backport deleted (2.4: UNKNOWN, 2.5: UNKNOWN)

- Project changed from Ruby master to CommonRuby

- Tracker changed from Bug to Feature

At first glance this topic is a bit vague. I like how unveil(2) works, but honestly I have no idea if adopting that way prevents dominictarr/event-stream issue.

So at the starting point: may I ask your goal of this thread? Do you want to prevent future domincitarr/event-stream? Or you want to introduce unveil(2)-like mechanism into the core? These two might or might not be closely related.

Regarding "what has been done here already" part. Ancient versions of ruby had a feature called $SAFE. It was somewhat similar to what unveil system call is trying to achieve. In stone age when everyone invoked ruby(1) via CGI, it was very important for CGI hosting companies to prevent their customer-created CGI scripts doing something evil.

Years passed. What turned out to be difficult about $SAFE was that sandboxing done in a userland process can hardly properly work. There are simply so many entry points to cover, and 3rd-party C extensions are almost completely unable to censor before using. We (mainly matz) had considerable efforts making the mechanism sane, and eventually gave up. The raise of Rails drastically decreases the needs of CGI. Running customers' scripts inside of hosting provider's machine is not a realistic story these days. People run codes they write on their machine. Ruby's $SAFE feature shrunk to what it is now and expected for removal in a long term future.

That being said, perhaps the situation may be changing again. There are so many gems to use, depending on each other. It is practically impossible to completely understand what they do before actually using them. I understand the needs of protection from malicious activities -- not by your customers, but by some unknown gem authors. It would definitely be better for us to provide such thing if any. The history shows, through, that designing such thing is challenging, at the very least.

Further reading: #8468

#2 - 11/27/2018 03:07 AM - shyouhei (Shyouhei Urabe)

- Related to Feature #8468: Remove $SAFE added

#3 - 11/27/2018 04:06 AM - ioquatix (Samuel Williams)

From mame (Yusuke Endoh): http://ruby.11.x6.nabble.com/ruby-core-26388-suggestion-gems-ruby-lang-org-tt3595295.html

#4 - 11/27/2018 04:10 AM - ioquatix (Samuel Williams)

shyouhei (Shyouhei Urabe) Thanks for updating this issue correctly.

Thanks for providing some background on $SAFE.

I think the goal should be ensuring Ruby has the tools and processes in place to minimise issues like the ones facing NodeJS. Whether this is just policy level changes as suggested by mame (Yusuke Endoh) (e.g. having an officially sanctioned list of gems vs development gems) or involves code changes (e.g $SAFE) or both, is really something I can't answer.

That being said, there are various different approaches to sandboxing that exist outside the scope of Ruby and they might be worth investigating. For example in the case of malicious gems accessing users personal files, sandbox could help.

On the other hand, gems which load code in-process could access form submission including credit card details, require a different approach, e.g. isolation of critical and non-critical parts of code.

#5 - 12/16/2018 09:52 PM - naruse (Yui NARUSE)

- Status changed from Open to Feedback