Ruby master - Feature #5455
$SAFE should be removed
10/18/2011 04:04 AM - kosi (Motohiro KOSAKI)

<table>
<thead>
<tr>
<th>Status:</th>
<th>Rejected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority:</td>
<td>Normal</td>
</tr>
<tr>
<td>Assignee:</td>
<td>matz (Yukihiro Matsumoto)</td>
</tr>
<tr>
<td>Target version:</td>
<td>3.0</td>
</tr>
</tbody>
</table>

Description
see
ruby-dev:44554
ruby-dev:44572

Related issues:
Has duplicate Ruby master - Bug #14353: $SAFE should stay at least thread-loc...
Closed

History
#1 - 10/22/2011 01:49 PM - trans (Thomas Sawyer)
A more granular safe system would be nice.

#2 - 10/23/2011 05:21 PM - naruse (Yui NARUSE)
- Project changed from CommonRuby to Ruby master

#3 - 03/27/2012 10:46 PM - mame (Yusuke Endoh)
- Target version set to 3.0

#4 - 04/02/2012 05:17 PM - nahi (Hiroshi Nakamura)
- Assignee set to matz (Yukihiro Matsumoto)
- Target version deleted (3.0)

I guess Endoh-san postponed the decision to 3.0 dev cycle, but please allow me to try a bit more now.

If removing $SAFE in 3.0 (or something, the version incompatible with 2.0) is acceptable, is it possible to deprecate it in 2.0? Warn $SAFE usage with -w I mean.

Matz, do you think we can decide it (removal and deprecation) now?

#5 - 04/02/2012 05:29 PM - matz (Yukihiro Matsumoto)
Hi,

I don't know why you are so eager to remove it. It's not part of "the spec". It's CRuby's implementation dependent feature. Why bother?

matz.

In message "Re: [ruby-core:44060] [ruby-trunk - Feature #5455] $SAFE should be removed" on Mon, 2 Apr 2012 17:17:26 +0900, "nahi (Hiroshi Nakamura)" nakahiro@gmail.com writes:

| Issue #5455 has been updated by nahi (Hiroshi Nakamura). |
| Assignee set to matz (Yukihiro Matsumoto) |
| Target version deleted (3.0) |
| I guess Endoh-san postponed the decision to 3.0 dev cycle, but please allow me to try a bit more now. |
| If removing $SAFE in 3.0 (or something, the version incompatible with 2.0) is acceptable, is it possible to deprecate it in 2.0? Warn $SAFE usage with -w I mean. |
| Matz, do you think we can decide it (removal and deprecation) now? |
#6 - 04/02/2012 05:43 PM - nahi (Hiroshi Nakamura)

- Assignee deleted (matz (Yukihiro Matsumoto))
- Target version set to 3.0

Thanks for the swift response!

I don't know why you are so eager to remove it. It's not part of "the spec". It's CRuby's implementation dependent feature. Why bother?

Though you might be thinking I'm on a JRuby side now, my concern is not on "the spec". The reason why I want to remove $SAFE from CRuby (not the spec) is that it's hard to implement properly. CRuby applications that depends on $SAFE should check if all possible third-party C extensions honor $SAFE.

Shugo said that it should be removed from "2.0" first at [ruby-dev:44554].

That said, I agree that Endoh-san want to postpone this to 3.0 now. I'll revert my "Target version" change. But feel free to discuss about this now :)

#7 - 04/03/2012 02:07 AM - headius (Charles Nutter)

A deprecation warning would be good in any case.

Is it the position of ruby-core/MRI/Matz that $SAFE should be used for security purposes? There are a number of Rubyists (not to mention content in some Ruby books) that claim this.

However, the equivalent feature from other languages (Perl, primarily) is not intended to be used to provide a secure environment. The warnings from safe mode in those languages are intended to be advisory, used before deployment, and it is discouraged to use safe mode in production. Enforcing $SAFE as a security mechanism also requires all code everywhere to properly handle tainting and untrust...including C extensions. $SAFE/taint/untrust is just a bad way to do security.

I suggest that $SAFE should at least be deprecated in 2.0. I'm guessing that the window has closed on coming up with a "better" security replacement, but people should know that $SAFE does not provide the security guarantees they think it does.

#8 - 04/03/2012 02:59 AM - matz (Yukihiro Matsumoto)

Hi,

In message "Re: [ruby-core:44075] [ruby-trunk - Feature #5455] $SAFE should be removed" on Tue, 3 Apr 2012 02:07:15 +0900, "headius (Charles Nutter)" headius@headius.com writes:

| A deprecation warning would be good in any case.
| Is it the position of ruby-core/MRI/Matz that $SAFE should be used for security purposes? There are a number of Rubyists (not to mention content in some Ruby books) that claim this.

I have never claimed $SAFE is safe enough for strong security in any way. It's for advisory, as other languages do.

| However, the equivalent feature from other languages (Perl, primarily) is not intended to be used to provide a secure environment. The warnings from safe mode in those languages are intended to be advisory, used before deployment, and it is discouraged to use safe mode in production. Enforcing $SAFE as a security mechanism also requires all code everywhere to properly handle tainting and untrust...including C extensions. $SAFE/taint/untrust is just a bad way to do security.
| I suggest that $SAFE should at least be deprecated in 2.0. I'm guessing that the window has closed on coming up with a "better" security replacement, but people should know that $SAFE does not provide the security guarantees they think it does.

I consider this advisory useful. Why do you want to remove?

matz.

#9 - 04/03/2012 03:23 AM - matz (Yukihiro Matsumoto)

Hi,

In message "Re: [ruby-core:44062] [ruby-trunk - Feature #5455] $SAFE should be removed" on Mon, 2 Apr 2012 17:43:47 +0900, "nahi (Hiroshi Nakamura)" nakahiro@gmail.com writes:

| I don't know why you are so eager to remove it. It's not part of "the spec". It's CRuby's implementation dependent feature. Why bother?
| Though you might be thinking I'm on a JRuby side now, my concern is not on "the spec". The reason why I want to remove $SAFE from CRuby (not the spec) is that it's hard to implement properly. CRuby applications that depends on $SAFE should check if all possible third-party C extensions honor $SAFE.
Shugo said that it should be removed from "2.0" first at [ruby-dev:44554].

That said, I agree that Endoh-san want to postpone this to 3.0 now. I'll revert my "Target version" change. But feel free to discuss about this now :)

It's still useful for advisory. Especially $SAFE=1. Since I don't trust myself, I don't claim $SAFE=4 is secure. So I can agree with removing $SAFE=4 feature.

Do you have any plan for "replacement"?

matz.

---

#10 - 04/16/2012 03:21 AM - headius (Charles Nutter)

My plan at the moment (in rough form) is to break out the individual restrictions the SAFE levels are intended to govern and allow controlling them via Java security policies. I already implemented one as a prototype, to permit evaluation of code (Java security policies are whitelists, not blacklists...another reason they do a good job of security):

https://github.com/headius/jruby/commit/b8f17f21f083207612bc234ab022b2a07a9b5e11

It should be possible to implement all the SAFE security restrictions this way, but the result will be more flexible (since users can mix and match features), more explicit, and in JRuby's case part of standard Java security policy management.

I would suggest that Ruby 2.0 put together a list of all restricted operations and form a similar security system to the JVM. I am willing to help with that.

#11 - 04/19/2012 01:35 PM - jballanc (Joshua Ballanco)

I just wanted to chime in here and suggest that, in the process of adding security restrictions, it might be worth considering the Sandbox implemented in MacRuby and Aaron's playpen library (https://github.com/tenderlove/playpen), both of which are built on the OS-level security framework. I wonder if SAFE might better be replaced by something like this (built on OS specific security frameworks)?

A problem that I see with this approach is that it would be hard to support this consistently across a variety of platforms. I think a more consistent approach would be to define an independent, abstract interface like Charles suggested. Then the individual implementations could very well use OS-specific helpers to realize the spec, while JRuby is still free to piggyback on Java's built-in features. Personally, I believe these kinds of abstraction layers help a lot to keep consistence and encourage a more testable, cleaner overall design.

-Martin

#12 - 04/20/2012 12:53 AM - MartinBosslet (Martin Bosslet)

On Apr 19, 2012 6:36 AM, "jballanc (Joshua Ballanco)" jballanc@gmail.com wrote:

Issue #5455 has been updated by jballanc (Joshua Ballanco).

I just wanted to chime in here and suggest that, in the process of adding security restrictions, it might be worth considering the Sandbox implemented in MacRuby and Aaron's playpen library (https://github.com/tenderlove/playpen), both of which are built on the OS-level security framework. I wonder if SAFE might better be replaced by something like this (built on OS specific security frameworks)?

#13 - 04/28/2012 12:25 PM - headius (Charles Nutter)

In an effort to be constructive here, I will attempt to break out specific, concrete permissions revoked at each SAFE level (or conversely, granted as SAFE levels are reduced). I base this on the publicly available edition of Programming Ruby (http://www.ruby-doc.org/docs/ProgrammingRuby/) under the chapter "Locking Ruby in the Safe", since I do not have a current copy handy.

I attempt to phrase these as they would be described in security documentation, with a proposed "permission" name for illustrative purposes.

These permissions would NOT necessarily be grouped as in the SAFE levels, other than for SAFE level compatibility. The user would be able to mix and match them at will, as with Java security policies.

Some permissions may imply other permissions must be granted.

I also omit restrictions relating to tainting, since I believe tainting is a fundamentally flawed mechanism of security.

From SAFE level 1:
OptionsFromEnvironment - whether RUBYLIB and RUBYOPT are observed
SearchCurrentDirectory - whether . is added to the load path. In 1.9 this one is moot, since . is not added to load path by default
NonlocalScripts - whether -e -i -l -r -s -S and -x options are observed
ExecFromWorldWritable - whether external commands can be executed if the directory containing them is world-writable
EvalString - whether arbitrary strings can be evaluated. Either allow strings or do not; don't rely on tainting bits for "safety".
LoadFile - whether external files can be loaded. Same argument regarding tainting.
QueryIOStatus - whether IO channels' statuses can be queried or modified. (I'm not quite sure what "status" means in this context)
ExecCommand - whether external commands can be executed.
SignalTrap - whether signals can be trapped and handled.

From SAFE level 2:
- ManipulateDirectories - whether directories can be modified, changed into, or chrooted. Note that Java provides for file/dir permissions to be granted on a per-file or per-hierarchy basis, which is much more flexible. You grant access to what you want to allow access to.
- LoadFromFileWritable - whether files can be loaded from directories that are world-writable.
- LoadFromFileHomeBase - whether files can be loaded from paths starting with ~.
- QueryFileStatus - whether methods like File.stat, File.umask can be used.
- ModifyFileStatus - whether methods like File.chown, File.chmod can be used. Java may have a better grouping of these operations, I'm not sure.
- ModifyProcess - whether methods like Kernel#fork, Process.setpgid, etc can be used.
- ModifyObject - whether Object class can be modified in any way. Perhaps includes top-level?
- ModifyInstanceVariablesExternally - whether instance variables can be modified or removed via instance_variable_set and friends.
- ModifyThread - whether currently running threads can be modified in any way.
- ModifyThreadLocalVariable - whether thread-local variables can be modified.
- TerminateThread - whether threads can be terminated from outside the thread itself.
- ModifyThreadGroup - whether threads can be removed or added to thread groups.
- ExitProcess - whether the current process can be terminated.
- IncludeModule - whether modules can be included into existing class hierarchies.
- WalkObjects - whether methods like each_object or _id2ref are allowed.
- WriteToIO - whether writes to IO channels are allowed.
- DefineAutoload - whether new autoloads can be defined.

From SAFE level 3:
NONE..the only restrictions at this level are related to tainting.

From SAFE level 4:
- ModifyGlobalVariable - whether global variables can be updated.
- AccessInstanceVariableExternally - whether methods like instance_variable_get can be used.
- ModifyEnvironmentVariable - whether ENV changes are permitted.
- CloseFile - whether already open files can be closed.
- OpenFile - whether new files can be opened.
- FreezeObject - whether objects not already frozen can be frozen.
- AlterVisibility - whether existing methods can have their visibility changed.
- AliasMethod - whether new method aliases can be defined.
- QueryMetadata - whether method and variable lists can be queried.
- DefineMethod - whether methods can be defined, redefined, removed, or undef'ed.
- ModifyObject - whether the Object class can be modified in any way. Perhaps includes top-level?
- ModifyInstanceVariablesExternally - whether instance variables can be modified or removed via instance_variable_set and friends.
- ModifyThread - whether currently running threads can be modified in any way.
- ModifyThreadLocalVariable - whether thread-local variables can be modified.
- TerminateThread - whether threads can be terminated from outside the thread itself.
- ModifyThreadGroup - whether threads can be removed or added to thread groups.
- ExitProcess - whether the current process can be terminated.
- IncludeModule - whether modules can be included into existing class hierarchies.
- WalkObjects - whether methods like each_object or _id2ref are allowed.
- WriteToIO - whether writes to IO channels are allowed.
- DefineAutoload - whether new autoloads can be defined.

This is just a rough example from 15 minutes of flipping the safe level documentation around a little. It's obviously not all-encompassing; there are things you might expect an attacker to do in untrusted code that aren't covered here. But hopefully it shows how the current safe levels could be broken into finer-grained permissions that users can assemble in any way they choose, and which different implementations can implement either in terms of VM restrictions or OS-level restrictions.

#14 - 02/14/2014 10:04 AM - akr (Akira Tanaka)
- Description updated

#15 - 06/12/2015 06:50 AM - hsbt (Hiroshi SHIBATA)
- Description updated

#16 - 06/12/2015 07:06 AM - matz (Yukihiro Matsumoto)
CRuby 2.1 has removed $SAFE=4 which is half-baked sandboxing. After serious consideration and discussion, we decided to remove $SAFE=3 (tainting mode) and $SAFE=2 (process operation prohibition) in CRuby 2.3. Of course other implementation can ignore $SAFE at all.
Matz.

#17 - 06/12/2015 11:30 PM - hsbt (Hiroshi SHIBATA)
- Assignee set to hsbt (Hiroshi SHIBATA)

#18 - 06/18/2015 02:22 PM - hsbt (Hiroshi SHIBATA)
removed $SAFE=2 at r50958, and $SAFE=3 at r50932
We should warn to use $SAFE = 1 at Ruby 2.6 for Ruby 3.0.

Assignee changed from hsbt (Hiroshi SHIBATA) to matz (Yukihiro Matsumoto)
Status changed from Open to Assigned

Status changed from Assigned to Rejected

The latest decision by Matz (making $SAFE no effect in 2.7) was made in #14353. So closing this ticket to prefer that ticket as a log.

Has duplicate Bug #14353: $SAFE should stay at least thread-local for compatibility added