Description

CRuby over time moves more and more code to default gems, to bundled gems and or even stops shipping some gems which used to be stdlibs (#5481).

I believe the main motivation for that is being to fix security issues without needing to make a CRuby release, and that makes sense.

There are however multiple unwanted side effects of this:

1. Removing gems from stdlib (e.g., #17873) is a breaking change, which makes upgrading to a new Ruby version more difficult. I think this should only be done if there is a clear gain. Being a default gem is already enough to fix a security issue without a CRuby release.

2. When any gem depends on a default gem, it tends to break on all Ruby implementations except CRuby, and for older Ruby versions.

Let's focus on this second point.

There can be good reasons to depend on a specific version (or ~>/>=) of a default gem, for instance to ensure a given security issue is addressed.

In other cases, I think there is no value to depend explicitly on a default gem, it would work without an explicit dependency since it is still "in stdlib".

And it is actually harmful to depend on default gems for JRuby, TruffleRuby and older Ruby versions, because the default gem does typically not work there yet, but the stdlib works just fine!

The reason for that is simple, those gems used to be stdlib and so were implemented directly in the Ruby implementation.

Also depending on default gem will typically be redundant with what's in stdlib, which is then a waste of network, time and disk.

For larger default gems (e.g., openssl), I believe the solution is those gems to support JRuby, TruffleRuby, etc. This is useful so the behavior for a given version of the gem is compatible between Ruby implementations, has the same security fixes, etc.

It is however a large effort and overhead to do this, and it only makes sense if people are going to need to depend on such a gem explicitly (either for security or new features in a given version), otherwise the version in stdlib is good enough and much simpler.

Here are I think some clear cases of default gems which are clearly more overhead than what they gain:

- io-wait: just a few methods very tight to IO internals, should really be core
- io-nonblock just a few methods very tight to IO internals, should really be core
- digest: has a public header and so versioning it doesn't work. Also it makes sense to reuse e.g. MessageDigest on JVM for better performance.
- strscan: this accesses a lot of Regexp internal, it would fit better in each implementation repo as a non-gem stdlib.

These are all small, they are all fairly stable, and it's unclear why they are even default gems in the first place. They also seem fairly unlikely to have security issues.

So this is what I propose:

- Do not depend on default gems in stdlib gems unless necessary (for security or feature), or unless we know the next version of Ruby will no longer ship that gem. An example is net-protocol depending needlessly on io-wait, I'll make a PR for that.
- I think those gems listed just above should no longer be default gems in the future to clarify the situation. They should either be core or regular non-gem stdlib.

Related issues:

Related to Ruby master - Feature #18566: Merge 'io-wait' gem into core IO
Here is a practical example of needless dependencies on default gems causing JRuby to not be able to install e.g. net-imap, even though JRuby has the needed stdlibs:
https://gist.github.com/eregon/5ea86a9625fc32844c81c5f1448d02d0

Here is a set of PR to solve the issue for JRuby and TruffleRuby for net-* gems:
https://github.com/ruby/net-protocol/pull/6
https://github.com/ruby/net-imap/pull/56
https://github.com/ruby/net-pop/pull/10
https://github.com/ruby/net-smtp/pull/38

I think this is an easy step in the right direction, and this lets those gems just work on JRuby and TruffleRuby.

Another issue with upstreaming code to default gems is some default gems need many internals and those should not be gems, they should be core or stdlib.

For TruffleRuby for instance, I don't want to expose many internals to support strscan as a default gem.
As an example, this code for strscan is in TruffleRuby and it should remain there, things like Primitive make no sense in public gem code:
https://github.com/oracle/truffleruby/blob/eb1f5b4c4510f105fe6c88d58f1252272738842b/lib/truffle/strscan.rb
strscan depends on many details of the Regexp implementation, so it's not reasonable to leak all these details to the gem, and doing so would hurt maintainability of that code.

I think this is an easy step in the right direction, and this lets those gems just work on JRuby and TruffleRuby.

Another issue with upstreaming code to default gems is some default gems need many internals and those should not be gems, they should be core or stdlib.

For TruffleRuby for instance, I don't want to expose many internals to support strscan as a default gem.
As an example, this code for strscan is in TruffleRuby and it should remain there, things like Primitive make no sense in public gem code:
https://github.com/oracle/truffleruby/blob/eb1f5b4c4510f105fe6c88d58f1252272738842b/lib/truffle/strscan.rb
strscan depends on many details of the Regexp implementation, so it's not reasonable to leak all these details to the gem, and doing so would hurt maintainability of that code.

I think this is an easy step in the right direction, and this lets those gems just work on JRuby and TruffleRuby.

Another issue with upstreaming code to default gems is some default gems need many internals and those should not be gems, they should be core or stdlib.

For TruffleRuby for instance, I don't want to expose many internals to support strscan as a default gem.
As an example, this code for strscan is in TruffleRuby and it should remain there, things like Primitive make no sense in public gem code:
https://github.com/oracle/truffleruby/blob/eb1f5b4c4510f105fe6c88d58f1252272738842b/lib/truffle/strscan.rb
strscan depends on many details of the Regexp implementation, so it's not reasonable to leak all these details to the gem, and doing so would hurt maintainability of that code.

I think this is an easy step in the right direction, and this lets those gems just work on JRuby and TruffleRuby.

I believe the correct approach is to actually specify the dependencies and specify them always and everywhere. Yes, that will break something somewhere, but once everything is gemifie and all dependencies are set, the problem is solved.

The opposite case is quite dangerous. Lets say you have application A which depends on default gem Z without specifying the dependency, and it explicitly depends on gem B, where B depends on gem C. Now if gem C grows explicit dependency on a gem Z of lets say some older version, the whole application A might be e.g. broken by some API change or even vulnerable. Whereas if the application A specified explicitly the dependency on Z and taken care about its dependencies, this new dependency in its dep chain would have no influence.

I know one should audit the dependencies, but there is BUT I am afraid.

These gems are default gems and so available by default, and for the foreseeable future they will always be shipped with Ruby, so there is no risk to not being super explicit about these dependencies (and most code out there using them does not specify the dependencies to stdlib, there is no point and it breaks things needlessly).

These gems are default gems and so available by default, and for the foreseeable future they will always be shipped with Ruby, so there is no risk to not being super explicit about these dependencies (and most code out there using them does not specify the dependencies to stdlib, there is no point and it breaks things needlessly).

Whether those smaller libraries really deserve to be gems is a separate debate. I am generally in favor of merging io-wait and io-nonblock directly into core IO, but I'm less convinced about digest and strscan.
These gems are default gems and so available by default, and for the foreseeable future they will always be shipped with Ruby, so there is no risk to not being super explicit about these dependencies (and most code out there using them does not specify the dependencies to stdlib, there is no point and it breaks things needlessly).

Several more arguments in favor of specifying dependencies:

1) Not specifying dependencies prevents major changes. If there is default gem A of version 1.5.2 and there should be major update in future version of Ruby, lets say to the version 2.0.0. If some gem specified the dependency on A ~> 1.0, there would not be problem, it could keep using the old version. But your proposal is going against this, so I am not sure what would be the suggestion for this case.

2) WEBrick used to be part of StdLib while it is not anymore. It would not be problem if every app/library specified this dependency prior the removal, but you suggest against it.

So yes, I agree that the starting line, where there were no dependency specifications, is quite unfortunate. But the focus should be on specifying them instead of their removal. The focus should be on making the default and bundled gems less special instead of giving them some exceptions and false assumptions about their availability. Because the worst case is the current mixed situation.

I agree that io-wait at a minimum should not be gemified. Perhaps digest. I think this is a different argument than the overall issue you raised.

I disagree that gems made default should not listed. My main scenario is you have a project that uses a gem which depends on a default gem. You bundle and get a .lock file which gives you the ability to reproduce your dependencies...except...for default gems. If there is a security release for a default gem then the gem you depend on may change and your project no longer has any record of which version you actually used. You effectively lost your ability to reproduce a build.

This is more important if you are a Ruby user who has n release branches and some of those branches are maintained over longer periods. You do not want to try and figure out what gems you are using after the fact.

The second scenario is realistic but probably less common. MRI decides some default gems will just become ordinary gems (and removed from the distribution). This has already happened for pieces of stdlib. By not having a transitive dep then any application or library which depends on it will stop working in that release.

I said it in other places too but It's also my opinion that dependencies on default gems should be explicit, for the reasons stated by others.

It's very unlikely those parts of stdlib would have major changes, and even if they do feature checking is always better to support more Ruby versions (e.g., respond_to?). Many scripts/programs/gems out depend on these 4 stdlibs, they do not specify the dependencies on such basic default gems, and the majority of them will not specify such dependencies needlessly in the future either.

And if a feature of a specific version of that default gem is needed, then specifying the dependency makes sense, but there is no need currently.

I believe these 4 gems will never become external.

But even if at some point we want to make them external, we can add the dependencies then. That will be very few changes compared to the many users that would need to adapt their program for the removal of such a default gem from stdlib.

I said it in other places too but It's also my opinion that dependencies on default gems should be explicit, for the reasons stated by others.

The vast majority of gems out there (is there even one popular non-stdlib gem doing that?) don't specify needless dependencies on default gems, so it really doesn't seem worth it for just a few stdlib gems to do it.

Without the explicit dependency the version used is the well known one in stdlib, and all of stdlib might be/is used without explicit dependencies.

There is no need or gain to have dependencies on these gems now, especially with an unbounded version requirement. And it actually hurts compatibility as explained.
But somehow people think it's worth sacrificing compatibility just to sometimes have a default gem version in a lockfile when one of these stdlib gems is used?

#15 - 02/03/2022 11:32 AM - deivid (David Rodríguez)
As an example of 1), I run into trouble with bigdecimal 1.x vs 2.x due to missing proper dependency specifications.

Just to be clear, if in this particular case, for pragmatism, it is considered best that these dependencies should be removed, that's fine with me, but I don't think that warrants making this a general recommendation reading “Depending on default gems when not needed considered harmful”.

#16 - 02/03/2022 11:36 AM - deivid (David Rodríguez)
Also I recall issues with the spring gem that were fixed by correctly specifying their dependency on default gems like tmpdir. I could probably find more.

And again, I do understand the argument made that some stuff is not worth gemifying, but once it's decided that something should become a gem, it should be treated as a regular gem as much as possible.

#17 - 02/03/2022 11:51 AM - Eregon (Benoit Daloze)
- Subject changed from Depending on default gems when not needed considered harmful to Depending on default gems in stdlib gems when not needed considered harmful

deivid (David Rodríguez) wrote in #note-15:

> Just to be clear, if in this particular case, for pragmatism, it is considered best that these dependencies should be removed, that's fine with me, but I don't think that warrants making this a general recommendation reading “Depending on default gems when not needed considered harmful”.

Right, I get it's not the same for all default gems and for some it makes a lot of sense to depend on them (e.g., when needing behavior from a specific version).
I meant this mainly in the context of the dependencies declared in gems which are part of stdlib, i.e., default or bundled gems, which themselves sometimes explicitly depend on default gems and that seems unnecessary and is problematic in some cases.

I'll adjust the issue title to try to reflect that.

#18 - 02/03/2022 11:52 AM - Eregon (Benoit Daloze)
- Description updated

#19 - 02/03/2022 01:22 PM - vo.x (Vit Ondruch)
Eregon (Benoit Daloze) wrote in #note-14:

vo.x (Vit Ondruch) wrote in #note-11:

> 1) Not specifying dependencies prevents major changes. If there is default gem A of version 1.5.2 and there should be major update in future version of Ruby, lets say to the version 2.0.0. If some gem specified the dependency on A ~> 1.0, there would not be problem, it could keep using the old version. But your proposal is going against this, so I am not sure what would be the suggestion for this case.

Unlikely? How about openssl in Ruby 3.1?

, and even if they do feature checking is always better to support more Ruby versions (e.g., respond_to?).

No doubt about this.

Many scripts/programs/gems out depend on these 4 stdlibs, they do not specify the dependencies on such basic default gems, and the majority of them will not specify such dependencies needlessly in the future either.

We should not mix some specific case, such as the io* which were proposed to be merged, but in the context of $subject, the proposal is wrong and harmful.

#20 - 02/03/2022 01:34 PM - Eregon (Benoit Daloze)
vo.x (Vit Ondruch) wrote in #note-19:

Unlikely? How about openssl in Ruby 3.1?
By "those parts of stdlib" I meant the 4 gems listed above.
io-wait & io-nonblock have < 5 methods each. Digest and strscan APIs are very stable and need to remain that way.
Because people are used to them being stdlibs it's anyway unreasonable to make incompatible changes there.

I updated the subject, I meant to only discuss dependencies of stdlib gems but it was probably not clear before, sorry about that.

Other gems can do whatever they want, we have no control over that.
However, I notice almost none of those non-stdlib gems include dependencies on default gems, unless they need it (e.g., needs some new method/behavior from version X).
So the arguments about "good to always specify default gem deps" seem disconnected from reality.

#21 - 02/03/2022 02:18 PM - deivid (David Rodriguez)

Being a default gem is already enough to fix a security issue without a CRuby release.

This is currently enough to be able to release security fixes independently, but only if dependencies are explicitly recorded in lockfiles these fixes will get to end users of Bundler. Otherwise users are stuck with the insecure version even if they manually install a secure version through RubyGems.
The only way to get out of this as of now is to specify the secure dependency directly in end users Gemfile, which is very unnatural since most of the times they might not be using the dependency directly. This is also how most people ended up work around the issue with the mail gem being uninstallable in Ruby 3.1, and wouldn't be necessary once mail releases a version that properly declares its own dependencies.

#22 - 02/03/2022 06:05 PM - Eregon (Benoit Daloze)

deivid (David Rodriguez) That is a good point.
For io-wait/io-nonblock it just seems very unlikely to have security vulnerabilities in them, but for digest/strscan that is less clear.

#23 - 02/07/2022 10:02 AM - Eregon (Benoit Daloze)

A concrete issue caused by adding digest as a needless dependency for net-*: [https://github.com/oracle/truffleruby/issues/2577](https://github.com/oracle/truffleruby/issues/2577)
It's also a particularly obscure error because TruffleRuby does define these classes but the default gem wins and breaks everything.

#24 - 02/17/2022 11:14 AM - Eregon (Benoit Daloze)

Another instance of this issue: [https://github.com/mikel/mail/pull/1472#discussion_r808927322](https://github.com/mikel/mail/pull/1472#discussion_r808927322)

I think new default gems should initially noop (i.e., just use the stdlib version) on RUBY_ENGINE != "ruby", that would be much less breaking when it goes out (if I'd just use the stdlib version), and later enabled for specific Ruby implementations where it makes sense.

The new default gem can also initially support more than just "ruby" if the CI of that gem passes on other Ruby implementations.

We've seen many instances of gems breaking on JRuby & TruffleRuby over the years due to new default gems, this would significantly help avoiding that.
So:

- Either add JRuby & TruffleRuby to the new default gem CI, if it works all is good.
- If it fails then make the gem delegate to stdlib on that Ruby implementation and ping [headius (Charles Nutter)](https://github.com/headius) (for JRuby) / [Eregon (Benoit Daloze)](https://github.com/eragon) (for TruffleRuby).

#25 - 02/17/2022 05:28 PM - Dan0042 (Daniel DeLorme)

Here's one idea: what if there was a list of all standard libraries in a given ruby version, such as RbConfig::STDLIBS = Set["digest", "strscan", "delegate", etc...]
Then rubygems itself could use this list to ensure that spec.add_dependency(name) is a no-op if RbConfig::STDLIBS.include?(name)

It might sound crazy but I think that would work universally for all gems and implementations.

#26 - 02/17/2022 05:39 PM - Eregon (Benoit Daloze)

Dan0042 (Daniel DeLorme) One issue is some people want the dependency on default gems to be in the Gemfile.lock, which e.g. enables the gem to be checked for security issues easily (e.g. dependabot).
And of course if someone specifies a specific version for a default gem they would like the version to be respected and not use the potentially-older version in stdlib (especially if they need a new method or so).

That also means a default gem being noop is not always ideal from a security POV, but:
1) that can be fixed by supporting the Ruby implementation in that gem directly
2) the alternative Ruby implementation might not have that security issue anyway since it has a different implementation (e.g., date ReDoS don't affect truffleruby due to the better Regexp engine, less buffer overflows on JVM, etc).
3) it is unlikely to have security issues for some small default gems
4) a Ruby implementations can always decide to make an extra release to fix a security issue in stldlb

#27 - 02/17/2022 06:26 PM - Dan0042 (Daniel DeLorme)
Eregon (Benoît Daloze) sorry I was not talking about default gems; by “stdlib” I meant the libraries that have not (yet) been gemified. So I must apologize that my example was completely flawed, as I didn't realize those 3 libs are actually gemified. A correct example would have been:
RbConfig::STDLIBS = Set["io/wait", "objspace", "nkf", etc...]

More concretely, we could say that "net/map" and "strscan" are part of STDLIBS on Jruby but not on Cruby. That way we have the security benefits on implementations that support the gem, and fallback to the stldlb otherwise.

#28 - 02/17/2022 10:49 PM - headius (Charles Nutter)

We've seen many instances of gems breaking on JRuby & TruffleRuby over the years due to new default gems

We (JRuby team) have been working to support the various default gems on JRuby for several years (or more, if you consider json, psych) and we support the gemification of as much as possible of the standard library for security updates and ease of updating. Larger gems like bigdecimal and openssl should probably delegate to JRuby-specific gems (e.g. jruby-openssl), but as often as possible we will incorporate our versions of extensions directly into the gem repositories.

I'm not sure how I feel about having any of these gems be no-ops because it seems misleading.

#29 - 02/21/2022 11:32 AM - hsbt (Hiroshi SHIBATA)

- Assignee set to hsbt (Hiroshi SHIBATA)

I'm not sure what the main problem is for this. Many of concerns, opinions, facts are mixed for me.

I agreed headius's comment that is https://bugs.ruby-lang.org/issues/18567#note-10.

- I agreed io-wait and io-noblock will be integrate core class of IO. So, We should remove them from stldlb dependencies.
- I'm bit of against to remove strscan, digest and others. Because it's better to manage a dependencies without Ruby versions.

I'm working and worked the default gems with JRuby. It works for Ruby ecosystem. I don't understand what a problem for this.

I will do summary of "Non dependencies list for stldlb" like io-wait for Ruby 3.2.