Ruby trunk - Bug #9424

ruby 1.9 & 2.x has insecure SSL/TLS client defaults

01/17/2014 05:39 PM - jmhodges (Jeff Hodges)

Status:	Closed		
Priority:	Normal		
Assignee:	nagachika (Tomoyuki Chikanaga)		
Target version:	2.2.0		
ruby -v:	-	**Backport:**	1.9.3: DONE, 2.0.0: DONE, 2.1: DONE

Description

Ruby 1.9, 2.0, and 2.1 use insecure defaults for SSL/TLS client connections. They have inherited or overridden configs that make the OpenSSL-controlled connections insecure. Note: both OpenSSL's and Ruby's defaults in all tested versions are currently insecure. Confirmation of the issues with Ruby's TLS client can be done with the code in [1].

Ruby is using TLS compression by default. This opens Ruby clients to the CRIME attack[2].

Ruby also uses a variety of insecure cipher suites. These cipher suites either use key sizes much smaller than the currently recommended size, making brute forcing a decryption easy, or do not check the veracity of the server's certificate making them susceptible to man-in-the-middle attacks[3][4].

Ruby also appears to allow SSLv2 connections by default. It does so by first trying to connect with a SSLv2 client hello with a higher SSL/TLS version inside of it which allows SSLv2 servers to work. SSLv2 was broken in the 1990s and is considered unsafe.

These issues expose Ruby users to attacks that have been known for many years, and are trivial to discover. These defaults are often build specific, and are not the same across platforms, but are consistently poor (the code in [1] can evaluate the build). A patch from a core developer on the security@ list is attached. However, the patch does not correct the suspect SSLv2 configuration. It is believed that Ruby 1.8 is also a concern, but, since it was obsoleted, it's not been investigated.

A report similar to this was sent to security@ruby-lang.org four days ago. The Ruby core developers have been unable to patch these problems in a timely manner for it for what I and others believe are concerning reasons. This ticket is being made to allow engineers outside of the small group that are on security@ to protect themselves from these attacks.

[1] https://gist.github.com/cscotta/8302049
[2] https://www.howsmyssl.com/s/about.html#tls-compression
[3] https://www.howsmyssl.com/s/about.html#insecure-cipher-suites
[4] TLS_DHE_DSS_WITH_DES_CBC_SHA - small keys
TLS_DHE_RSA_WITH_DES_CBC_SHA - small keys
TLS_ECDH_anon_WITH_3DES_EDE_CBC_SHA - MITM
TLS_ECDH_anon_WITH_AES_128_CBC_SHA - MITM
TLS_ECDH_anon_WITH_AES_256_CBC_SHA - MITM
TLS_ECDH_anon_WITH_RC4_128_SHA - MITM
TLS_RSA_WITH_DES_CBC_SHA - small keys
TLS_SRP_SHA_WITH_3DES_EDE_CBC_SHA - MITM
TLS_SRP_SHA_WITH_AES_128_CBC_SHA - MITM
TLS_SRP_SHA_WITH_AES_256_CBC_SHA - MITM

Related issues:

Related to Backport21 - Backport #9640: Please backport SSL fixes to 2.1	**Closed**	**03/15/2014**

Associated revisions

Revision 699b209c - 03/06/2014 01:43 AM - emboss

- lib/openssl/ssl.rb: Explicitly whitelist the default SSL/TLS ciphers. Forbid SSLv2 and SSLv3, disable compression by default. Reported by Jeff Hodges. [ruby-core:59829] [Bug #9424]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@45274 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 45274 - 03/06/2014 01:43 AM - emboss

- lib/openssl/ssl.rb: Explicitly whitelist the default SSL/TLS ciphers. Forbid SSLv2 and SSLv3, disable compression by default. Reported by Jeff Hodges. [ruby-core:59829] [Bug #9424]

Revision 45274 - 03/06/2014 01:43 AM - emboss

- lib/openssl/ssl.rb: Explicitly whitelist the default SSL/TLS ciphers. Forbid SSLv2 and SSLv3, disable compression by default. Reported by Jeff Hodges. [ruby-core:59829] [Bug #9424]

Revision 45274 - 03/06/2014 01:43 AM - emboss

- lib/openssl/ssl.rb: Explicitly whitelist the default SSL/TLS ciphers. Forbid SSLv2 and SSLv3, disable compression by default. Reported by Jeff Hodges. [ruby-core:59829] [Bug #9424]

Revision 45274 - 03/06/2014 01:43 AM - emboss

- lib/openssl/ssl.rb: Explicitly whitelist the default SSL/TLS ciphers. Forbid SSLv2 and SSLv3, disable compression by default. Reported by Jeff Hodges. [ruby-core:59829] [Bug #9424]

Revision 45274 - 03/06/2014 01:43 AM - emboss

- lib/openssl/ssl.rb: Explicitly whitelist the default SSL/TLS ciphers. Forbid SSLv2 and SSLv3, disable compression by default. Reported by Jeff Hodges. [ruby-core:59829] [Bug #9424]

Revision 45274 - 03/06/2014 01:43 AM - emboss

- lib/openssl/ssl.rb: Explicitly whitelist the default SSL/TLS ciphers. Forbid SSLv2 and SSLv3, disable compression by default. Reported by Jeff Hodges. [ruby-core:59829] [Bug #9424]

Revision 8d67a06b - 03/06/2014 04:42 PM - naruse (Yui NARUSE)

fix r45274; it change default but doesn't change tests [Bug #9424]

RUN TESTS BEFORE COMMIT!!!

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@45278 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 45278 - 03/06/2014 04:42 PM - naruse (Yui NARUSE)

fix r45274; it change default but doesn't change tests [Bug #9424]

RUN TESTS BEFORE COMMIT!!!

Revision 45278 - 03/06/2014 04:42 PM - naruse (Yui NARUSE)

fix r45274; it change default but doesn't change tests [Bug #9424]

RUN TESTS BEFORE COMMIT!!!

Revision 45278 - 03/06/2014 04:42 PM - naruse (Yui NARUSE)

fix r45274; it change default but doesn't change tests [Bug #9424]

RUN TESTS BEFORE COMMIT!!!

Revision 45278 - 03/06/2014 04:42 PM - naruse (Yui NARUSE)

fix r45274; it change default but doesn't change tests [Bug #9424]

RUN TESTS BEFORE COMMIT!!!

Revision 45278 - 03/06/2014 04:42 PM - naruse (Yui NARUSE)

fix r45274; it change default but doesn't change tests [Bug #9424]

RUN TESTS BEFORE COMMIT!!!

Revision 45278 - 03/06/2014 04:42 PM - naruse (Yui NARUSE)

fix r45274; it change default but doesn't change tests [Bug #9424]

RUN TESTS BEFORE COMMIT!!!

Revision 7b0635d1 - 10/22/2014 01:55 PM - nagachika (Tomoyuki Chikanaga)

- ext/openssl/lib/openssl/ssl.rb (DEFAULT_PARAMS): override options even if OpenSSL::SSL::OP_NO_SSLv3 is not defined. this is pointed out by Stephen Touset. [ruby-core:65711] [Bug #9424]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@48097 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 48097 - 10/22/2014 01:55 PM - nagachika (Tomoyuki Chikanaga)

- ext/openssl/lib/openssl/ssl.rb (DEFAULT_PARAMS): override options even if OpenSSL::SSL::OP_NO_SSLv3 is not defined. this is pointed out by Stephen Touset. [ruby-core:65711] [Bug #9424]

Revision 48097 - 10/22/2014 01:55 PM - nagachika (Tomoyuki Chikanaga)

- ext/openssl/lib/openssl/ssl.rb (DEFAULT_PARAMS): override options even if OpenSSL::SSL::OP_NO_SSLv3 is not defined. this is pointed out by Stephen Touset. [ruby-core:65711] [Bug #9424]

Revision 48097 - 10/22/2014 01:55 PM - nagachika (Tomoyuki Chikanaga)

- ext/openssl/lib/openssl/ssl.rb (DEFAULT_PARAMS): override options even if OpenSSL::SSL::OP_NO_SSLv3 is not defined. this is pointed out by Stephen Touset. [ruby-core:65711] [Bug #9424]

Revision 48097 - 10/22/2014 01:55 PM - nagachika (Tomoyuki Chikanaga)

- ext/openssl/lib/openssl/ssl.rb (DEFAULT_PARAMS): override options even if OpenSSL::SSL::OP_NO_SSLv3 is not defined. this is pointed out by Stephen Touset. [ruby-core:65711] [Bug #9424]

Revision 48097 - 10/22/2014 01:55 PM - nagachika (Tomoyuki Chikanaga)

- ext/openssl/lib/openssl/ssl.rb (DEFAULT_PARAMS): override options even if OpenSSL::SSL::OP_NO_SSLv3 is not defined. this is pointed out by Stephen Touset. [ruby-core:65711] [Bug #9424]

Revision 48097 - 10/22/2014 01:55 PM - nagachika (Tomoyuki Chikanaga)

- ext/openssl/lib/openssl/ssl.rb (DEFAULT_PARAMS): override options even if OpenSSL::SSL::OP_NO_SSLv3 is not defined. this is pointed out by Stephen Touset. [ruby-core:65711] [Bug #9424]

Revision c8137d67 - 10/22/2014 02:14 PM - nagachika (Tomoyuki Chikanaga)

merge revision(s) r45274,r45278,r45280,r48097: [Backport #9424] [Backport #9640]

```
    * lib/openssl/ssl.rb: Explicitly whitelist the default
      SSL/TLS ciphers. Forbid SSLv2 and SSLv3, disable
      compression by default.
      Reported by Jeff Hodges.
      [ruby-core:59829] [Bug #9424]

    * test/openssl/test_ssl.rb: Reuse TLS default options from
      OpenSSL::SSL::SSLContext::DEFAULT_PARAMS.

    * ext/openssl/lib/openssl/ssl.rb (DEFAULT_PARAMS): override
      options even if OpenSSL::SSL::OP_NO_SSLv3 is not defined.
      this is pointed out by Stephen Touset. [ruby-core:65711] [Bug #9424]
```

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/branches/ruby_2_1@48098 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 48098 - 10/22/2014 02:14 PM - nagachika (Tomoyuki Chikanaga)

merge revision(s) r45274,r45278,r45280,r48097: [Backport #9424] [Backport #9640]

```
* lib/openssl/ssl.rb: Explicitly whitelist the default
  SSL/TLS ciphers. Forbid SSLv2 and SSLv3, disable
  compression by default.
  Reported by Jeff Hodges.
  [ruby-core:59829] [Bug #9424]

* test/openssl/test_ssl.rb: Reuse TLS default options from
  OpenSSL::SSL::SSLContext::DEFAULT_PARAMS.

* ext/openssl/lib/openssl/ssl.rb (DEFAULT_PARAMS): override
```

```
  options even if OpenSSL::SSL::OP_NO_SSLv3 is not defined.
  this is pointed out by Stephen Touset. [ruby-core:65711] [Bug #9424]
```

Revision 4a5d839c - 10/23/2014 09:59 AM - usa (Usaku NAKAMURA)

merge revision(s) 45274,45278,45280,48097: [Backport #9424]

```
    * lib/openssl/ssl.rb: Explicitly whitelist the default
      SSL/TLS ciphers. Forbid SSLv2 and SSLv3, disable
      compression by default.
      Reported by Jeff Hodges.
      [ruby-core:59829] [Bug #9424]

    * test/openssl/test_ssl.rb: Reuse TLS default options from
      OpenSSL::SSL::SSLContext::DEFAULT_PARAMS.

    * ext/openssl/lib/openssl/ssl.rb (DEFAULT_PARAMS): override
      options even if OpenSSL::SSL::OP_NO_SSLv3 is not defined.
      this is pointed out by Stephen Touset. [ruby-core:65711] [Bug #9424]
```

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/branches/ruby_2_0_0@48110 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 48110 - 10/23/2014 09:59 AM - usa (Usaku NAKAMURA)

merge revision(s) 45274,45278,45280,48097: [Backport #9424]

```
* lib/openssl/ssl.rb: Explicitly whitelist the default
  SSL/TLS ciphers. Forbid SSLv2 and SSLv3, disable
  compression by default.
  Reported by Jeff Hodges.
  [ruby-core:59829] [Bug #9424]

* test/openssl/test_ssl.rb: Reuse TLS default options from
  OpenSSL::SSL::SSLContext::DEFAULT_PARAMS.

* ext/openssl/lib/openssl/ssl.rb (DEFAULT_PARAMS): override
  options even if OpenSSL::SSL::OP_NO_SSLv3 is not defined.
  this is pointed out by Stephen Touset. [ruby-core:65711] [Bug #9424]
```

Revision 26c0acf5 - 10/24/2014 03:06 AM - usa (Usaku NAKAMURA)

merge revision(s) 45274,45278,45280,48097: [Backport #9424]

```
    * ext/openssl/lib/openssl/ssl-internal.rb (DEFAULT_PARAMS): override
      options even if OpenSSL::SSL::OP_NO_SSLv3 is not defined.
      this is pointed out by Stephen Touset. [ruby-core:65711] [Bug #9424]

    * test/openssl/test_ssl.rb: Reuse TLS default options from
      OpenSSL::SSL::SSLContext::DEFAULT_PARAMS.

    * lib/openssl/ssl-internal.rb: Explicitly whitelist the default
      SSL/TLS ciphers. Forbid SSLv2 and SSLv3, disable
      compression by default.
      Reported by Jeff Hodges.
      [ruby-core:59829] [Bug #9424]
```

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/branches/ruby_1_9_3@48121 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 48121 - 10/24/2014 03:06 AM - usa (Usaku NAKAMURA)

merge revision(s) 45274,45278,45280,48097: [Backport #9424]

```
* ext/openssl/lib/openssl/ssl-internal.rb (DEFAULT_PARAMS): override
  options even if OpenSSL::SSL::OP_NO_SSLv3 is not defined.
  this is pointed out by Stephen Touset. [ruby-core:65711] [Bug #9424]

* test/openssl/test_ssl.rb: Reuse TLS default options from
  OpenSSL::SSL::SSLContext::DEFAULT_PARAMS.

* lib/openssl/ssl-internal.rb: Explicitly whitelist the default
  SSL/TLS ciphers. Forbid SSLv2 and SSLv3, disable
  compression by default.
  Reported by Jeff Hodges.
  [ruby-core:59829] [Bug #9424]
```

History

#1 - 01/17/2014 06:14 PM - jmhodges (Jeff Hodges)

I've published the email thread with security@ruby-lang.orghttps://gist.github.com/jmhodges/d6480f5f81f25b0dfa15

#2 - 01/17/2014 11:31 PM - nobu (Nobuyoshi Nakada)

Couldn't you split for each mails?

#3 - 01/20/2014 11:33 PM - jmhodges (Jeff Hodges)

That was how it was laid out originally, but GitHub wouldn't display the last 6 or so emails and said "Not showing 6 files" or something.

#4 - 01/21/2014 01:52 AM - nobu (Nobuyoshi Nakada)

- Status changed from Open to Third Party's Issue

This is primarily an issue of OpenSSL.
And the next candidates are OS/Platform/Packager providers.

Fixing all other tools is the last resort.

#5 - 01/21/2014 07:40 PM - abedra (Aaron Bedra)

Nobuyoshi Nakada wrote:

> This is primarily an issue of OpenSSL.
> And the next candidates are OS/Platform/Packager providers.
>
> Fixing all other tools is the last resort.

Waiting for OpenSSL to fix these things means that they will never get fixed. Even if somehow these defaults/options do disappear it will take years for people to benefit from the update. Most people are still stuck on some flavor of OpenSSL 0.98 because of their distro/packager of choice. This is low hanging fruit for Ruby and has a significant impact to the users of the language.

There are people here that will happily do the work (and it looks like Jeff already has). Please reconsider allowing these changes. +1's welcome here.

#6 - 01/21/2014 07:53 PM - markkropf (Mark Kropf)

+1

#7 - 01/21/2014 09:50 PM - gabrielg (Gabriel Gironda)

There are many other places where Ruby takes a stance intended to avoid users shooting themselves in the foot. For example:

```
irb(main):008:0> rd, wr = IO.pipe
=> [#<IO:fd 7>, #<IO:fd 8>]
irb(main):009:0> rd.close_on_exec = false
=> false
irb(main):010:0> wr.close_on_exec = false
=> false
irb(main):011:0> rd.close_on_exec?
=> false
irb(main):012:0> wr.close_on_exec?
=> false
irb(main):013:0> system("ls /dev/fd/7")
ls: /dev/fd/7: Bad file descriptor
=> false
irb(main):014:0> system("ls /dev/fd/8")
ls: /dev/fd/8: Bad file descriptor
=> false
irb(main):015:0> system("ls /dev/fd/8", close_others: false)
/dev/fd/8
=> true
```

I assume the intent here is to prevent accidental file descriptor leakage.

I'm not sure where the line is drawn-why is altering default file descriptor behaviour okay, but changing known-bad OpenSSL defaults is considered out of bounds?

Please reconsider the decision to not apply this patch.

#8 - 01/21/2014 10:32 PM - normalperson (Eric Wong)

I'm mildy in favor of overriding OpenSSL defaults for now.

However I suggest removing openssl from the standard library to
encourage saner SSL implementations.
IMNSHO, OpenSSL is a disaster.

#9 - 01/22/2014 01:27 AM - nobu (Nobuyoshi Nakada)

- Category set to ext/openssl

- Status changed from Third Party's Issue to Assigned

- Assignee set to MartinBosslet (Martin Bosslet)

- Target version set to 2.2.0

I agree the removal.

BTW, what's the reason you believe that people who don't upgrade even OpenSSL will upgrade Ruby?

#10 - 01/22/2014 02:33 AM - sgonyea (Scott Gonyea)

Nobuyoshi Nakada wrote:

> I agree the removal.
>
> BTW, what's the reason you believe that people who don't upgrade even OpenSSL will upgrade Ruby?

Those people are damned regardless. For anyone who does upgrade their version of Ruby, at the very least, this will protect them from OpenSSL's known / past insecure defaults. That is the pragmatic (correct) approach. OpenSSL 0.9.8 is the default on OS X and many still-supported Linux distros.

I'd strongly be in favor of a backport to 1.9.3 as well. Thank you.

#11 - 01/22/2014 04:33 AM - nobu (Nobuyoshi Nakada)

Scott Gonyea wrote:

> Nobuyoshi Nakada wrote:
>
> > I agree the removal.
> >
> > BTW, what's the reason you believe that people who don't upgrade even OpenSSL will upgrade Ruby?
>
> Those people are damned regardless. For anyone who does upgrade their version of Ruby, at the very least, this will protect them from OpenSSL's known / past insecure defaults. That is the pragmatic (correct) approach.

Why won't they upgrade OpenSSL, but only Ruby?

> OpenSSL 0.9.8 is the default on OS X and many still-supported Linux distros.

It's the responsibility of distributors.
Then we should reject older versions?

#12 - 01/22/2014 04:51 AM - myronmarston (Myron Marston)

> Why won't they upgrade OpenSSL, but only Ruby?

I can't speak for everyone, but speaking for myself: before Jeff raised this issue I had no idea that OpenSSL was something I needed to pay attention to and consider upgrading. I vaguely knew that ruby used it, but I didn't even know it was possible to upgrade apart from ruby. I'm a bit embarrassed to say this, but even now, I have no idea how to upgrade OpenSSL even if I wanted to. Ruby, on the other hand, I'm quite used to upgrading.

I'm sure there are plenty of ruby users that know about this stuff and are aware of the need to upgrade OpenSSL, but I believe the majority of ruby users, like me, have no idea about this stuff. I'm by no means a ruby newbie; I've been using the language since 2007 and maintain a major ruby open source project (RSpec).

I think it's irresponsible for ruby to do nothing here and rely on users to mess with their OpenSSL installation.

#13 - 01/22/2014 06:20 AM - ralish (Samuel Leslie)

Nobuyoshi Nakada wrote:

> Why won't they upgrade OpenSSL, but only Ruby?

OpenSSL on effectively all mainstream/general-purpose distributions tends to be a core system library which many applications depend on. This makes upgrading it outside of the standard distribution updating process (via yum, apt, pacman, etc...) decidedly non-trivial and generally ill-advised unless you **really** know what you're doing. The more sane alternative for where the base OpenSSL libraries don't suffice is to compile your application against a custom OpenSSL build instead of using the system provided libraries. This of course means that you have to compile a custom OpenSSL in the first place instead of using distribution provided pre-built packages and is likely to entail some level of extra work on the users part. It's often relatively difficult/complex to do and we shouldn't expect Ruby users to have to go through this process just to get a secure installation.

Nobuyoshi Nakada wrote:

> It's the responsibility of distributors.

The distributions responsibility only extends to ensuring that the OpenSSL libraries included with the system function as expected and are secure to the extent there are no vulnerabilities in the installed version. Note that what we're discussing here is not a vulnerability due to a flaw in the OpenSSL codebase itself but rather vulnerabilities being potentially exposed due to poor configuration. Many libraries can be misused in a way to result in an insecure configuration and crypto libraries are a particularly prominent case. OpenSSL admittedly makes it easier to do than other libraries due to some particularly strange or outright poor defaults, but there's a long tail of legacy reasons for many of these choices, and ultimately, Ruby as a user of the OpenSSL library should take responsibility for using the library properly. This extends to ensuring we use sane defaults so Ruby users are not exposed to unnecessary risk. Requesting OpenSSL just change these for our benefit is both not going to happen and trying to avoid responsibility.

Nobuyoshi Nakada wrote:

> Then we should reject older versions?

No, as there would be a huge installed base of users running older OpenSSL versions which aren't technically at fault (in that they aren't inherently broken when used with sane settings). Upgrading these installations is impractical or overly difficult for the reasons noted above alongside the fact the libraries themselves are fine when configured appropriately.

Myron Marston wrote:

> I can't speak for everyone, but speaking for myself: before Jeff raised this issue I had no idea that OpenSSL was something I needed to pay attention to and consider upgrading.

Which just adds to the importance that the Ruby project take responsibility for overriding any defaults which OpenSSL uses that are either outright insecure or just simply a poor default choice for Ruby. It's completely understandable that Myron wouldn't be aware of the links between Ruby and OpenSSL and I suspect this is true for the vast majority of Ruby users. Expecting that Ruby users who utilise OpenSSL in a direct or indirect way have in-depth knowledge of OpenSSL default settings and their respective security ramifications is absurd.

It's true that security is hard, that's why Ruby as a respected and prominent programming language needs to ensure that appropriate steps are taken to protect users of the language by ensuring they are not exposed to unnecessary risk via poor security defaults.

#14 - 01/22/2014 06:45 AM - shyouhei (Shyouhei Urabe)

Samuel Leslie wrote:

> Nobuyoshi Nakada wrote:
>
> > Why won't they upgrade OpenSSL, but only Ruby?
>
> OpenSSL on effectively all mainstream/general-purpose distributions tends to be a core system library which many applications depend on. This makes upgrading it outside of the standard distribution updating process (via yum, apt, pacman, etc...) decidedly non-trivial and generally ill-advised unless you **really** know what you're doing. The more sane alternative for where the base OpenSSL libraries don't suffice is to compile your application against a custom OpenSSL build instead of using the system provided libraries. This of course means that you have to compile a custom OpenSSL in the first place instead of using distribution provided pre-built packages and is likely to entail some level of extra work on the users part. It's often relatively difficult/complex to do and we shouldn't expect Ruby users to have to go through this process just to get a secure installation.

So do ruby. Don't know what OS you're using but the situation doesn't change between OpenSSL and Ruby in vast majority of Linux distributions, maybe also on BSDs.

> Nobuyoshi Nakada wrote:
>
> > It's the responsibility of distributors.

> The distributions responsibility only extends to ensuring that the OpenSSL libraries included with the system function as expected and are secure to the extent there are no vulnerabilities in the installed version. Note that what we're discussing here is not a vulnerability due to a flaw in the OpenSSL codebase itself but rather vulnerabilities being potentially exposed due to poor configuration. Many libraries can be misused in a way to result in an insecure configuration and crypto libraries are a particularly prominent case. OpenSSL admittedly makes it easier to do than other libraries due to some particularly strange or outright poor defaults, but there's a long tail of legacy reasons for many of these choices, and ultimately, Ruby as a user of the OpenSSL library should take responsibility for using the library properly. This extends to ensuring we use sane defaults so Ruby users are not exposed to unnecessary risk. Requesting OpenSSL just change these for our benefit is both not going to happen and trying to avoid responsibility.

We are amateur about security. It might be possible to change
something, then we have no idea what happens with that modification
and even worse, we cannot maintain that bit when security research
develops and turned out our change was in fact ill.

Talking about responsibility, I have to say that providing what we
don't understand is worse than just doing nothing.

> Nobuyoshi Nakada wrote:
>
> > Then we should reject older versions?
>
> No, as there would be a huge installed base of users running older OpenSSL versions which aren't technically at fault (in that they aren't inherently broken when used with sane settings). Upgrading these installations is impractical or overly difficult for the reasons noted above alongside the fact the libraries themselves are fine when configured appropriately.

OK, fine about this point.

> Myron Marston wrote:
>
> > I can't speak for everyone, but speaking for myself: before Jeff raised this issue I had no idea that OpenSSL was something I needed to pay attention to and consider upgrading.
>
> Which just adds to the importance that the Ruby project take responsibility for overriding any defaults which OpenSSL uses that are either outright insecure or just simply a poor default choice for Ruby. It's completely understandable that Myron wouldn't be aware of the links between Ruby and OpenSSL and I suspect this is true for the vast majority of Ruby users. Expecting that Ruby users who utilise OpenSSL in a direct or indirect way have in-depth knowledge of OpenSSL default settings and their respective security ramifications is absurd.
>
> It's true that security is hard, that's why Ruby as a respected and prominent programming language needs to ensure that appropriate steps are taken to protect users of the language by ensuring they are not exposed to unnecessary risk via poor security defaults.

Security is hard because NO ONE HELPS US BUT JUST COMPLAIN. Fuck off.

#15 - 01/22/2014 07:15 AM - charliesome (Charlie Somerville)

Shyouhei Urabe wrote:

> Security is hard because NO ONE HELPS US BUT JUST COMPLAIN. Fuck off.

This is a completely unreasonable reply.

Several members of the community have already offered to assist in maintaining more secure defaults in the OpenSSL extension.

Perhaps we should accept these offers and work with people who are willing to help improve Ruby, rather than telling people to "fuck off" for arguing why ruby-core's decision on this issue may not be in the best interest of the majority of Ruby's users.

#16 - 01/22/2014 08:59 AM - spatulasnout (B Kelly)

shyouhei@ruby-lang.org wrote:

> We are amateur about security. It might be possible to change
> something, then we have no idea what happens with that modification
> and even worse, we cannot maintain that bit when security research
> develops and turned out our change was in fact ill.

I am also an amateur. But I read the logic of your statement above as
being in favor of discarding security research that /already exists/
about the weak ciphers and protocol versions.

But: if we are to disregard current research, should not the reason given
be something other than concern over possible future research?

With regard to maintenance, could it be useful to incorporate a check
like https://gist.github.com/cscotta/8302049 in the form of an automated
test which can be run by maintainers prior to the release of a new version
of ruby? The idea being that such a test may assist in proactively warning
maintainers if/when further improvements to ruby's OpenSSL defaults are
warranted.

Regards,

Bill

#17 - 01/22/2014 09:05 AM - shyouhei (Shyouhei Urabe)

Charlie Somerville wrote:

> Shyouhei Urabe wrote:
>
>> Security is hard because NO ONE HELPS US BUT JUST COMPLAIN. Fuck off.
>
> This is a completely unreasonable reply.

Sorry, I said too much.

> Several members of the community have already offered to assist in maintaining more secure defaults in the OpenSSL extension.
>
> Perhaps we should accept these offers and work with people who are willing to help improve Ruby, rather than telling people to "fuck off" for arguing why ruby-core's decision on this issue may not be in the best interest of the majority of Ruby's users.

I have been reading this thread as a request for us to be perfect
to provide 128% secure product or just die. If there are people
interested in being a super hero, please just kill us take the
position. Oust us from being perfect. I welcome them very much.

#18 - 01/22/2014 10:03 AM - mame (Yusuke Endoh)

Charlie Somerville wrote:

> Several members of the community have already offered to assist in maintaining more secure defaults in the OpenSSL extension.

Please give us not only words but also a concrete plan.

The following is just my personal opinion, not the consensus of ruby-core. This issue must be eventually determined by Matz and Martin Bosslet who is the maintainer of Ruby OpenSSL.

The matter is how we determine what is "secure".
I believe that the (current) Ruby committers are not qualified to determine that. We are more or less experts for the Ruby implementation, but not for crypto. In an extreme case, default settings chosen by ignoramus are not only invalid but also harmful. Providing such a thing is really unfaithful.
It is possibly dangerous to trust a bug reporter blindly. At least we must validate their reports. But we might get tricked.

I thought that the best person to determine that was the OpenSSL project itself. However, according to some people, OpenSSL seems not to be interested in secure defaults. That's unfortunate, but I respect their decision.

Then, I think of three possibilities:

- To follow the same policy as OpenSSL, i.e., delegating the task to users.
- To replace Ruby OpenSSL with another implementations, such as krypt.
- To find another authority to follow.

I think the third is best, but we must determine the following things.

- The authority that suggests a good configuration. NIST? NESSIE? CRYPTREC? Or what? I'm not sure.
- A trusted, motivated, and expert committer(s) who can interpret and implement the suggested configuration into Ruby OpenSSL.
- Preparation to continue following the changes of the configuration.

--
Yusuke Endoh mame@tsg.ne.jp

#19 - 01/22/2014 10:59 AM - dbussink (Dirkjan Bussink)

Yusuke Endoh wrote:

> Please give us not only words but also a concrete plan.

As said in the original mailing list, I'm more than willing to maintain this list for Ruby.

> The matter is how we determine what is "secure".
>
> - The authority that suggests a good configuration. NIST? NESSIE? CRYPTREC? Or what? I'm not sure.

Actually OpenSSL already provides information on this. They already classify ciphers in categories like LOW, MEDIUM and HIGH. Currently there is no reason to use LOW anymore, but that is still enabled in the OpenSSL defaults. A fix would mean basically just use these classifications and make Ruby only use MEDIUM and HIGH out of the box.

There are plenty of tools these days available that give trustworthy recommendations such as https://www.ssllabs.com/ which has a best practices guide at https://www.ssllabs.com/projects/best-practices/index.html and also Jeff's new project https://www.howsmyssl.com/. These already provide more than enough information to have enough information to make the best decision possible right now on this issue.

These good defaults might change, because new attacks are found and papers are published on these topics. This information finds there way quick as well to these websites and other best practices for SSL deployments. I feel that for this specific decision about default ciphers there is plenty of information publicly available through the channels that a good choice can be made. I'm not at all stating that such a clear solution would apply for every problem ever in OpenSSL, just that this specific problem is a known problem and has a known solution.

> - A trusted, motivated, and expert committer(s) who can interpret and implement the suggested configuration into Ruby OpenSSL.
> - Preparation to continue following the changes of the configuration.

On both these accounts I again volunteer myself. I already work on the GitHub.com SSL deployment and keep an eye on these topics for this reason and also because I care about security. In cases where there is no consensus on what is a good approach to tackle problems, I will reach out to other people with more knowledge. This also means keeping an eye on what other languages like Python and PHP are doing and how they come to their decisions. In this case these are the decisions made:

http://bugs.python.org/issue13636

Which led to weak ciphers being disabled:
http://hg.python.org/cpython/rev/f9122975fd80

Also PHP made decisions like disabling compression:

https://github.com/php/php-src/commit/4a01ddfb5569da1b87dd4cac95c3f709fb607396

Again, I agree that making security decisions is hard. I also think however that in this context each situation is different. This is why I believe general view of "we don't know" can't be generally applied. In this specific case the consensus on what secure defaults are is very clear so therefore I don't see any problems with applying the solution in this specific case.

So given my statements above, this is a description of how I would have handled this issue in this role:

- After the initial report of Jeff Hodges, review what recommendations from others are. What do deployment guides say for example about these ciphers. What is documented on weak ciphers? Discern the consensus here.
- This would have led to the conclusion that there is a broad consensus on this topic and there is currently no good reason to support weak ciphers by default.
- Review what other languages / environments have done in this area. Seeing languages like Python etc. changed this only strengthens the decision making this change.
- In this case apply the proposed patch and backport it to maintained Ruby versions, cooperating with the appropriate people here.

I'm still willing to do this work now and help people out here in applying the suggested patch. I hope you are able to accept my offer to help out so that we can improve the security of Ruby and keep it up to date in the future.

--
Regards,

Dirkjan Bussink

#20 - 01/22/2014 11:42 AM - samkottler (Sam Kottler)

Dirkjan Bussink wrote:

> Yusuke Endoh wrote:
>
> > Please give us not only words but also a concrete plan.
>
> As said in the original mailing list, I'm more than willing to maintain this list for Ruby.

The matter is how we determine what is "secure".

- The authority that suggests a good configuration. NIST? NESSIE? CRYPTREC? Or what? I'm not sure.

Actually OpenSSL already provides information on this. They already classify ciphers in categories like LOW, MEDIUM and HIGH. Currently there is no reason to use LOW anymore, but that is still enabled in the OpenSSL defaults. A fix would mean basically just use these classifications and make Ruby only use MEDIUM and HIGH out of the box.

There are plenty of tools these days available that give trustworthy recommendations such as https://www.ssllabs.com/ which has a best practices guide at https://www.ssllabs.com/projects/best-practices/index.html and also Jeff's new project https://www.howsmyssl.com/. These already provide more than enough information to have enough information to make the best decision possible right now on this issue.

These good defaults might change, because new attacks are found and papers are published on these topics. This information finds there way quick as well to these websites and other best practices for SSL deployments. I feel that for this specific decision about default ciphers there is plenty of information publicly available through the channels that a good choice can be made. I'm not at all stating that such a clear solution would apply for every problem ever in OpenSSL, just that this specific problem is a known problem and has a known solution.

- A trusted, motivated, and expert committer(s) who can interpret and implement the suggested configuration into Ruby OpenSSL.
- Preparation to continue following the changes of the configuration.

On both these accounts I again volunteer myself. I already work on the GitHub.com SSL deployment and keep an eye on these topics for this reason and also because I care about security. In cases where there is no consensus on what is a good approach to tackle problems, I will reach out to other people with more knowledge. This also means keeping an eye on what other languages like Python and PHP are doing and how they come to their decisions. In this case these are the decisions made:

http://bugs.python.org/issue13636

Which led to weak ciphers being disabled:
http://hg.python.org/cpython/rev/f9122975fd80

Also PHP made decisions like disabling compression:

https://github.com/php/php-src/commit/4a01ddfb5569da1b87dd4cac95c3f709fb607396

Again, I agree that making security decisions is hard. I also think however that in this context each situation is different. This is why I believe general view of "we don't know" can't be generally applied. In this specific case the consensus on what secure defaults are is very clear so therefore I don't see any problems with applying the solution in this specific case.

So given my statements above, this is a description of how I would have handled this issue in this role:

- After the initial report of Jeff Hodges, review what recommendations from others are. What do deployment guides say for example about these ciphers. What is documented on weak ciphers? Discern the consensus here.
- This would have led to the conclusion that there is a broad consensus on this topic and there is currently no good reason to support weak ciphers by default.
- Review what other languages / environments have done in this area. Seeing languages like Python etc. changed this only strengthens the decision making this change.
- In this case apply the proposed patch and backport it to maintained Ruby versions, cooperating with the appropriate people here.

I'm still willing to do this work now and help people out here in applying the suggested patch. I hope you are able to accept my offer to help out so that we can improve the security of Ruby and keep it up to date in the future.

I'm also completely willing to spend my time helping make ruby more secure. I really hope ruby-core will take Dirkjan, Jeff, and myself and any others who step up on our offers.

--
Regards,

Dirkjan Bussink

#21 - 01/22/2014 12:12 PM - MartinBosslet (Martin Bosslet)

- File change_ssl_defaults.diff added

First some words why I (and others here) believe that it's not a good idea to deviate from OpenSSL defaults:

Security is a delicate issue and typically consumers relying on a library like OpenSSL often do so because they don't want to be burdened with low-level decisions. Reasons can be manyfold, lack of domain knowledge, not wanting to take on that responsibility etc. Therefore, at least in my very own opinion, it should be the responsibility of the library itself to keep sane defaults and also to leave as few options as possible remaining for consumers to decide. There should be one central place where these defaults are kept, so anyone knows where to look when trying to analyze or update them. DRY principle and all that. This is why in an ideal world, everybody would rely on OpenSSL's defaults. Rolling your own defaults is dangerous: Even skilled developers like the Debian developers can get it wrong sometimes, with disastrous consequences. It hurts even more that in such cases everyone will start pointing fingers, asking: "Why didn't you stick to the library defaults???" Deviating from library defaults also comes with the burden that it forces you to keep a close eye on any changes regarding upstream defaults. Even the slightest change might completely break the security of any hand-rolled defaults, e.g. if a new parameter adds a new "dimension" or if the meaning of some existing parameter changed etc. Now you might say that this should be my/our duty anyway, but then I'd like to remind you that most of us work on Ruby in our leisure time and we might be on vacation, too busy with work - hell, even if you're paid to watch those changes it can still happen that you simply miss something.

So obviously there are many reasons against maintaining your own defaults, from a theoretical as well as a practical point of view. But I do also agree that OpenSSL might sometimes react too slowly when it comes to updating their defaults. They might even have good reasons to leave some dubious defaults in there because they have to support a myriad of different platforms and environments. So I guess it's time for a compromise. Here's one I could live with:

I would prefer a whitelisting approach instead of blacklisting as in the patch that was proposed. Blacklisting is never airtight, as it doesn't protect us from future shitty algorithms creeping in. I have prepared the attached patch which basically limits the TLS ciphers to AES in all its combinations that are currently considered secure. It should pass Jeff's test suite and only allows TLS in version 1.0 upwards - I can't see a reason for keeping SSL v3.

jEFF (Jean-François Berroyer): Can you confirm this patch passes all your tests (in particular, it no longer allows the v2 handshake upgrade?)

But again, I firmly believe doing this (our own defaults) is a bad idea. That's why I would like anyone who cares about Ruby's security to go the OpenSSL mailing list and ask them to update *their* defaults. That's something that everybody would benefit from, not just Ruby. I also believe that once (if?) OpenSSL defaults have been sanitized we should immediately discard our own custom settings in the following point release. What about Ruby users with old OpenSSL versions? I think it can not and should not be our goal to try to patch security holes in old versions of OpenSSL in Ruby's OpenSSL extension just so that Ruby users are relieved from having to upgrade. The closer we can stick to OpenSSL in Ruby OpenSSL, the better for everyone's security. Keeping your OpenSSL up-to-date is and will be mandatory in order to guarantee a safe-to-use Ruby OpenSSL extension.

#22 - 01/22/2014 12:56 PM - mame (Yusuke Endoh)

Dirkjan Bussink, and Sam Kottler,
Thank you for commenting, but I'd like to drop my suggestion since emboss presented his direction. Could you please contribute by commenting/reviewing his patch?

Emboss, thank you as always with all your great work!

--
Yusuke Endoh mame@tsg.ne.jp

#23 - 01/22/2014 04:25 PM - jmhodges (Jeff Hodges)

On my local machine, that patch does give a Probably Okay response, having removed the TLS compression and the weak cipher suites. Thank you very much for your work and patience.

#24 - 01/22/2014 07:40 PM - abedra (Aaron Bedra)

Jeff Hodges wrote:

> On my local machine, that patch does give a Probably Okay response, having removed the TLS compression and the weak cipher suites. Thank you very much for your work and patience.

Thanks for taking an additional look here and acting on this! Based on some of the earlier responses, it seems like the other take away here is to get some additional attention and support from security folks and several (including myself) have volunteered. Perhaps we can be invited to the ruby core security list or there could be some other input channel on issues like this.

#25 - 01/23/2014 10:19 AM - spatulasnout (B Kelly)

Martin.Bosslet@gmail.com wrote:

> Rolling your own defaults is dangerous: Even skilled developers like the
> Debian developers can get it wrong sometimes, with disastrous consequences.

The Debian blunder has been referenced twice in this discussion, but I think
the comparison is not apt.

The Debian maintainer removed lines of code from the OpenSSL PRNG
implementation. [1]

This is hardly in the same category as tightening the defaults to exclude
specific ciphers or protocol features already known to be weak or exploitable.

> It hurts even more that in such cases everyone will start pointing fingers,
> asking: "Why didn't you stick to the library defaults???"

> As opposed to asking: "Why didn't you remove known weak ciphers and exploitable
> protocol features from the defaults when you were warned about them???"
>
>> I would prefer a whitelisting approach instead of blacklisting as in the
>> patch that was proposed. Blacklisting is never airtight, as it doesn't protect
>> us from future shitty algorithms creeping in.

I wonder. In the blacklisting case, we're not required to make guesses about
the future. We're merely switching off already-known weak or exploitable
features.

Whitelisting goes a step further, gambling that what we know today about the
subset of defaults considered superior will continue to hold true down the road.

It's not clear to me that's better than the more conservative step of simply
blacklisting specific defaults already known to be problematic.

Regards,

Bill

[1] The details are perhaps interesting: http://research.swtch.com/openssl

#26 - 01/23/2014 04:25 PM - MartinBosslet (Martin Bosslet)

Yusuke Endoh wrote:

> Emboss, thank you as always with all your great work!

Np, thank you!

B Kelly wrote:

> Martin.Bosslet@gmail.com wrote:
>
>> Rolling your own defaults is dangerous: Even skilled developers like the
>> Debian developers can get it wrong sometimes, with disastrous consequences.
>
> The Debian blunder has been referenced twice in this discussion, but I think
> the comparison is not apt.
>
> The Debian maintainer removed lines of code from the OpenSSL PRNG
> implementation. [1]
>
> This is hardly in the same category as tightening the defaults to exclude
> specific ciphers or protocol features already known to be weak or exploitable.

And it is. It doesn't matter if you remove something or if you think (!) you are improving the situation. The final patch we all agree on might be perfect. It might also be broken. The problem is that it is our custom patch. Things like this need to be dealt with in one spot and one spot only. It's taken for granted in every other aspect of software development that DRY is the way to go. Yet somehow when it comes to security, it shall suddenly be better for everyone to do their own thing? Take certificate validation for example. It's also more or less broken by default in OpenSSL. To get it right, a lot of projects depending on OpenSSL implemented their own certificate validation code. Because it's a highly non-trivial job, the result is: Many different implementations, many of them incorrect, not secure, not by the book/RFC. It's this kind of proliferation that hurts security in general much, much more in the end.

>> It hurts even more that in such cases everyone will start pointing fingers,
>> asking: "Why didn't you stick to the library defaults???"
>
> As opposed to asking: "Why didn't you remove known weak ciphers and exploitable
> protocol features from the defaults when you were warned about them???"

Because it is a very bad idea trying to fix OpenSSL inside of Ruby! We need to ask OpenSSL developers to fix it centrally, so that everyone can benefit from the change.

>> I would prefer a whitelisting approach instead of blacklisting as in the
>> patch that was proposed. Blacklisting is never airtight, as it doesn't protect
>> us from future shitty algorithms creeping in.

> I wonder. In the blacklisting case, we're not required to make guesses about
> the future. We're merely switching off already-known weak or exploitable
> features.
>
> Whitelisting goes a step further, gambling that what we know today about the
> subset of defaults considered superior will continue to hold true down the road.
>
> It's not clear to me that's better than the more conservative step of simply
> blacklisting specific defaults already known to be problematic.

Whitelisting is the preferred approach. Because at every point, you know what you're dealing with [1]. Take SHA-3 for example. If by surprise, it turns out to be severely broken, a blacklist defined today will happily accept SHA-3 in the future because we didn't know better today. Of course we need to adapt the whitelist, too, once it turns out that one of the whitelisted algorithms gets broken. Or if new algorithms are developed that are considered superior. Still, it's always easier to reason about a finite set of algorithms than about a probably unbounded complementary set. Things get even more complicated in our case because the blacklist result is potentially affected by changes in OpenSSL itself. A whitelist would not be affected by external influences.

Having said that, the proposed patch still includes some form of blacklisting. I will get rid of it and make it a pure whitelist, just listing the algorithms one by one.

[1] http://www.testingsecurity.com/whitelists_vs_blacklists

#27 - 01/23/2014 11:41 PM - spatulasnout (B Kelly)

Martin.Bosslet@gmail.com wrote:

> B Kelly wrote:
>
> > The Debian maintainer removed lines of code from the OpenSSL PRNG
> > implementation. [1]
> >
> > This is hardly in the same category as tightening the defaults to exclude
> > specific ciphers or protocol features already known to be weak or exploitable.
>
> And it is. It doesn't matter if you remove something or if you think (!)
> you are improving the situation. The final patch we all agree on might be
> perfect. It might also be broken. The problem is that it is our custom patch.
> Things like this need to be dealt with in one spot and one spot only. It's
> taken for granted in every other aspect of software development that DRY is
> the way to go. Yet somehow when it comes to security, it shall suddenly be
> better for everyone to do their own thing?

I think we're talking at cross-purposes. Your arguments focus on what would
be ideal: an upstream patch by OpenSSL. I think nobody disagrees that would be
ideal, and presumably most among us are familiar with the downsides of maintaining
downstream patches.

> > > It hurts even more that in such cases everyone will start pointing fingers,
> > > asking: "Why didn't you stick to the library defaults???"
> >
> > As opposed to asking: "Why didn't you remove known weak ciphers and exploitable
> > protocol features from the defaults when you were warned about them???"
>
> Because it is a very bad idea trying to fix OpenSSL inside of Ruby!

The phrasing seems dramatic. Are we fixing OpenSSL inside of Ruby? Or are we
adopting a policy for Ruby that stipulates our defaults should favor security over
maximum compatibility?

> We need to ask OpenSSL developers to fix it centrally, so that everyone can
> benefit from the change.

Ideally. Though one can imagine the possibility that OpenSSL may always prefer
maximum compatibility by default. In which case, the Ruby policy might simply
differ.

> > > I would prefer a whitelisting approach instead of blacklisting as in the
> > > patch that was proposed. Blacklisting is never airtight, as it doesn't protect

> > > us from future shitty algorithms creeping in.
> >
> > I wonder. In the blacklisting case, we're not required to make guesses about the future. We're merely switching off already-known weak or exploitable features.
> >
> > Whitelisting goes a step further, gambling that what we know today about the subset of defaults considered superior will continue to hold true down the road.
> >
> > It's not clear to me that's better than the more conservative step of simply blacklisting specific defaults already known to be problematic.
>
> Whitelisting is the preferred approach. Because at every point, you know what you're dealing with [1].
> [...]
> [1] http://www.testingsecurity.com/whitelists_vs_blacklists

Sorry, I didn't explain myself properly here.

I get whitelisting vs. blacklisting in principle.

The thrust of your arguments had seemed to be that we should be biased toward trusting upstream (OpenSSL) to get things right in general, and my reasoning was that a blacklist seemed most conservatively aligned with that approach.

However, if our position instead is we don't trust upstream at all, and we will be actively maintaining our own whitelist, then sure: whitelist sounds good.

Regards,

Bill

#28 - 01/24/2014 02:47 AM - shyouhei (Shyouhei Urabe)

B Kelly wrote:

> Martin.Bosslet@gmail.com wrote:
>
> > B Kelly wrote:
> >
> > > The Debian maintainer removed lines of code from the OpenSSL PRNG implementation. [1]
> > >
> > > This is hardly in the same category as tightening the defaults to exclude specific ciphers or protocol features already known to be weak or exploitable.
> >
> > And it is. It doesn't matter if you remove something or if you think (!) you are improving the situation. The final patch we all agree on might be perfect. It might also be broken. The problem is that it is our custom patch. Things like this need to be dealt with in one spot and one spot only. It's taken for granted in every other aspect of software development that DRY is the way to go. Yet somehow when it comes to security, it shall suddenly be better for everyone to do their own thing?
>
> I think we're talking at cross-purposes. Your arguments focus on what would be ideal: an upstream patch by OpenSSL. I think nobody disagrees that would be ideal, and presumably most among us are familiar with the downsides of maintaining downstream patches.

Then how can it be legitimate for you to blame Debian people?
I don't wanna be raped like them.

> > > > It hurts even more that in such cases everyone will start pointing fingers, asking: "Why didn't you stick to the library defaults???"
> > >
> > > As opposed to asking: "Why didn't you remove known weak ciphers and exploitable protocol features from the defaults when you were warned about them???"

> > Because it is a very bad idea trying to fix OpenSSL inside of Ruby!
>
> The phrasing seems dramatic. Are we fixing OpenSSL inside of Ruby? Or are we
> adopting a policy for Ruby that stipulates our defaults should favor security over
> maximum compatibility?
>
> > We need to ask OpenSSL developers to fix it centrally, so that everyone can
> > benefit from the change.
>
> Ideally. Though one can imagine the possibility that OpenSSL may always prefer
> maximum compatibility by default. In which case, the Ruby policy might simply
> differ.

Listen. I know you want to save the world. If you are serious
about changing the picture don't care about trivial entities like
Debian or Ruby. There are literally tens of thousands of wild
applications that use OpenSSL out there and you can't change them
at once. You have to focus on the real problem to achieve your
goal. The real problem here is the OpenSSL's providing insecure
menus and slow to fix them.

If you are not interested in securing the world and you want just
to fuck us, it's enough. Leave here.

> However, if our position instead is we don't trust upstream at all, and we will
> be actively maintaining our own whitelist, then sure: whitelist sounds good.

Stop using untrusted upstream. Now. Just fix it.

#29 - 01/24/2014 03:25 AM - kivikakk (Ashe Connor)

Shyouhei Urabe wrote:

> I don't wanna be raped like them.

How on earth can you think this is appropriate?

#30 - 01/24/2014 04:38 AM - spatulasnout (B Kelly)

shyouhei@ruby-lang.org wrote:

> B Kelly wrote:
>
> > I think we're talking at cross-purposes. Your arguments focus on what would
> > be ideal: an upstream patch by OpenSSL. I think nobody disagrees that would be
> > ideal, and presumably most among us are familiar with the downsides of maintaining
> > downstream patches.
>
> Then how can it be legitimate for you to blame Debian people?
> I don't wanna be raped like them.

Interesting. I feel I must be communicating unclearly.

I'm not someone who blamed Debian. (It's my preferred Linux distro.) Indeed, the
Debian maintainer who removed lines of code affecting the OpenSSL PRNG first
posted on the OpenSSL mailing list explaining his situation and asked if it was
OK to remove the code.

As I wrote in an earlier post, I think the details of what transpired in the
Debian/OpenSSL blunder are interesting.

Particularly, I think the details show it's difficult to point fingers at a specific
person or part of the process in the Debian/OpenSSL situation. Mistakes were made;
and yet the actions taken at each discrete step in the process seemed fairly
reasonable.

And in that /particular/ sense I recognize the parallels being drawn to the debate here about hardening the OpenSSL defaults for Ruby.

My position has simply been that I regard the following scenarios as categorically
distinct:

1. "I don't know what these lines of code in OpenSSL do, but Valgrind complains.
   Is it OK if I remove them?"

2. "SSLv2, TLS compression, and certain specific ciphers are regarded by the
   security community as weak or exploitable. Is it reasonable and beneficial to
   Ruby users if we exclude them from our defaults?"

To me, there appears to be a vast distance between #1 and #2. My recent posts on
this thread have been in part an attempt to understand the opposing view by
eliciting responses from those who disagree.

Regards,

Bill

#31 - 01/24/2014 04:52 AM - shyouhei (Shyouhei Urabe)

Yuki Izumi wrote:

> Shyouhei Urabe wrote:
>
>> I don't wanna be raped like them.
>
> How on earth can you think this is appropriate?

Maybe not, because I wasn't intend to attack women (but you replied
like this). Sorry about this. I don't know how to express this
situation then. Very depressed.

#32 - 01/24/2014 06:49 PM - mame (Yusuke Endoh)

Cooperatively with some committers, I investigated the current condition of default settings in OpenSSL (and OS X). It is very complicated. Correct me if I'm wrong.

Summary:

- [IMPORTANT] Emboss's patch does NOT always work. There is no way Ruby can fix the issue if OpenSSL 0.9.8 built with zlib is used.
- The current OpenSSL binary bundled by OS X, has secure defaults even without Emboss's patch.
- The current OpenSSL binary provided by MacPorts (and maybe homebrew), has insecure defaults. Emboss's patch will help in this case.

Details:

- I'm assuming that TLS compression allows the CRIME attack.
- OpenSSL 0.9.8 or later enables TLS compression by default only when it is built with zlib. The current latest version (1.0.1f) still does.
- OpenSSL 0.9.8 does NOT provide an option SSL_OP_NO_COMPRESSION to disable TLS compression. Without it, there is no way to fix the issue. Emboss's patch does NOT work since it uses the option.
- OpenSSL 0.9.8 is bundled with OS X, and is NOT built with zlib. So there is no risk of the CRIME attack.
- OpenSSL 1.0.1 is provided by MacPorts, and IS built with zlib. So the CRIME attack could occur, I think. Emboss's patch will work in this case.

Note that I could be wrong because I'm newbie for OpenSSL, Verification and correction are really welcome.

If I'm correct, when we release the patch, it should be emphasized that the patch is NOT perfect, e.g., it will not help those who are using the old OpenSSL 0.9.8 provided by MacPorts. Eventually, each user must choose a secure OS/distribution carefully, and update it appropriately.

Thank you,

--
Yusuke Endoh mame@tsg.ne.jp

#33 - 01/24/2014 07:55 PM - dbussink (Dirkjan Bussink)

First of all thanks to Martin for working on the white list. I still believe specifying defaults here is the best of bad choices to make.

Regarding defaults on OS X, tools like RVM, rbenv and chruby are used a lot. All either build OpenSSL themselves (RVM), or include instructions to install a recent OpenSSL, often through Homebrew. So this means that people using Ruby on OS X often have a recent OpenSSL that is at least 1.0.1. Changing these defaults would improve the security of all these users.

I've also checked on Ubuntu 12.04 and they provide 1.0.1 as well, but the defaults aren't safe there either. With these changes it works properly there and gives people a safe default.

I agree we can't guarantee every user a perfect situation because of different OS's and distributions, but changes the defaults improved the security greatly for a lot of Ruby users. Also CRIME isn't really the biggest problem I think. There are much bigger problems. For example on a local Ruby compile with rbenv, I see a cipher like TLS_ECDH_anon_WITH_3DES_EDE_CBC_SHA enabled. This cipher allows for not validating the server side certificate, which means someone can launch a man in the middle attack without me knowing. I think preventing these changes is much more important and we can also improve security for users of 0.9.8 against these kinds of attack.

--
Dirkjan Bussink

#34 - 01/24/2014 11:11 PM - jmhodges (Jeff Hodges)

Why do people think it's impossible to disable TLS compression on 0.9.8 with zlib installed?

#35 - 01/25/2014 05:33 AM - shyouhei (Shyouhei Urabe)

B Kelly wrote:

> Interesting. I feel I must be communicating unclearly.
>
> I'm not someone who blamed Debian. (It's my preferred Linux distro.) Indeed, the
> Debian maintainer who removed lines of code affecting the OpenSSL PRNG first
> posted on the OpenSSL mailing list explaining his situation and asked if it was
> OK to remove the code.
>
> As I wrote in an earlier post, I think the details of what transpired in the
> Debian/OpenSSL blunder are interesting.
>
> Particularly, I think the details show it's difficult to point fingers at a specific
> person or part of the process in the Debian/OpenSSL situation. Mistakes were made;
> and yet the actions taken at each discrete step in the process seemed fairly
> reasonable.
>
> And in that /particular/ sense I recognize the parallels being drawn to the
> debate here about hardening the OpenSSL defaults for Ruby.
>
> My position has simply been that I regard the following scenarios as categorically
> distinct:
>
> 1. "I don't know what these lines of code in OpenSSL do, but Valgrind complains.
>   Is it OK if I remove them?"
>
> 2. "SSLv2, TLS compression, and certain specific ciphers are regarded by the
>   security community as weak or exploitable. Is it reasonable and beneficial to
>   Ruby users if we exclude them from our defaults?"
>
> To me, there appears to be a vast distance between #1 and #2. My recent posts on
> this thread have been in part an attempt to understand the opposing view by
> eliciting responses from those who disagree.
>
> Regards,
>
> Bill

Alright, I see you are not blaming Debian people. Thank you.

But I see "I don't know what these lines of code in OpenSSL do, but Valgrind
complains." is a completely valid reason to fix something. In fact I have
just read the patch denbian introduced and still see no problem on it. So I
can't draw a line between #1 and #2. They are equally true. And the history
tells the patch was wrong; how can you say #2 is OK?

Recap. Ruby is not just requested to have a particular patch. Ruby is
requested to act as a sanity proxy over OpenSSL to prevent it from going mad.
Without any patch you CAN operate Ruby safely already, right? But you say
that's not sufficient. You request us to provide Ruby that you CANT fail.

Then how can we say it's safe? I see no way but OpenSSL itself to get sane.
No one else can be perfect. I believe the person behind Debian's failed patch
was far more skillfull than me, perhaps anyone on this thread. That helped
nothing. Those patches proposed here SEEMS to provide adequate defaults to

OpenSSL and so what? I think that doesn't finish this story. Because no one
can say those patches are ultimate solutions. And the request here is for
us to provide ultimate solution for users. If not, Ruby provides something
already. That's not a good option maybe, but that also applies to the patches
here.

#36 - 01/25/2014 06:24 AM - shyouhei (Shyouhei Urabe)

I can settle for compromise to introduce the proposed patch if everyone agrees to give up requesting ruby to be secure by default; it HAPPEN TO become what you call secure and that's not a future-guaranteed status. The devs are not going to actively keep it so. Right?

#37 - 01/25/2014 06:52 AM - charliesome (Charlie Somerville)

Shyouhei Urabe wrote:

> I can settle for compromise to introduce the proposed patch if everyone agrees to give up requesting ruby to be secure by default; it HAPPEN TO become what you call secure and that's not a future-guaranteed status. The devs are not going to actively keep it so. Right?

I think this is a good compromise.

Nobody is requesting Ruby to be perfectly secure, or to provide ultimate solutions to security problems. But if there is a simple thing we can do to improve security for most of users, then we should do that.

#38 - 01/26/2014 06:48 AM - sag47 (Sam Gleske)

Take a leaf out of Apache's book. Give sane defaults (or use OpenSSL) but give users a way to configure ciphers.

http://httpd.apache.org/docs/2.2/mod/mod_ssl.html#sslciphersuite

#39 - 01/26/2014 02:16 PM - shyouhei (Shyouhei Urabe)

Sam Gleske wrote:

> Take a leaf out of Apache's book. Give sane defaults (or use OpenSSL) but give users a way to configure ciphers.
>
> http://httpd.apache.org/docs/2.2/mod/mod_ssl.html#sslciphersuite

Why you can't say that to OpenSSL people?

Is it because you think it is easier to trample down Ruby than OpenSSL?

In spite of our multiple explicit decline we are still about to be forced unwilling thing and by ding so also forced to take risk of being pointed by fingers asking why did you do that. Why is it us? Why?

Follow the right path. Please. Fix OpenSSL first.

#40 - 01/27/2014 01:47 AM - MartinBosslet (Martin Bosslet)

Yusuke Endoh wrote:

> Cooperatively with some committers, I investigated the current condition of default settings in OpenSSL (and OS X). It is very complicated. Correct me if I'm wrong.

Thanks for that!

> - [IMPORTANT] Emboss's patch does NOT always work. There is no way Ruby can fix the issue if OpenSSL 0.9.8 built with zlib is used.
>
> If I'm correct, when we release the patch, it should be emphasized that the patch is NOT perfect, e.g., it will not help those who are using the old OpenSSL 0.9.8 provided by MacPorts. Eventually, each user must choose a secure OS/distribution carefully, and update it appropriately.

That's the kind of proliferation I was talking about. Although Ruby OpenSSL is a direct consumer of OpenSSL, we already face a more complicated task of trying to fix the problem on our end. a) I want to apologize for overlooking this, but then b) I want to emphasize again that this is exactly what I was talking about earlier. I would be OK with patching the defaults, but only in line with what mame (Yusuke Endoh) and shyouhei (Shyouhei Urabe) already said: it must be clear to everyone that this patch will not relieve you from keeping OpenSSL up to date. This should never be our goal, and I hope it's not what anyone is asking from us, because I believe it's simply impossible. It would mean trying to merge every security bugfix ever made in all versions of OpenSSL at least since 0.9.6 in a way that yields one single patch that fits all versions that might be used with Ruby OpenSSL. That's insane! If someone is willingly running OpenSSL 0.9.8 or even earlier, it should not become our burden to fix this.

Like shyouhei (Shyouhei Urabe), I still believe the best solution would be asking OpenSSL to fix this for all of us. But if the majority is for patching, OK. I'll update the patch with a real whitelist, appending it for review here.

#41 - 01/27/2014 03:34 AM - mame (Yusuke Endoh)

Sam Gleske wrote:

> Take a leaf out of Apache's book. Give sane defaults (or use OpenSSL) but give users a way to configure ciphers.
>
> http://httpd.apache.org/docs/2.2/mod/mod_ssl.html#sslciphersuite

I can't understand your point at all. Please elaborate.

According to the URL you gave, Apache's default seems to include ciphers that is called "weak" in this thread.

SSLCipherSuite ALL:!ADH:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP

(Caveat: I'm not sure how weak they are.)

Also, Ruby OpenSSL actually provides a way to configure ciphers, as Apache does.

http://ruby-doc.org/stdlib-2.0/libdoc/openssl/rdoc/OpenSSL/SSL/SSLContext.html#method-i-ciphers-3D

--
Yusuke Endoh mame@tsg.ne.jp

#42 - 01/27/2014 03:44 AM - mame (Yusuke Endoh)

Martin Bosslet wrote:

> a) I want to apologize for overlooking this

Ah, you don't need to apologize at all! I just wanted to clarify what is relieved and what is not.

> Like shyouhei (Shyouhei Urabe), I still believe the best solution would be asking OpenSSL to fix this for all of us.

Me too, but I'm curious about the reason why OpenSSL people don't "improve" the defaults.
(OT: insecure default is not a bug itself; I'd like to use "improve" rather than "fix".)

One possible answer: They are simply unable, due to various reasons such as compatibility, lack of resource, etc. They have intention of doing that in the future. There is no problem in this case.

Another answer: Their goal is just to provide toolkit, and secure defaults are out of scope. In this case, they won't improve it. (I have no intention of blaming them. Deciding secure defaults is a hard task. Effort allocation looks quite reasonable to me.) Anyway, I'm afraid if just waiting will not solve our issue in this case.

--
Yusuke Endoh mame@tsg.ne.jp

#43 - 01/27/2014 05:51 AM - spatulasnout (B Kelly)

shyouhei@ruby-lang.org wrote:

> Alright, I see you are not blaming Debian people. Thank you.
>
> But I see "I don't know what these lines of code in OpenSSL do, but Valgrind
> complains." is a completely valid reason to fix something. In fact I have
> just read the patch denbian introduced and still see no problem on it. So I
> can't draw a line between #1 and #2. They are equally true. And the history
> tells the patch was wrong; how can you say #2 is OK?

To me, the difference hinges on the premise that the defaults function by
selecting from a set of features which are intended to be enabled or disabled
/by design/.

For example, the first time my servers were scanned with tools like Nessus or
OpenVAS, I received reports similar to the following:

Your https is vulnerable due to old protocols and weak ciphers.

Remove vulnerable Apache SSL defaults as follows:

enable SSLv3 and TLSv1, but not SSLv2
SSLProtocol all -SSLv2

exclude weak ciphers
SSLCipherSuite
!EXPORT40:!EXPORT56:!LOW:!ADH:DHE-RSA-AES256-SHA:EDH-RSA-DES-CBC3-SHA:DHE-RSA-AES128-SHA:AES256-SHA:DES-CBC3-SHA:AES128-SHA:RC4-SHA

This seems to me very different than if the report were to tell me instead:
"Apply the following ad hoc patch to source code that affects the OpenSSL
internals."

Rather, the protocol list and cipher suite configuration seems intended to be
customizable by design.

> Recap. Ruby is not just requested to have a particular patch. Ruby is
> requested to act as a sanity proxy over OpenSSL to prevent it from going mad.
> Without any patch you CAN operate Ruby safely already, right? But you say
> that's not sufficient. You request us to provide Ruby that you CANT fail.

I believe my own position to be less extreme: I have not been arguing for a
Ruby that can't fail; just a Ruby whose default configuration excludes already-
known weak ciphers or protocol versions.

However, since I'm not a security expert, my argument has been based on the
assumption that information provided by tools like Nessus (and various
security blogs) is correct.

My presumption had been that customizing Ruby's OpenSSL defaults is something
which could be accomplished in approximately as simple a manner as the Apache
SSL customization above.

(It sounds like this has mostly been true, apart from some difficulty arising
from supporting older OpenSSL versions.)

> Those patches proposed here SEEMS to provide adequate defaults to
> OpenSSL and so what? I think that doesn't finish this story. Because no one
> can say those patches are ultimate solutions. And the request here is for
> us to provide ultimate solution for users.

I'm not sure which posts have been advocating an ultimate solution?

My understanding is that once specific protocol versions or ciphers have been
identified by security experts as weak or exploitable, there's no plausible
future in which this will cease to be true.

So my reasoning is, if specific ciphers or protocol versions are known now
to be weak, and will continue to be weak until the end of time, then it would
seem to benefit Ruby users if these were disabled by default.

(Of course, my viewpoint is the same should also be true for Apache users, but
I've never joined any Apache development mailing lists.)

Regards,

Bill

#44 - 01/27/2014 07:56 AM - shyouhei (Shyouhei Urabe)

Just wanted to tell @emboss that (step aside from my position here)
the patch he proposed itself seems adequate for the purpose. 0.9.8
with zlib provies no way to disable compression so you cannot save
that case anyway.

My seeing it ok doesn't mean it is a sane default though; I cannot
swear such thing.

#45 - 01/28/2014 12:00 AM - kosaki (Motohiro KOSAKI)

- ruby -v changed from ruby 2.1.0p0 (2013-12-25 revision 44422) [x86_64-darwin12] to -

On Sun, Jan 26, 2014 at 10:44 PM, mame@tsg.ne.jp wrote:

> Issue #9424 has been updated by Yusuke Endoh.
>
> Martin Bosslet wrote:

> > a) I want to apologize for overlooking this
>
> Ah, you don't need to apologize at all! I just wanted to clarify what is relieved and what is not.
>
> > Like shyouhei (Shyouhei Urabe), I still believe the best solution would be asking OpenSSL to fix this for all of us.
>
> Me too, but I'm curious about the reason why OpenSSL people don't "improve" the defaults.
> (OT: insecure default is not a bug itself; I'd like to use "improve" rather than "fix".)
>
> One possible answer: They are simply unable, due to various reasons such as compatibility, lack of resource, etc. They have intention of doing that in the future. There is no problem in this case.
>
> Another answer: Their goal is just to provide toolkit, and secure defaults are out of scope. In this case, they won't improve it. (I have no intention of blaming them. Deciding secure defaults is a hard task. Effort allocation looks quite reasonable to me.) Anyway, I'm afraid if just waiting will not solve our issue in this case.

I'm afraid I'm missing something. But I'd like to ask first. Why do
nobody ask OpenSSL first?
They only can answer their intension. I don't think debate a guess on
this list is a good idea.
I believe the best way is a fixing by OpenSSL because, as you pointed
out, either Ruby and
OpenSSL can not make secure Ruby + old OpenSSL case. Therefore, to
workaround for old
OpenSSL is a pointless.

I agree security is important and Ruby sometimes accepted a workaround
patch and should
do in the future too, if we really need to do.
But I disagree just to continue a guess talk. Fixing right place is
always better than a workaround.

I hope my stand point is close to yours.

#46 - 01/29/2014 10:50 PM - MartinBosslet (Martin Bosslet)

After discussing the issue with Dirkjan and also internally, I feel that updating our own TLS cipher list is the best option we have at this point. So far, there is no indication as to when OpenSSL might update the defaults themselves. The hope that "sitting this out" solves the problem eventually is risky. Even if OpenSSL updates their default list, these updates take a long time to reach end users. This would mean exposing Ruby users to these risks for a long time.

By not proactively making the decision for our users, we will put that burden on their shoulders, which is an even worse situation. Maintaining a list of suitable ciphers within Ruby OpenSSL seems like the best option in this situation, especially with OpenSSL providing explicit support for this kind of thing. But I'm afraid there currently is no easy way to solve the compression issue with older versions of OpenSSL, any attempt made to solve it will likely mean working actively against OpenSSL internals.

In conclusion, we believe it is best to apply a patch like the ones proposed, at least as an interim solution. As always, Ruby users are still expected to keep their systems up to date. This includes running recent versions of operating systems and distributions that are still receiving security updates. We cannot work around old OpenSSL versions that come distributed with outdated operating systems that do not allow for improving these defaults.

We also believe that in the long term secure defaults should be provided by OpenSSL. This would be in the interest not just of Ruby users alone, but any direct or indirect user of OpenSSL in general. That's why we will also make an effort to ask them for updating the defaults on their end.

#47 - 02/02/2014 10:37 PM - MartinBosslet (Martin Bosslet)

- File change_ssl_defaults.2.diff added

Attached the last patch updated with a whitelist of 30 ciphers. The rationale:

- prefer ephemeral DH to enable forward secrecy
- prefer GCM over CBC mode
- prefer AES-128 over AES-256 (due to performance mostly, both are secure)
- prefer ECDSA over RSA over DSS/DSA
- prefer SHA256 or higher over SHA1
- no MD5, AES only (no RC4)

Although RC4 is not supported at all, backwards compatibility should be given by supporting AES128-SHA. Although AES128-SHA uses CBC mode (as do other ciphers on this list), all recent versions of OpenSSL (the C library) define SSL_OP_DONT_INSERT_EMPTY_FRAGMENTS. This constant is negated in the configuration if available (i.e. empty fragments are inserted) in order to protect users from attacks against CBC mode like BEAST. Here's the list of ciphers for review and comments:

```
      ECDHE-ECDSA-AES128-GCM-SHA256
      ECDHE-RSA-AES128-GCM-SHA256
      ECDHE-ECDSA-AES256-GCM-SHA384
      ECDHE-RSA-AES256-GCM-SHA384
      DHE-RSA-AES128-GCM-SHA256
      DHE-DSS-AES128-GCM-SHA256
      DHE-RSA-AES256-GCM-SHA384
      DHE-DSS-AES256-GCM-SHA384
      ECDHE-ECDSA-AES128-SHA256
      ECDHE-RSA-AES128-SHA256
      ECDHE-ECDSA-AES128-SHA
      ECDHE-RSA-AES128-SHA
      ECDHE-ECDSA-AES256-SHA384
      ECDHE-RSA-AES256-SHA384
      ECDHE-ECDSA-AES256-SHA
      ECDHE-RSA-AES256-SHA
      DHE-RSA-AES128-SHA256
      DHE-RSA-AES256-SHA256
      DHE-RSA-AES128-SHA
      DHE-RSA-AES256-SHA
      DHE-DSS-AES128-SHA256
      DHE-DSS-AES256-SHA256
      DHE-DSS-AES128-SHA
      DHE-DSS-AES256-SHA
      AES128-GCM-SHA256
      AES256-GCM-SHA384
      AES128-SHA256
      AES256-SHA256
      AES128-SHA
      AES256-SHA
```

The mapping to corresponding TLS constants defined by the RFCs can be found at https://www.openssl.org/docs/apps/ciphers.html.

#48 - 03/05/2014 12:14 AM - masterleep (Bill Lipa)

Is there anything Ruby users should be doing in the meantime to protect themselves? Does the lack of urgency around this update (1 month since last update) imply that there isn't a real problem here?

#49 - 03/06/2014 01:44 AM - Anonymous

- Status changed from Assigned to Closed

- % Done changed from 0 to 100

Applied in changeset r45274.

- lib/openssl/ssl.rb: Explicitly whitelist the default SSL/TLS ciphers. Forbid SSLv2 and SSLv3, disable compression by default. Reported by Jeff Hodges. [ruby-core:59829] [Bug #9424]

#50 - 03/06/2014 01:52 AM - MartinBosslet (Martin Bosslet)

The patch has been committed. After discussing the issue with Dirkjan, the decision was made to additionally add

```
ECDHE-ECDSA-RC4-SHA
ECDHE-RSA-RC4-SHA
RC4-SHA
```

to the end of the list because RC4 has been widely deployed to mitigate the BEAST attack.

#51 - 03/17/2014 06:55 PM - naruse (Yui NARUSE)

- Related to Backport #9640: Please backport SSL fixes to 2.1 added

#52 - 10/14/2014 09:13 PM - stouset (Stephen Touset)

This patch looks broken.

```
:options => -> {
  opts = …
  opts |= OpenSSL::SSL::OP_NO_SSLv3 if defined?(OpenSSL::SSL::OP_NO_SSLv3)
}.call
```

If that last constant isn't defined, the lambda will return nil and no options will be overridden.

#53 - 10/15/2014 06:18 AM - reed (Reed Loden)

This patch is even more important now that SSLv3 has basically been completely deprecated by the POODLE attack.

#54 - 10/22/2014 01:24 PM - nagachika (Tomoyuki Chikanaga)

- Backport changed from 1.9.3: UNKNOWN, 2.0.0: UNKNOWN, 2.1: UNKNOWN to 1.9.3: REQUIRED, 2.0.0: REQUIRED, 2.1: REQUIRED

According to the discussion at #9640, we decide to backport the patch into stable branches.

#55 - 10/22/2014 01:34 PM - nagachika (Tomoyuki Chikanaga)

Hello, Stephen

Good catch! Thank you so much.
Your right. I will check in my patch.

```
--- a/ext/openssl/lib/openssl/ssl.rb
+++ b/ext/openssl/lib/openssl/ssl.rb
@@ -64,6 +64,7 @@ module OpenSSL
           opts |= OpenSSL::SSL::OP_NO_COMPRESSION if defined?(OpenSSL::SSL::OP_NO_COMPRESSION)
           opts |= OpenSSL::SSL::OP_NO_SSLv2 if defined?(OpenSSL::SSL::OP_NO_SSLv2)
           opts |= OpenSSL::SSL::OP_NO_SSLv3 if defined?(OpenSSL::SSL::OP_NO_SSLv3)
+          opts
         }.call
       }
```

#56 - 10/22/2014 01:43 PM - usa (Usaku NAKAMURA)

- Backport changed from 1.9.3: REQUIRED, 2.0.0: REQUIRED, 2.1: REQUIRED to 1.9.3: WONTFIX, 2.0.0: REQUIRED, 2.1: REQUIRED

Result of contemplation, I have decided that I don't apply this patch for 1.9.3 for compatibility.

#57 - 10/22/2014 02:18 PM - nagachika (Tomoyuki Chikanaga)

- Backport changed from 1.9.3: WONTFIX, 2.0.0: REQUIRED, 2.1: REQUIRED to 1.9.3: WONTFIX, 2.0.0: REQUIRED, 2.1: DONE

r45274, r45278, r45280 and r48097 are backported into ruby_2_1 branch at r48098.

#58 - 10/22/2014 05:15 PM - zzak (Zachary Scott)

- Status changed from Closed to Feedback

- Assignee changed from MartinBosslet (Martin Bosslet) to nagachika (Tomoyuki Chikanaga)

Why not backport to 1.9.3?

Is there a workaround we can tell 1.9 users about?

Should we also add something to NEWS?

#59 - 10/23/2014 03:28 AM - usa (Usaku NAKAMURA)

Zachary Scott wrote:

> Why not backport to 1.9.3?
>
> Is there a workaround we can tell 1.9 users about?

Users who still use 1.9.3 are assumed that in the situation extremely severe for the maintenance of compatibility.
So, I determined that to keep compatibility is the top priority.
Of course, I'll provide a patch for for users who wish it.

> Should we also add something to NEWS?

I think so.

#60 - 10/23/2014 09:59 AM - usa (Usaku NAKAMURA)

- Status changed from Feedback to Closed

Applied in changeset ruby-200:r48110.

---

merge revision(s) 45274,45278,45280,48097: [Backport #9424]

```
* lib/openssl/ssl.rb: Explicitly whitelist the default
  SSL/TLS ciphers. Forbid SSLv2 and SSLv3, disable
  compression by default.
  Reported by Jeff Hodges.
  [ruby-core:59829] [Bug #9424]

* test/openssl/test_ssl.rb: Reuse TLS default options from
  OpenSSL::SSL::SSLContext::DEFAULT_PARAMS.

* ext/openssl/lib/openssl/ssl.rb (DEFAULT_PARAMS): override
  options even if OpenSSL::SSL::OP_NO_SSLv3 is not defined.
  this is pointed out by Stephen Touset. [ruby-core:65711] [Bug #9424]
```

#61 - 10/23/2014 10:00 AM - usa (Usaku NAKAMURA)

Backported into ruby_2_0_0 at r48110.

#62 - 10/23/2014 10:01 AM - usa (Usaku NAKAMURA)

- Backport changed from 1.9.3: WONTFIX, 2.0.0: REQUIRED, 2.1: DONE to 1.9.3: WONTFIX, 2.0.0: DONE, 2.1: DONE

#63 - 10/23/2014 11:40 AM - usa (Usaku NAKAMURA)

BTW, these changes are only affect to the scripts which call OpenSSL::SSL:SSLContext#set_params.
Can we say "Yeh, we are safe!" in such a situation?

#64 - 10/23/2014 12:08 PM - nagachika (Tomoyuki Chikanaga)

Hi,

I investigated about usage of SSLContext in std libs.

- net/pop, net/imap
  - OpenSSL::SSL:SSLContext#set_params is called for setup
- net/smtp
  - set_params is not called.
- drb
  - set_params is not called. verify_mode is overridden by configuration.
- webrick
  - set_params is not called. verify_mode and options are overridden by configuration.

Hmm, OpenSSL::SSL::SSLContext#initialize should setup with DEFAULT_PARAMS?

#65 - 10/23/2014 12:20 PM - usa (Usaku NAKAMURA)

- Status changed from Closed to Feedback

#66 - 10/23/2014 12:27 PM - usa (Usaku NAKAMURA)

Incidentally, yesterday I was not aware about set_params.
If we decide to keep the status quo that Chikanaga-san described, there is no problem to backport to 1.9.3.

#67 - 10/24/2014 03:07 AM - usa (Usaku NAKAMURA)

- Status changed from Feedback to Closed

Applied in changeset ruby-193:r48121.

---

merge revision(s) 45274,45278,45280,48097: [Backport #9424]

```
* ext/openssl/lib/openssl/ssl-internal.rb (DEFAULT_PARAMS): override
  options even if OpenSSL::SSL::OP_NO_SSLv3 is not defined.
  this is pointed out by Stephen Touset. [ruby-core:65711] [Bug #9424]

* test/openssl/test_ssl.rb: Reuse TLS default options from
  OpenSSL::SSL::SSLContext::DEFAULT_PARAMS.

* lib/openssl/ssl-internal.rb: Explicitly whitelist the default
  SSL/TLS ciphers. Forbid SSLv2 and SSLv3, disable
  compression by default.
  Reported by Jeff Hodges.
  [ruby-core:59829] [Bug #9424]
```

#68 - 10/24/2014 03:10 AM - usa (Usaku NAKAMURA)

- Backport changed from 1.9.3: WONTFIX, 2.0.0: DONE, 2.1: DONE to 1.9.3: DONE, 2.0.0: DONE, 2.1: DONE

NaHi-san told me that this patch was for client use and not general-purpose, so I didn't have to worry.

So, I've backported it into ruby_1_9_3 at r48121.

Files

ruby_ssl.patch	1.08 KB	01/17/2014	jmhodges (Jeff Hodges)
change_ssl_defaults.diff	1.24 KB	01/22/2014	MartinBosslet (Martin Bosslet)
change_ssl_defaults.2.diff	2.13 KB	02/02/2014	MartinBosslet (Martin Bosslet)