# Ruby master - Bug #9569

## SecureRandom should try /dev/urandom first

02/26/2014 12:48 AM - cjcsuhta (Corey Csuhta)

| | | | |
|---|---|---|---|
| **Status:** | Closed | | |
| **Priority:** | Normal | | |
| **Assignee:** | | | |
| **Target version:** | | | |
| **ruby -v:** | | **Backport:** | |

**Description**

Right now, SecureRandom.random_bytes tries to detect an OpenSSL to use before it tries to detect /dev/urandom. I think it should be the other way around. In both cases, you just need random bytes to unpack, so SecureRandom could skip the middleman (and second point of failure) and just talk to /dev/urandom directly if it's available.

Is this a case of just re-ordering the two code chunks so that /dev/urandom is tried first?

Relevant lines: https://github.com/ruby/ruby/blob/trunk/lib/securerandom.rb#L59-L90

**Related issues:**

| | |
|---|---|
| Related to Ruby master - Bug #13885: Random.urandom □ securerandom □□□□ | **Closed** |
| Related to Ruby master - Bug #14716: SecureRandom throwing an error in Ruby 2... | **Open** |
| Related to Ruby master - Bug #15039: Random.urandom and SecureRandom arc4rand... | **Closed** |
| Related to Ruby master - Misc #17319: Rename Random.urandom to os_random and ... | **Rejected** |

### Associated revisions

**Revision abae70d6 - 01/20/2017 08:00 AM - shyouhei (Shyouhei Urabe)**

SecureRandom should try /dev/urandom first [Bug #9569]

```
    * random.c (InitVM_Random): rename Random.raw_seed to
      Random.urandom.  A quick search seems there are no practical use
      of this method than securerandom.rb so I think it's OK to rename
      but if there are users of it, this hunk is subject to revert.

    * test/ruby/test_rand.rb (TestRand#test_urandom): test for it.

    * lib/securerandom.rb (SecureRandom.gen_random): Prefer OS-
      provided CSPRNG if available. Otherwise falls back to OpenSSL.
      Current preference is:

      1. CSPRNG routine that the OS has; one of
         - getrandom(2),
         - arc4random(3), or
         - CryptGenRandom()
      2. /dev/urandom device
      3. OpenSSL's RAND_bytes(3)

      If none of above random number generators are available, you
      cannot use this module.  An exception is raised that case.
```

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@57384 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

**Revision 57384 - 01/20/2017 08:00 AM - shyouhei (Shyouhei Urabe)**

SecureRandom should try /dev/urandom first [Bug #9569]

```
* random.c (InitVM_Random): rename Random.raw_seed to
  Random.urandom.  A quick search seems there are no practical use
  of this method than securerandom.rb so I think it's OK to rename
  but if there are users of it, this hunk is subject to revert.

* test/ruby/test_rand.rb (TestRand#test_urandom): test for it.

* lib/securerandom.rb (SecureRandom.gen_random): Prefer OS-
  provided CSPRNG if available. Otherwise falls back to OpenSSL.
  Current preference is:
```

```
  1. CSPRNG routine that the OS has; one of
     - getrandom(2),
     - arc4random(3), or
     - CryptGenRandom()
  2. /dev/urandom device
  3. OpenSSL's RAND_bytes(3)

  If none of above random number generators are available, you
  cannot use this module.  An exception is raised that case.
```

**Revision 57384 - 01/20/2017 08:00 AM - shyouhei (Shyouhei Urabe)**

SecureRandom should try /dev/urandom first [Bug #9569]

```
* random.c (InitVM_Random): rename Random.raw_seed to
  Random.urandom.  A quick search seems there are no practical use
  of this method than securerandom.rb so I think it's OK to rename
  but if there are users of it, this hunk is subject to revert.

* test/ruby/test_rand.rb (TestRand#test_urandom): test for it.

* lib/securerandom.rb (SecureRandom.gen_random): Prefer OS-
  provided CSPRNG if available. Otherwise falls back to OpenSSL.
  Current preference is:

  1. CSPRNG routine that the OS has; one of
     - getrandom(2),
     - arc4random(3), or
     - CryptGenRandom()
  2. /dev/urandom device
  3. OpenSSL's RAND_bytes(3)

  If none of above random number generators are available, you
  cannot use this module.  An exception is raised that case.
```

**Revision 57384 - 01/20/2017 08:00 AM - shyouhei (Shyouhei Urabe)**

SecureRandom should try /dev/urandom first [Bug #9569]

```
* random.c (InitVM_Random): rename Random.raw_seed to
  Random.urandom.  A quick search seems there are no practical use
  of this method than securerandom.rb so I think it's OK to rename
  but if there are users of it, this hunk is subject to revert.

* test/ruby/test_rand.rb (TestRand#test_urandom): test for it.

* lib/securerandom.rb (SecureRandom.gen_random): Prefer OS-
  provided CSPRNG if available. Otherwise falls back to OpenSSL.
  Current preference is:

  1. CSPRNG routine that the OS has; one of
     - getrandom(2),
     - arc4random(3), or
     - CryptGenRandom()
  2. /dev/urandom device
  3. OpenSSL's RAND_bytes(3)

  If none of above random number generators are available, you
  cannot use this module.  An exception is raised that case.
```

**Revision a0acd82f - 02/24/2017 11:33 AM - rhe**

securerandom: fix up r57384

SecureRandom.gen_random_openssl still refers to Random.raw_seed, which
is renamed to Random.urandom by r57384.  [Bug #9569]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@57707 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

**Revision 57707 - 02/24/2017 11:33 AM - rhenium (Kazuki Yamaguchi)**

securerandom: fix up r57384

SecureRandom.gen_random_openssl still refers to Random.raw_seed, which

is renamed to Random.urandom by r57384.  [Bug #9569]

**Revision 57707 - 02/24/2017 11:33 AM - rhe**

securerandom: fix up r57384

SecureRandom.gen_random_openssl still refers to Random.raw_seed, which
is renamed to Random.urandom by r57384.  [Bug #9569]

**Revision 57707 - 02/24/2017 11:33 AM - rhe**

securerandom: fix up r57384

SecureRandom.gen_random_openssl still refers to Random.raw_seed, which
is renamed to Random.urandom by r57384.  [Bug #9569]

## History

**#1 - 02/26/2014 01:36 AM - akr (Akira Tanaka)**

*- Status changed from Open to Rejected*

/dev/urandom is not suitable to be used to generate directly session keys
and other application level random data which is generated frequently.

random(4) on GNU/Linux:

```
   The kernel random-number generator is designed to produce a small
   amount of high-quality seed material to seed a cryptographic pseudo-
   random number generator (CPRNG). It is designed for security, not
   speed, and is poorly suited to generating large amounts of random data.
   Users should be very economical in the amount of seed material that
   they read from /dev/urandom (and /dev/random); unnecessarily reading
   large quantities of data from this device will have a negative impact
   on other users of the device.
```

/dev/urandom should be used as "seed" for CPRNG.
OpenSSL do it.

/dev/urandom usage in securerandom.rb is not a good way.
So OpenSSL should be used at first.

**#2 - 02/26/2014 09:54 PM - cjcsuhta (Corey Csuhta)**

The random(4) manpage on Linux isn't accurate in this reguard. You **can** use it as more than just a seed source, and you can use it as frequently as you want.

On modern Linux, both /dev/random and /dev/urandom are CSPRNGs, and can be used safely (after system boot, see references). The only difference is that /dev/random attempts to keep some kind of measure of its available entropy, and will sometimes block if if feels unsatisfied about that. On FreeBSD, Unix, and OS X, there is no difference between /dev/random and /dev/urandom anymore, and the manpages on OS X at least don't include this "rate-limit" warning about /dev/urandom.

Two additional points:

OpenSSL seeds itself from /dev/urandom as you stated, but you could run a lot of OpenSSL processes on your system at one time and none of them would complain that your /dev/urandom is not currently to be trusted because you used it too much.

SecureRandom in Ruby will use /dev/urandom if OpenSSL is not available, based on the code snippet I linked in the original post. This is contrary to your statement that /dev/urandom is not safe for sessions, or frequent access. As currently implemented, SecureRandom **will** access /dev/urandom frequently if OpenSSL is not available.

References:
http://blog.cr.yp.to/20140205-entropy.html
http://sockpuppet.org/blog/2014/02/25/safely-generate-random-numbers/
https://developer.apple.com/library/mac/documentation/Darwin/Reference/ManPages/man4/random.4.html
http://security.stackexchange.com/questions/3936/is-a-rand-from-dev-urandom-secure-for-a-login-key

**#3 - 02/26/2014 10:01 PM - akr (Akira Tanaka)**

If you think a manpage is not accurate, please fix the manpage at first.

**#4 - 02/26/2014 11:32 PM - cjcsuhta (Corey Csuhta)**

Akira, can you address this point?

> SecureRandom in Ruby will use /dev/urandom if OpenSSL is not available, based on the code snippet I linked in the original post. This is contrary to your statement that /dev/urandom is not safe for sessions, or frequent access. As currently implemented, SecureRandom will access /dev/urandom frequently if OpenSSL is not available.

#### #5 - 02/27/2014 02:18 AM - akr (Akira Tanaka)

I said "/dev/urandom usage in securerandom.rb is not a good way." already.
It means securerandom.rb will consume too much entoropy if /dev/urandom is used directry.

It is not a big problem because most users use OpenSSL.
Also, we can say "Please install OpenSSL" if someone complain about the entoropy consumption.

Your proposal breaks this strategy.

#### #6 - 01/02/2016 01:48 AM - azet (Aaron Zauner)

*- Tracker changed from Feature to Bug*

*- Assignee set to ruby-core*

Hi,

This still seems to be the case according to the code available on GitHub.

I urge the core team to move to /dev/urandom. It is an urban-legend (as Thomas Ptacek notes in the sockpuppet.org blog-post referenced two years ago) that one should use /dev/random or even fiddle nor interfere with seeding or entropy on Linux. The kernel does this for you. The only instance where you actually might wanna add entropy is during boot-strap or first boot of embedded devices or (cloned) virtual machines. Ask any cryptographer about this and they'll tell you that random(4) is just plainly wrong and nobody cared to update it yet.

The issuer before me has linked to various cryptographers telling you in their blogs to use /dev/urandom. Why would you rather listen to an out-dated man page? Even how this works (CSPRNG and entropy) in the kernel has changed significantly since the man page was last updated.

An OS does not "run out of entropy". This is not how a CSPRNG works [0]. Think of these constructions like you'd with entropy when considering the 2nd law of thermodynamics. :)

Thanks,
Aaron

[0] Here a cryptographer explains it to you: http://crypto.stackexchange.com/a/12441

#### #7 - 01/02/2016 02:16 AM - nobu (Nobuyoshi Nakada)

SecureRandom without OpenSSL (or compatible alternatives) is nonsense.

#### #8 - 01/02/2016 06:57 AM - akr (Akira Tanaka)

Aaron Zauner wrote:

> The issuer before me has linked to various cryptographers telling you in their blogs to use /dev/urandom. Why would you rather listen to an out-dated man page? Even how this works (CSPRNG and entropy) in the kernel has changed significantly since the man page was last updated.

Please update the man page first, if it is really out-dated.

#### #9 - 01/02/2016 10:18 AM - azet (Aaron Zauner)

Nobuyoshi Nakada wrote:

> SecureRandom without OpenSSL (or compatible alternatives) is nonsense.

You evidently have no idea what you are talking about. Why would you want to use the PRNG that's local to OpenSSL? It can fail in many ways, is slower than the kernel and may introduce vulnerabilities. The OpenSSL PRNG is not even fork save!

https://wiki.openssl.org/index.php/Random_fork-safety
https://emboss.github.io/blog/2013/08/21/openssl-prng-is-not-really-fork-safe/

I'm not part of the Linux documentation team, why do you insist on updating the man page before you will fix a critical vulnerability? this is laughable. You are potentially harming many users of SecureRandom in Ruby.

#### #10 - 01/02/2016 02:36 PM - kosaki (Motohiro KOSAKI)

> You evidently have no idea what you are talking about. Why would you want to use the PRNG that's local to OpenSSL? It can fail in many ways, is >slower than the kernel and may introduce vulnerabilities. The OpenSSL PRNG is not even fork save!
>
> https://wiki.openssl.org/index.php/Random_fork-safety
> https://emboss.github.io/blog/2013/08/21/openssl-prng-is-not-really-fork-safe/

Your pointed issue was mainly investigated Ruby core team and, of course, it has been solved, at least, on Ruby.
Please don't rude. You just said you don't understand the issue.

#### #11 - 01/02/2016 03:11 PM - azet (Aaron Zauner)

The following is the output of your 'SecureRandom' construction vs. the Linux /dev/urandom facility:

http://nopaste.narf.at/show/EPVj9ETuMlcrCXKErsS6/
http://nopaste.narf.at/show/i0EJbkQrL3SXurfQZ524/

As you can see your RNG is less secure than the one my Debian installation serves via the kernel.

Sorry, I did not mean to be rude. Please reconsider on this issue, it's security relevant. Also you added an extra dependency: a bug in OpenSSL will now hurt your random number generation. And there is really no reason not to use /dev/urandom on Linux, BSD and Mac OS X these days - take a look at the kernel implementation:

https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/drivers/char/random.c
https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/drivers/char/hw_random

#### #12 - 01/02/2016 06:09 PM - kosaki (Motohiro KOSAKI)

First off, thank you for providing number. Then we can discuss scientific way.
blog pages are not considered a formal document.

However I can't reproduce your conclusion. In my result, securerandom has one
weak and /dev/urandom has two weaks.

https://gist.github.com/kosaki/3a9a9126cb39e601be2d

Of cource, this result doesn't mean urandom is crap. This just mean dieharder's output is unstable, I think.

#### #13 - 01/02/2016 07:40 PM - filippo (Filippo Valsorda)

Hi!

I happen to have just given a talk on urandom internals at 32C3: https://media.ccc.de/v/32c3-7441-the_plain_simple_reality_of_entropy

The main takeaway about urandom is that it's a CSPRNG, just like the OpenSSL one. (Just better, because the kernel can protect the entropy pool memory space better, the kernel one is continuously reseeded, and has a better security track record--see Debian.) So there is no security advantage, or actually any theoretical difference in using OpenSSL. In practice, urandom is significantly more secure.

What Ruby currently does is chaining two CSPRNGs, urandom and OpenSSL's. This makes problems twice as likely (not half), because now if any of the two fail, the entire system is compromised. If you just read from urandom, there's only one thing that can go wrong.

Another bad consequence of this scheme is that now there is no reliable failure detection. Take the case of exhausted file descriptors, where opening urandom is impossible, and let's say there's no getrandom support. An exception should be raised, as there's no recovery. But the current code will first ignore the failure of raw_seed, then call into OpenSSL which will hide the failure (which I argue is a OpenSSL API shortcoming, but I digress). If you just read from urandom, it's very clear when the system failed. (By the way, why adding the nanotime and pid at all? Nanotime and pid alone are not enough, and hide brokenness Debian-style, while if you have a real 16 byte of seed, that's enough and you don't need anything else.)

Finally, complexity is toxic. I *believe* that pre-7104a473, on Windows, when OpenSSL is installed, securerandom was only seeded with the time. (Because OpenSSL would take priority over the syscall, and OpenSSL is documented to seed itself automatically only on platforms that offer urandom. http://linux.die.net/man/3/rand_add I haven't checked the source.) This would be spectacularly bad (and I would actually be dropping a vulnerability right now considering that the patch is only one month old), but I'm not even sure that's the case after staring at code for a good hour. It's significantly bad that a reasonably experienced developer can't be sure of securerandom correctness easily. If you just read from urandom, it takes less than 30 minutes to audit the code.

I realize that we are some Internet strangers arguing against the Linux man page (which for the record was already recently amended to be FUD'y on urandom), but since this is all Open Source, we can thankfully discuss on the actual implementation instead of on what is written about it. If instead authority is what you are looking for, this slide in my 32C3 presentation has plenty, academic and not:
https://speakerdeck.com/filosottile/the-plain-simple-reality-of-entropy-at-32c3?slide=36

Finally, I invite you to check how other libraries like BoringSSL, Python's stdlib and Go's stdlib solve this problem. They all read from /dev/urandom.

#### #14 - 04/13/2016 09:02 AM - azet (Aaron Zauner)

There's now a new paper outlining RNG weaknesses in OpenSSL: https://eprint.iacr.org/2016/367.pdf

Again: Please switch to a proper RNG/seeding mechanism as suggested by multiple people earlier.

Thanks,
Aaron

#### #15 - 04/13/2016 03:47 PM - bascule (Tony Arcieri)

I would just like to +1 that the text on the random(4) page is incorrect. Regarding fixing it, this bug has been open on the Linux kernel bug tracker for 2 years:

https://bugzilla.kernel.org/show_bug.cgi?id=71211

I would look to something like libsodium's sysrandom implementation for how to write a modern portable secure RNG. On Linux it defaults to using the new getrandom() system call if available, and falls back on /dev/urandom if it is not:

https://github.com/jedisct1/libsodium/blob/master/src/libsodium/randombytes/sysrandom/randombytes_sysrandom.c

#### #16 - 04/15/2016 03:33 AM - shyouhei (Shyouhei Urabe)

I'm aware that current Linux urandom *happen to* be safe for our needs in current implementation. I'm also skeptical about OpenSSL's code quality in general. The problem preventing me to +1 this request is that I cannot find any statements inside Linux kernel that urandom is *meant* to be used like this. The only description I could find so far is (the argued) manpage, which says otherwise.

Does anyone have any pointers that officially describe this device?

#### #17 - 04/29/2016 08:43 AM - azet (Aaron Zauner)

A recent (2012) analysis of the RNG subsystem in the Linux Kernel can be found here: https://eprint.iacr.org/2012/251.pdf

The paper also describes, in detail, how random and urandom work. Apart from that, the code of the random char device itself is rather readable, I've linked to it above - it's not that difficult to understand for any proficient C programmer. I believe the Ruby core team should not have any problems in understanding the code.

https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/drivers/char/random.c
https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/drivers/char/hw_random

#### #18 - 04/30/2016 10:03 PM - mame (Yusuke Endoh)

Aaron, I don't mean to offend you, but I think you should lobby in the Linux kernel community.

https://bugzilla.kernel.org/show_bug.cgi?id=71211

#### #19 - 04/30/2016 10:24 PM - filippo (Filippo Valsorda)

I am completely puzzled by what is happening here.

A number of cryptographers and systems engineers stated publicly and loudly that /dev/urandom is the way to go.

**All other languages only expose /dev/urandom in the stdlib.** All other operating systems only have /dev/urandom equivalents.

A Linux subsystem maintainer is being stubborn, and is not explicitly stating in the manpage that /dev/urandom is now understood to be safe, and /dev/random a pointless interface. (However, that manpage has been iteratively improved and by now is just vague. Also, the recently added getrandom syscall behaves like urandom by default.)

So... why are you being stubborn *for* them?

(I get the "but if it's not documented it might break" point, but there's nothing in the ecosystem that would survive a weak urandom, everyone (else) is using urandom, there's no going back, don't worry.)

#### #20 - 05/01/2016 08:44 AM - shyouhei (Shyouhei Urabe)

Filippo Valsorda wrote:

> **All other languages only expose /dev/urandom in the stdlib.**

This is simply not true. Python has ssl.RAND_bytes().

#### #21 - 05/02/2016 09:22 AM - shyouhei (Shyouhei Urabe)

Note that, while I read through this thread again, no one is arguing /dev/urandom being insecure. No one states linux kernel source is unreadable. No one is skeptical about its current implementation. Just to be clear. The problem is whether it is designed to be eligible for a possibly high-frequency user-land access; which is not officially stated elsewhere, and the manpage VERY strongly denies.

It's just "everybody else's jumping off a bridge" is not enough for us to commit a wireless bungee jump.

#### #22 - 05/02/2016 06:08 PM - cjcsuhta (Corey Csuhta)

The Linux man page keeps getting mentioned, but what the current fallback code is actually doing is simply looking for any device named /dev/urandom on the host system:

https://github.com/ruby/ruby/blob/62b6e90d583e0a1c2be538a42640d69600608747/random.c#L459

The man page for random(4) on OS X, OpenBSD, and other UNIX-y OSes does not match the Linux man page. Every version of UNIX or Linux can have different text in their man pages. The actual code that implements /dev/urandom also varies across OSes.

If the man pages are the source of truth in this argument, you should be detecting the specific operating system to do what they say with their specific random device.

But of course, that is silly. I don't actually want Ruby to do that. I want Ruby to drop the use of OpenSSL for this operation. It was just worth noting this discrepancy.

#### #23 - 05/03/2016 03:24 AM - shyouhei (Shyouhei Urabe)

Corey Csuhta wrote:

> what the current fallback code is actually doing is simply looking for any device named /dev/urandom on the host system

This is considered "not a good way" years ago already in comment #1 (also #5). Read them again.

#### #24 - 05/03/2016 03:11 PM - azet (Aaron Zauner)

Shyouhei Urabe wrote:

> Filippo Valsorda wrote:
>
> > **All other languages only expose /dev/urandom in the stdlib.**
>
> This is simply not true. Python has ssl.RAND_bytes().

I'm sorry but your statement is completely and utterly false.

Python2's stdlib uses os.urandom:

- https://docs.python.org/2/library/random.html
- there's even a big message stating: Warning The pseudo-random generators of this module should not be used for security purposes. Use os.urandom() or SystemRandom if you require a cryptographically secure pseudo-random number generator.
- https://hg.python.org/cpython/file/2.7/Lib/random.py

Python3's stdlib uses os.urandom:

- https://docs.python.org/3/library/random.html
- https://hg.python.org/cpython/file/3.5/Lib/random.py

Another example; cryptography.io, the best Python crypto library I'm aware of uses os.urandom: https://cryptography.io/en/latest/random-numbers/

As for your mentioned ssl.RAND_bytes interface: https://docs.python.org/3/library/ssl.html#random-generation (In Python2.x these API's aren't even exposed to the user/developer)

```
ssl.RAND_bytes(num)
Return num cryptographically strong pseudo-random bytes. Raises an SSLError if the PRNG has not been seeded wi
th enough data or if the operation is not supported by the current RAND method. RAND_status() can be used to c
heck the status of the PRNG and RAND_add() can be used to seed the PRNG.

For almost all applications os.urandom() is preferable.

Read the Wikipedia article, Cryptographically secure pseudorandom number generator (CSPRNG), to get the requir
ements of a cryptographically generator.

New in version 3.3.

ssl.RAND_pseudo_bytes(num)
Return (bytes, is_cryptographic): bytes are num pseudo-random bytes, is_cryptographic is True if the bytes gen
erated are cryptographically strong. Raises an SSLError if the operation is not supported by the current RAND
method.

Generated pseudo-random byte sequences will be unique if they are of sufficient length, but are not necessaril
y unpredictable. They can be used for non-cryptographic purposes and for certain purposes in cryptographic pro
tocols, but usually not for key generation etc.
```

```
For almost all applications os.urandom() is preferable.

New in version 3.3.
```

Read again; see what it says? right, it says: For almost all applications os.urandom() is preferable. Because only a complete amateur would use other third-party APIs in Python for randomness.

I think I'm officially giving up on this one. Four months ago I warned the Ruby Core team that a bug in the OpenSSL PRNG can cause security problems for Ruby proper. This has happened of course. Still the team thinks it's a better approach to use a userland RNG instead of the one provided (and well maintained) in the respective kernels and accessible via system calls.

I'm a long-time Ruby developer, but I think I will never use Ruby again for *any* security-critical code parts. This reminds me of API discussions we had for PHP4 years back and just makes me sad. Expect weaponized exploits to full-disclosure lists in the future, your bug-tracker is obviously useless.

So long. And thanks for all the fishy crypto.

#### #25 - 05/03/2016 10:19 PM - sarciszewski (Scott Arciszewski)

Let's look at some OpenSSL RNG failures outside the Ruby microcosm:

- https://emboss.github.io/blog/2013/08/21/openssl-prng-is-not-really-fork-safe/
- https://github.com/ramsey/uuid/issues/80
- https://eprint.iacr.org/2016/367

If you're still dead set on using OpenSSL for a RNG, just know that when people say things like https://twitter.com/raptortech97/status/727345563440656385 about PHP being insecure, it applies tenfold to Ruby because of everyone's stubborn refusal to fix their RNG.

To wit (here's a CSPRNG done right): https://github.com/php/php-src/blob/master/ext/standard/random.c

#### #26 - 05/04/2016 12:41 AM - normalperson (Eric Wong)

Has anybody here brought up the issue to the OpenSSL team to get OpenSSL fixed?

Fixing OpenSSL would benefit far more people than just working around the problem in Ruby.

I'm definitely no expert on RNGs, but when I encounter bugs in other software; I try to get it fixed at the source rather than working problems at a higher level.

#### #27 - 05/04/2016 12:53 AM - Icey (Chris MacNaughton)

Eric Wong wrote:

> Has anybody here brought up the issue to the OpenSSL team to get OpenSSL fixed?
>
> Fixing OpenSSL would benefit far more people than just working around the problem in Ruby.
>
> I'm definitely no expert on RNGs, but when I encounter bugs in other software; I try to get it fixed at the source rather than working problems at a higher level.

Yes: https://github.com/openssl/openssl/issues/898

#### #28 - 05/04/2016 01:44 AM - duerst (Martin Dürst)

Chris MacNaughton wrote:

> Eric Wong wrote:
>
> > Has anybody here brought up the issue to the OpenSSL team to get OpenSSL fixed?
>
> Yes: https://github.com/openssl/openssl/issues/898

And on that bug, the speed issue was mentioned, too, like at https://bugs.ruby-lang.org/issues/9569#note-21.

So I see several ways forward for this:

1) Get OpenSSL to fix it. It would definitely benefit more people than just Ruby users.

2) Convince Ruby committers that urandom can be used without speed problems on *all* relevant platforms and for *all* relevant application.

#### #29 - 05/04/2016 02:07 AM - delan (Delan Azabani)

For anyone reading this thread after me:

- SecureRandom.gen_random calls OpenSSL::Random.random_bytes before falling back on Random.raw_seed [1]
- Random.raw_seed calls fill_random_bytes [2]
- fill_random_bytes calls fill_random_bytes_syscall before falling back on fill_random_bytes_urandom [3]
- fill_random_bytes_syscall is compiled to wrap CryptGenRandom if available, falling back on getrandom(2) [4]
- fill_random_bytes_urandom reads from /dev/urandom if it considers the device reasonable [5]

It's good to see that fill_random_bytes already prefers getrandom(2) over /dev/urandom, as getrandom(2) blocks only if it has been called so early in the boot process that there isn't enough *initial* entropy [6], and it isn't susceptible to file descriptor exhaustion attacks [7].

Changing the order of SecureRandom.gen_random should be enough to fix this bug, but I would also suggest adding arc4random(3) [8] or getentropy(2) [9] to the collection of methods tried by fill_random_bytes_syscall, which will mitigate file descriptor exhaustion attacks for OpenBSD users.

1. https://github.com/ruby/ruby/blob/8ef6dacb248876b444595a26ea78c35eb07a188b/lib/securerandom.rb#L50-78
2. https://github.com/ruby/ruby/blob/8ef6dacb248876b444595a26ea78c35eb07a188b/random.c#L620-635
3. https://github.com/ruby/ruby/blob/8ef6dacb248876b444595a26ea78c35eb07a188b/random.c#L549-555
4. https://github.com/ruby/ruby/blob/8ef6dacb248876b444595a26ea78c35eb07a188b/random.c#L493-547
5. https://github.com/ruby/ruby/blob/8ef6dacb248876b444595a26ea78c35eb07a188b/random.c#L444-481
6. http://www.2uo.de/myths-about-urandom/
7. https://lwn.net/Articles/606141/
8. http://man.openbsd.org/OpenBSD-current/man3/arc4random.3
9. http://man.openbsd.org/OpenBSD-current/man2/getentropy.2

#### #30 - 05/04/2016 02:21 AM - jeremyevans0 (Jeremy Evans)

Delan Azabani wrote:

> Changing the order of SecureRandom.gen_random should be enough to fix this bug, but I would also suggest adding arc4random(3) or getentropy(2) to the collection of system calls tried by fill_random_bytes_syscall, which will mitigate file descriptor exhaustion attacks for OpenBSD users.

Nothing needs to be done on OpenBSD, since SecureRandom.random_bytes uses OpenSSL::Random.random_bytes, which calls RAND_bytes(3). OpenBSD uses LibreSSL, which on OpenBSD has RAND_bytes(3) call arc4random_buf(3) which calls getentropy(2). OpenBSD's libssl already mitigates file descriptor exhaustion attacks.

#### #31 - 05/04/2016 01:32 PM - kosaki (Motohiro KOSAKI)

> Delan Azabani wrote:
>
> > Changing the order of SecureRandom.gen_random should be enough to fix this bug, but I would also suggest adding arc4random(3) or getentropy(2) to the collection of system calls tried by fill_random_bytes_syscall, which will mitigate file descriptor exhaustion attacks for OpenBSD users.
>
> Nothing needs to be done on OpenBSD, since SecureRandom.random_bytes uses OpenSSL::Random.random_bytes, which calls RAND_bytes(3). OpenBSD uses LibreSSL, which on OpenBSD has RAND_bytes(3) call arc4random_buf(3) which calls getentropy(2). OpenBSD's libssl already mitigates file descriptor exhaustion attacks.

Great.

This is what application developers want to (Open|Libre)SSL exactly.

#### #32 - 05/04/2016 03:35 PM - akr (Akira Tanaka)

I think switching OpenSSL to getrandom() is possible if the performance is acceptable.
The man page for getrandom(),
http://man7.org/linux/man-pages/man2/getrandom.2.html
doesn't discourage non-seed cryptgraphic purposes.

OpenBSD's getentropy() has limitation on buffer size: 256 bytes.
http://man.openbsd.org/OpenBSD-current/man2/getentropy.2

Some gems use SecureRandom.random_bytes(more-than-256).
So, making SecureRandom.random_bytes as single getentropy() invocation
causes incompatibility.

arc4random may be possible.
However, I'm not sure that arc4random is preferable over OpenSSL.

#### #33 - 05/06/2016 02:55 AM - bascule (Tony Arcieri)

In the meantime until upstream changes are made to SecureRandom, if anyone has come across this thread and is looking for a RubyGem that provides a /dev/urandom-based RNG (and a Windows equivalent), RbNaCl's RandomBytes module wraps libsodium's randombytes.c RNG implementation which uses this strategy:

https://github.com/cryptosphere/rbnacl/wiki/Random-Number-Generation

#### #34 - 05/07/2016 02:42 AM - daira (Daira Hopwood)

Many competent cryptographers and engineers have told you that using the OpenSSL RNG is significantly less safe than directly using /dev/urandom. They are right. Even better would be to use libsodium's RNG API, which handles portability issues well and will use getrandom where available.

OpenSSL's RNG has several serious design problems, and I would not be at all surprised if more were found. Please listen to what people with experience in security engineering are telling you about this. This RNG and API is very poorly designed: consider for example the mistake of attempting to using uninitialized memory as an entropy source, which hides failures without adding any reliable entropy. The paper referenced in Aaron Zauner's comment - https://eprint.iacr.org/2016/367.pdf - describes how this can potentially leak secrets. To my mind this problem on its own would be sufficient reason to stop using this RNG.

#### #35 - 05/08/2016 05:52 AM - shyouhei (Shyouhei Urabe)

Daira Hopwood wrote:

> Many competent cryptographers and engineers have told you that using the OpenSSL RNG is significantly less safe than directly using /dev/urandom. They are right. Even better would be to use libsodium's RNG API, which handles portability issues well and will use getrandom where available.
>
> OpenSSL's RNG has several serious design problems, and I would not be at all surprised if more were found. Please listen to what people with experience in security engineering are telling you about this. This RNG and API is very poorly designed: consider for example the mistake of attempting to using uninitialized memory as an entropy source, which hides failures without adding any reliable entropy. The paper referenced in Aaron Zauner's comment - https://eprint.iacr.org/2016/367.pdf - describes how this can potentially leak secrets. To my mind this problem on its own would be sufficient reason to stop using this RNG.

I'm not going against the point you said about OpenSSL's RNG, but in the current implementation of SecureRandom, we do add some entropy to it before using.  So the problem (might be real but) does not affect us I believe.

#### #36 - 05/09/2016 04:57 AM - kernigh (George Koehler)

I made myself a benchmark:
https://gist.github.com/kernigh/d169895a700c6511d08511c005a28d88

RAND_bytes() from OpenSSL seems to be faster than /dev/urandom on my computer. I'm running OpenBSD, and OpenSSL is really LibreSSL. At first it seems that /dev/urandom is just as fast as RAND_bytes():

```
$ ruby -v
ruby 2.4.0dev (2016-05-05 trunk 54913) [x86_64-openbsd5.9]
$ ruby benchrand.rb
                           user     system      total        real
Random::DEFAULT        0.410000   0.190000   0.600000 (  0.597455)
/dev/urandom           0.000000   1.340000   1.340000 (  1.395878)
arc4random_buf         1.210000   0.160000   1.370000 (  1.373183)
RAND_bytes             1.160000   0.260000   1.420000 (  1.508660)
gnutls_rnd NONCE       0.810000   0.180000   0.990000 (  0.988610)
gnutls_rnd RANDOM      1.940000   0.210000   2.150000 (  2.147149)
gnutls_rnd KEY         2.010000   0.140000   2.150000 (  2.147888)
PK11_GenerateRandom    4.030000   0.230000   4.260000 (  4.266837)
```

But if I run two instances in parallel on my dual-core machine:

```
$ for i in 1 2; do xterm -hold -e ruby benchrand.rb& done
```

then /dev/urandom is suddenly much slower:

```
                           user     system      total        real
Random::DEFAULT        0.450000   0.250000   0.700000 (  0.706498)
/dev/urandom           0.000000   2.470000   2.470000 (  2.549602)
arc4random_buf         1.260000   0.530000   1.790000 (  1.838425)
```

```
RAND_bytes              1.210000   0.470000   1.680000 (  1.795881)
gnutls_rnd NONCE        0.840000   0.450000   1.290000 (  1.335806)
gnutls_rnd RANDOM       1.960000   0.430000   2.390000 (  2.440970)
gnutls_rnd KEY          1.960000   0.440000   2.400000 (  2.434435)
PK11_GenerateRandom     4.070000   0.410000   4.480000 (  4.535968)
```

Seems that /dev/urandom on OpenBSD doesn't scale to multiple cores. So switching from RAND_bytes() to /dev/urandom would be bad idea. The other generators scale better: they are arc4random_buf() from libc, RAND_bytes() from LibreSSL, gnutls_rnd() from GnuTLS, and PK11_GenerateRandom() from NSS. In OpenBSD, RAND_bytes() seems to call arc4random_buf(), so their times are similar. I know that RAND_bytes() is different between OpenSSL and LibreSSL, and arc4random_buf() is different in other BSDs.

My benchmark is for 256 MB of random data. Real programs might not need so much randomness. PK11_GenerateRandom() imposes a limit of 65536 bytes per call, but this seems enough for NSS users like Firefox. So my benchmark might not be realistic.

Ruby has been avoiding /dev/urandom if possible. Ruby trunk works as Delan Azabani wrote in #9569-29, with one addition: as of a few days ago, fill_random_bytes_syscall() now tries arc4random_buf() first. So SecureRandom tries in order RAND_bytes(), arc4random_buf(), CryptGenRandom(), getrandom() before /dev/urandom.

#### #37 - 05/11/2016 05:55 AM - bascule (Tony Arcieri)

George Koehler: you're missing the point: the OS RNG should be used instead of OpenSSL's. That isn't necessarily /dev/urandom on every platform. Clearly /dev/urandom is a poor candidate for Windows (CryptGenRandom or RtlGenRandom should be used)

Again, I'll point to libsodium's randombytes implementation (and take the opportunity to say it's available today via the RbNaCl gem).

https://github.com/jedisct1/libsodium/blob/master/src/libsodium/randombytes/sysrandom/randombytes_sysrandom.c

This implementation uses arc4random() on OpenBSD:

https://github.com/jedisct1/libsodium/blob/master/src/libsodium/randombytes/sysrandom/randombytes_sysrandom.c#L36

Specifically it checks for a "safe" arc4random(). On OpenBSD, despite the name arc4random() doesn't use RC4 internally, but instead uses the ChaCha20 stream cipher, which is modern, fast, and secure.

#### #38 - 05/11/2016 09:31 AM - naruse (Yui NARUSE)

*- Status changed from Rejected to Open*

In general SecureRandom should work as it works on OpenBSD.
It uses arc4random_buf, which gathers entropy from /dev/urandom.

A problem of this direction is the risk of the vulnerability of arc4random_buf.
Other than recent BSDs including Linux doesn't have arc4random_buf with ChaCha20.
It means Ruby needs to have the logic.
But if Ruby has, it requires security update if it happen to be found vulnerability.
To ease the such case I think SecureRandom should be separated into securerandom.gem and Ruby bundle it like Rake and RubyGems.

#### #39 - 05/12/2016 01:06 AM - shyouhei (Shyouhei Urabe)

naruse (Yui NARUSE) Do you think it's inadequate for Linux users to fall back to getrandom(2)?  If so, why?

#### #40 - 05/17/2016 06:37 AM - naruse (Yui NARUSE)

*- Assignee deleted (ruby-core)*

#### #41 - 05/24/2016 05:19 AM - naruse (Yui NARUSE)

Shyouhei Urabe wrote:

> naruse (Yui NARUSE) Do you think it's inadequate for Linux users to fall back to getrandom(2)?  If so, why?

getrandom has some limitations like its max output (33554431), and consumes entropy.

Anyway I'm creating a securerandom.gem which uses arc4random_buf internally like libressl RAND_bytes.
https://github.com/nurse/securerandom

#### #42 - 05/24/2016 06:43 AM - delan (Delan Azabani)

Yui NARUSE wrote:

> getrandom has some limitations like its max output (33554431), and consumes entropy.

This probably won't add much to the conversation, but getrandom(2) and /dev/random don't "consume" entropy any more than /dev/urandom does.

getrandom(2) can be told to read from either of the devices. /dev/random and /dev/urandom have separate entropy pools, but they use the same algorithm. The only difference between them is that /dev/urandom doesn't block when its "entropy estimate" is "low" — a misleading and simplistic mistake that we're now stuck with on Linux. The "estimate" would only matter slightly if you're in information theoretic territory and you also believe that SHA-1 or the Linux PRNG is broken.

If you don't agree, then using /dev/urandom (or anything that's based on it) *also* consumes entropy, but now you simply won't be told when it's too low.

#### #43 - 05/24/2016 07:46 AM - shyouhei (Shyouhei Urabe)

Yui NARUSE wrote:

> Shyouhei Urabe wrote:
>
> > naruse (Yui NARUSE) Do you think it's inadequate for Linux users to fall back to getrandom(2)? If so, why?
>
> getrandom has some limitations like its max output (33554431), and consumes entropy.

According to getrandom(2) manpage, that 33554431-bytes limitation only applies to 32bit environments. Why not repeatedly call it again and again to fulfill the needs? Given 32bit, repetition cannot go any further than 128 times.

My private feeling is it's even worse than current situation to copy & paste arc4random source code. I don't pretend everything requested in this thread is totally bad. I especially support the motivation to write less code to maximize security. Your securerandom.gem seems like replacing one nightmare with another.

#### #44 - 05/24/2016 04:24 PM - naruse (Yui NARUSE)

Shyouhei Urabe wrote:

> My private feeling is it's even worse than current situation to copy & paste arc4random source code. I don't pretend everything requested in this thread is totally bad. I especially support the motivation to write less code to maximize security. Your securerandom.gem seems like replacing one nightmare with another.

https://github.com/nurse/securerandom/blob/master/Rakefile#L41-63 can be an answer for you, though you wouldn't agree.

#### #45 - 05/24/2016 10:23 PM - mwpastore (Mike Pastore)

Yui NARUSE wrote:

> Anyway I'm creating a securerandom.gem which uses arc4random_buf internally like libressl RAND_bytes.
> https://github.com/nurse/securerandom

You might want to take a look at (shameless self-plug, sorry) https://github.com/mwpastore/securer_randomer for a comparable solution.

#### #46 - 05/25/2016 05:12 AM - naruse (Yui NARUSE)

Mike Pastore wrote:

> Yui NARUSE wrote:
>
> > Anyway I'm creating a securerandom.gem which uses arc4random_buf internally like libressl RAND_bytes.
> > https://github.com/nurse/securerandom
>
> You might want to take a look at (shameless self-plug, sorry) https://github.com/mwpastore/securer_randomer for a comparable solution.

You mean utility methods which are implemented in Random::Formatter on Ruby 2.4?
They are out of scope now, though I may merge the module from Ruby repo.

#### #47 - 05/25/2016 06:13 AM - mwpastore (Mike Pastore)

Yui NARUSE wrote:

> Mike Pastore wrote:
>
> > Yui NARUSE wrote:
> >
> > > Anyway I'm creating a securerandom.gem which uses arc4random_buf internally like libressl RAND_bytes.
> > > https://github.com/nurse/securerandom

> > You might want to take a look at (shameless self-plug, sorry) https://github.com/mwpastore/securer_randomer for a comparable solution.
>
> You mean utility methods which are implemented in Random::Formatter on Ruby 2.4?
> They are out of scope now, though I may merge the module from Ruby repo.

What? No, it uses RbNaCl.random_bytes instead of OpenSSL's RAND_bytes.

#### #48 - 05/26/2016 04:49 AM - naruse (Yui NARUSE)

Mike Pastore wrote:

> Yui NARUSE wrote:
>
> > Mike Pastore wrote:
> >
> > > Yui NARUSE wrote:
> > >
> > > > Anyway I'm creating a securerandom.gem which uses arc4random_buf internally like libressl RAND_bytes.
> > > > https://github.com/nurse/securerandom
> > >
> > > You might want to take a look at (shameless self-plug, sorry) https://github.com/mwpastore/securer_randomer for a comparable solution.
> >
> > You mean utility methods which are implemented in Random::Formatter on Ruby 2.4?
> > They are out of scope now, though I may merge the module from Ruby repo.
>
> What? No, it uses RbNaCl.random_bytes instead of OpenSSL's RAND_bytes.

I know RbNaCl uses libsodium and it is explained above.

#### #49 - 05/29/2016 12:19 AM - bascule (Tony Arcieri)

I've created a new gem which packages the libsodium's "randombytes_sysrandom" as a self-contained RubyGem. It's called "sysrandom":

https://github.com/cryptosphere/sysrandom

Ideally I'd like to see SecureRandom adopt this same sort of approach, but in the meantime this supports patching "Sysrandom" in for "SecureRandom" so you can take advantage of OS-level RNG anywhere SecureRandom is being used.

#### #50 - 05/30/2016 08:12 PM - sarciszewski (Scott Arciszewski)

I'm now actively discouraging anyone from ever using the incorrectly-named SecureRandom, in favor of Tony Arcieri's Sysrandom (which can be monkey-patched in place of SecureRandom).

https://paragonie.com/blog/2016/05/how-generate-secure-random-numbers-in-various-programming-languages#ruby-csprng

I move, instead, to rename this SecureRandom to InsecureRandom, since it still relies on OpenSSL.

#### #51 - 05/31/2016 12:37 AM - shyouhei (Shyouhei Urabe)

Thank you for your interesting idea of OpenSSL being insecure.  If you really believe so (and think you are a security expert), what you should do is actually fixing OpenSSL, not ranting here.

Why I think it's OK to wait for upstream fix (be they Linux manpage or OpenSSL or whatever) is that the problem people argue is theoretical at worst. I don't think there is actual flaw yet.  That doesn't mean there will never be, but I see no reason to urgent-fix this.  If there is a real threat today, the picture drastically changes.  Is there?

EDIT: To be precise I agree OpenSSL is not the only solution today, and I can understand people's motivation to want if off, but being insecure or not is totally different story than that.

#### #52 - 06/15/2016 12:03 PM - azet (Aaron Zauner)

Yui NARUSE wrote:

> Shyouhei Urabe wrote:
>
> > naruse (Yui NARUSE) Do you think it's inadequate for Linux users to fall back to getrandom(2)?  If so, why?

> getrandom has some limitations like its max output (33554431), and consumes entropy.
>
> Anyway I'm creating a securerandom.gem which uses arc4random_buf internally like libressl RAND_bytes.
> https://github.com/nurse/securerandom

This uses CryptGenRandom on Windows (as does OpenSSL currently).

```
Historically, we always told developers not to use functions such as rand to generate keys, nonces and passwor
ds, rather they should use functions like CryptGenRandom, which creates cryptographically secure random number
s. The problem with CryptGenRandom is you need to pull in CryptoAPI (CryptAcquireContext and such) which is fi
ne if you’re using other crypto functions.

On a default Windows XP and later install, CryptGenRandom calls into a function named ADVAPI32!RtlGenRandom, w
hich does not require you load all the CryptAPI stuff. In fact, the new Whidbey CRT function, rand_s calls Rtl
GenRandom.

The following snippet shows how to call the function.

HMODULE hLib=LoadLibrary(“ADVAPI32.DLL”);
if (hLib) {
 BOOLEAN (APIENTRY *pfn)(void*, ULONG) =
      (BOOLEAN (APIENTRY *)(void*,ULONG))GetProcAddress(hLib,”SystemFunction036”);
 if (pfn) {
  char buff[32];
  ULONG ulCbBuff = sizeof(buff);
  if(pfn(buff,ulCbBuff)) {

   // use buff full of random goop

  }
 }

 FreeLibrary(hLib);
}

The good news is you can get good random numbers, without the memory overhead of pulling in all of CryptoAPI!

RtlGenRandom is documented at http://msdn.microsoft.com/library/en-us/seccrypto/security/rtlgenrandom.asp.
```

https://blogs.msdn.microsoft.com/michael_howard/2005/01/14/cryptographically-secure-random-number-on-windows-without-using-cryptoapi/

If you use API's like the one provided by libsodium/NaCl this will be handled for you automatically. If things change in the OS, the library maintainers make sure to use the latest secure OS mechanisms and provide support for legacy systems.

Shyouhei Urabe wrote:

> Thank you for your interesting idea of OpenSSL being insecure.  If you really believe so (and think you are a security expert), what you should do is actually fixing OpenSSL, not ranting here.

FYI; this is exactly what Scott did back in March: https://github.com/openssl/openssl/issues/898 and they're working on it. If you currently use OpenSSL with an old release (for instance if you run a legacy RedHat or CentOS  system because an application or specific vendor requirements [e.g. prominent closed-source databases] make you depend on it) and you want to use Ruby on that system as well, SecureRandom is -- as Scott points out -- InsecureRandom.

> EDIT: To be precise I agree OpenSSL is not the only solution today, and I can understand people's motivation to want if off, but being insecure or not is totally different story than that.

I've audited Tony's Sysrandom and it's a sound solution and wrapper for SecureRandom. I urge the Ruby team to at least look into it.

A few years back Ruby was exploitable exactly because of the OpenSSL RNG, remember the fix [0] from Martin Bosslet back then? He wrote a nice blog post about the whole ordeal: https://emboss.github.io/blog/2013/08/21/openssl-prng-is-not-really-fork-safe/

[0] - https://github.com/ruby/ruby/blob/4c661094c9d2c6800a7f43f41b812fa4aee18634/lib/securerandom.rb#L53-L63

#### #53 - 06/16/2016 02:03 AM - shyouhei (Shyouhei Urabe)

Aaron Zauner wrote:

> FYI; this is exactly what Scott did back in March: https://github.com/openssl/openssl/issues/898 and they're working on it.

I wasn't aware that Scott is the one who opened that ticket.  Thank you for pointing this out.

So he is (at least trying to) make the world better.  I'd like to appreciate him about it.

Now, there are several choices proposed:

- Use OS-provided random device (OP's choice)
- Use OpenSSL (current choice)
- Use libsodium (Tony's choice)
- Make a tailored library to provide arc4random (Yui's choice)

Everyone advocates their advantages.  Maybe "OpenSSL should die" can be the only thing everyone agree?  Several also seem to agree that Linux kernel devs are toxic, but then I don't understand why they think it's OK to continue using it (but not OpenSSL).  It would make sense if people propose Ruby to drop Linux support and move to OpenBSD but ...

#### #54 - 06/16/2016 06:55 AM - azet (Aaron Zauner)

Shyouhei Urabe wrote:

> Everyone advocates their advantages.  Maybe "OpenSSL should die" can be the only thing everyone agree?  Several also seem to agree that Linux kernel devs are toxic, but then I don't understand why they think it's OK to continue using it (but not OpenSSL).  It would make sense if people propose Ruby to drop Linux support and move to OpenBSD but ...

The Linux Kernel crypto subsystem maintainers have acknowledged that many user-land applications get this wrong and/or require performance from the urandom char device (having read a bit on the history and discussion on the random char devices; it seems they assumed user-land programmers will be competent enough to get this right by themselves years back, but not everybody is a Kernel developer). They're currently working on various improvements. Ted Ts'o recently committed code into urandom that bascially behaves like arc4random on current OpenBSD systems: It'll seed ChaCha20. He also added backtracking protection. There're a lot of other things being worked on, Stephan Mueller is implementing a generic interface for pluggable DRBGs for the kernel (e.g. AES-CTR-DRBG with AESNI support, hence, fast performance). Mind that this all happened in the past couple of weeks/months and is far from being available in any current Linux distribution.

See:
https://marc.info/?l=linux-crypto-vger&m=146217043829396&w=2
https://marc.info/?l=linux-crypto-vger&m=146588259306658&w=2

That being said: for a language this isn't a universal solution as you have to support multiple Operating Systems. Libraries like libsodium (as mentioned earlier) take care of that and are well maintained and regularly updated. Sysrandom seems like a solid solution unless Ruby wants to switch to libsodium proper. I think most security/crypto engineers in this thread will acknowledge that this is the right approach. OpenSSL is also moving into a similar direction (see previously mentioned issue).

#### #55 - 06/16/2016 04:10 PM - bascule (Tony Arcieri)

Note that libsodium's randombytes module just provides a cross-platform wrapper for OS RNGs, so really these two are the same:

- Use OS-provided random device (OP's choice)
- Use libsodium (Tony's choice)

#### #56 - 01/06/2017 01:47 PM - bdewater (Bart de Water)

Akira Tanaka wrote:

> Please update the man page first, if it is really out-dated.

This has happened with the Linux 4.09 release according to https://bugzilla.kernel.org/show_bug.cgi?id=71211. The random(4) man page at http://man7.org/linux/man-pages/man4/random.4.html now reads:

> The /dev/random interface is considered a legacy interface, and /dev/urandom is preferred and sufficient in all use cases, with the exception of applications which require randomness during early boot time; for these applications, getrandom(2) must be used instead, because it will block until the entropy pool is initialized.

#### #57 - 01/09/2017 08:07 AM - shyouhei (Shyouhei Urabe)

Bart de Water wrote:

> Akira Tanaka wrote:
>
> > Please update the man page first, if it is really out-dated.
>
> This has happened with the Linux 4.09

Very good news.  Thank you for letting us know this.

Now I think we have no reason to avoid /dev/urandom.  It doesn't immediately mean OpenSSL became bad but I understand the OP's motivation to "skip the middleman".  Because we already use getrandom(2) when available (see also Delan's comment #29), I think it's OK to simply skip OpenSSL and let securerandom try the system call first.

#### #58 - 01/19/2017 10:02 AM - akr (Akira Tanaka)

Bart de Water wrote:

> Akira Tanaka wrote:
>
> > Please update the man page first, if it is really out-dated.
>
> This has happened with the Linux 4.09 release according to https://bugzilla.kernel.org/show_bug.cgi?id=71211. The random(4) man page at http://man7.org/linux/man-pages/man4/random.4.html now reads:

Great.

We discussed this issue today.

Our plan is :

1. Rename Random.raw_seed to Random.urandom to make sure that it is usable for non-seed purpose
2. SecureRandom use Random.urandom.

#### #59 - 01/20/2017 08:00 AM - shyouhei (Shyouhei Urabe)

*- Status changed from Open to Closed*

Applied in changeset r57384.

---

SecureRandom should try /dev/urandom first [Bug #9569]

```
* random.c (InitVM_Random): rename Random.raw_seed to
  Random.urandom.  A quick search seems there are no practical use
  of this method than securerandom.rb so I think it's OK to rename
  but if there are users of it, this hunk is subject to revert.

* test/ruby/test_rand.rb (TestRand#test_urandom): test for it.

* lib/securerandom.rb (SecureRandom.gen_random): Prefer OS-
  provided CSPRNG if available. Otherwise falls back to OpenSSL.
  Current preference is:

  1. CSPRNG routine that the OS has; one of
     - getrandom(2),
     - arc4random(3), or
     - CryptGenRandom()
  2. /dev/urandom device
  3. OpenSSL's RAND_bytes(3)

  If none of above random number generators are available, you
  cannot use this module.  An exception is raised that case.
```

#### #60 - 09/10/2017 12:43 PM - shyouhei (Shyouhei Urabe)

*- Related to Bug #13885: Random.urandom  securerandom  added*

#### #61 - 04/27/2018 06:27 AM - nobu (Nobuyoshi Nakada)

*- Related to Bug #14716: SecureRandom throwing an error in Ruby 2.5.1 added*

#### #62 - 08/29/2018 03:17 AM - shyouhei (Shyouhei Urabe)

*- Related to Bug #15039: Random.urandom and SecureRandom arc4random use added*

#### #63 - 11/26/2020 05:20 AM - naruse (Yui NARUSE)

*- Related to Misc #17319: Rename Random.urandom to os_random and document random data sources added*