Because "use urandom" isn’t everything: a deep dive into CSPRNGs in Operating Systems & Programming Languages

Aaron Zauner
azet@azet.org

lambda.co.at:
Highly-Available, Scalable & Secure Distributed Systems

SHA2017 - 06/08/2017

Why do we need Random Numbers?

- randomize stuff in your operating system / language
- `man rand`
- Python: `os.urandom`
- TLS session cookies
- Key generation (e.g. RSA / Diffie-Hellman)
- TCP SYN cookies
- Bash: `${RANDOM}` :)

CSPRNG i

- "Cryptographically Secure Pseudo Random Number Generator"
- aka "RNG", "Random number generator"..
- Crypto nerds tend to call them "CSPRNGs" you may call them RNG or whatever, I don't care that much as long as it's secure!

CSPRNG ii

- Widely implemented in OS kernels
  - Linux: **`/dev/urandom`**
    1. manpage **`man random`** has been wrong for years
    2. many myths about kernel entropy
  - FreeBSD: **`/dev/*random`**
    1.
    2. Replace the RC4 algorithm for generating in-kernel secure random numbers with Chacha20. Keep the API, though, as that is what the other *BSD's have done. Use the boot-time entropy stash (if present) to bootstrap the in-kernel entropy source. (https://svnweb.freebsd.org/base?view=revision&revision=317015 - Sun Apr 16 09:11:02 2017 UTC)
  - Windows: **`RtlGenRandom()`**
- ..and in programming languages
  - (i.e. Python **`os.urandom`**, PHP **`rand()`**,..)
  - some had really bad bugs for a long time (i.e. debian predictable SSH keys: CVE-2008-0166)
  - many use the kernel provided CSPRNG, others use OpenSSL or custom RNGs - which is **BAD**!
  - OpenSSL provides a user-space RNG many link to or make use of (don't!)
  - Whoops: **`CVE-2017-11671: GCC generates incorrect code for RDRAND/RDSEED intrinsics`** (recent)

Some History

- the `/dev/random` and `/dev/urandom` devices used to be **really** old code (mid-90ties) originated from Ted Tso and a few others
- the manpage for them was **wrong** until fixed in late last december!
- you don't have to worry about kernel entropy - this is a **myth**!
- `HAVEGE` won't save you! it can make things worse (See: https://blog.cr.yp.to/20140205-entropy.html)

Old Linux Kernel implementation `0.x>4.x`

- mixing different pools of interrupts
- quite complicated to understand even for well versed C programmers
- it worked without larger incidents - probably pure luck and researchers unable to read char device code
- old design described well here:
  - Blog Post: https://pthree.org/2014/07/21/the-linux-random-number-generator/
  - Academic: https://eprint.iacr.org/2012/251.pdf

curiosities in (drivers/char/random.c)

Splevin ARX

- `fast_mix` implemented by George Splevin - never explained and no crypto experience - homebrewed ARX
- that code is still around - even in the upstream kernel git repo:
- Code here: https://tinyurl.com/fastmix

Current implementation i

- after long discussions and advice by crytographers the old design in **`random.c`** was changed in 4.2
- based on the old pools, AES-NI (if available - modern Intel/AMD CPUs have those), ChaCha20 XOR RdSEED (via Google's BoringSSL / Adam Langley - https://marc.info/?l=linux-crypto-vger&m=146584488030185&w=2)
- neat design, backtracking resistant, **pretty** fast, too:

```
azet@nd01 ~ % dd if=/dev/urandom of=/dev/null bs=1M count=1024
1024+0 records in
1024+0 records out
1073741824 bytes (1.1 GB) copied, 11.8289 s, 90.8 MB/s
```

Current implementation ii

- major work overhauling crypto-code in the kernel started with Linux 4.2
- Backtracking protection (https://marc.info/?l=linux-crypto-vger&m=146583297126471&w=2)
- Ted Tso (Jun 13, 2016): `With /dev/urandom we were always emitting more bytes than we had entropy available, because not blocking was considered more important.  Previously we were relying on the security of SHA-1.  With AES CTR-DRBG, you rely on the security with AES.` (https://marc.info/?l=linux-crypto-vger&m=146584488030185&w=2)
- Doesn't track entropy anymore because the "CRNG" (terminology,..) is faster (https://marc.info/?l=linux-crypto-vger&m=146458684806389&w=2)

Current implementation iii

- `random: replace urandom pool with a CRNG` (https://marc.info/?l=linux-crypto-vger&m=146217043829396&w=2)
- Nikos Mavrogiannopoulos (https://marc.info/?l=linux-crypto-vger&m=146229250001030&w=2):

```
I know, and I share this opinion. To their defense they will have to
provide a call which doesn,t make applications fail in the following
scenario:
1. crypto/ssl libraries are compiled to use getrandom() because it is
available in libc and and in kernel
2. everything works fine
3. the administrator downgrades the kernel to a version without
getrandom() because his network card works better with that version
4. Mayhem as applications fail
```

Current implementation iv

- `random: make /dev/urandom scalable for silly userspace programs` (https://marc.info/?l=linux-crypto-vger&m=146583311726544&w=2):

> On a system with a 4 socket (NUMA) system where a large number of application threads were all trying to read from /dev/urandom, this can result in the system spending 80% of its time contending on the global urandom spinlock. The application should have used its own PRNG, but let,s try to help it from running, lemming-like, straight over the locking cliff.

Current implementation v

- Myths and lies in `man 4 random` finally corrected: https://bugzilla.kernel.org/show_bug.cgi?id=71211&utm_content=buffer1d02b
  - this took years of convincing the original upstream authors etc.
  - had a huge impact on use of RNGs in programming languages etc.

Language issues: Ruby

- using OpenSSL RNG designed for fast TLS use, not general purpose
- multiple security engineers and cryptographers tried to convince them to switch to **`/dev/urandom`**
- took more than a year but finally they implemented a similar design to **`libsodium`** (I've made a T-Shirt!)
- **`SecureRandom without OpenSSL (or compatible alternatives) is nonsense.`**
- **`Please don't rude.`**
- Legendary bug: https://bugs.ruby-lang.org/issues/9569
- Tony Arcieri (@bascule) wrote a wrapper for the time being: https://github.com/cryptosphere/sysrandom

Language issues: Node.js

- similar story to Ruby
- lots of input from normal users (useless)
- https://github.com/nodejs/node/issues/5798 (endless thread)
- Latest comment: 'Note that OpenSSL has just landed a commit to use DRGB with AES-CTR of NIST SP 800-90A as openssl/openssl@75e2c87. We can use it with the os-specific seeding source (e.g. /dev/urandom) by a default define flag of OPENSSL_RAND_SEED_OS. I think it is best for us to wait for the next release of OpenSSL-1.1.1."

Language issues: Erlang

- same as Ruby and Node.js
- https://github.com/erlang/otp/blob/maint/lib/crypto/c_src/crypto.c

Python imrprovement

- warns if there’re insecure values: https://bugs.python.org/issue27292

OpenSSL

- Not thread safe - userspace - prone to bugs
- https://github.com/openssl/openssl/issues/898
- https://wiki.openssl.org/index.php/Random_Numbers
- Not even recommended by OpenSSL to use it as non-TLS CSPRNG

‘ HAVEGE

- dangerous to use!
- not maintained in more than 10yrs
- no current contacts / security audits except by the original authors
- doesn’t improve security!

Parting words

- . . .

Thanks for your patience. Are there any questions?

Twitter:

`@a_z_e_t`

E-Mail:

`azet@azet.org`

XMPP:

`azet@jabber.ccc.de`

GitHub:

`https://github.com/azet`

GPG Fingerprint:

`7CB6 197E 385A 02DC 15D8 E223 E4DB 6492 FDB9 B5D5`