Quintessence Labs

SOLUTION GUIDE

Are you Entropy Starved?

QuintessenceLabs' qRand is a unique product which uses our quantum random numbers to feed the entropy pool of a Linux system. This capability is very important, especially for virtual machines. This solution guide explores some real-world examples of why this matters.

What Does this Mean and Why Does it Matter?

Random numbers are vital to every operating system. They are used everywhere. Even a simple task like starting a program uses random numbers (ASLR) and obviously every cryptographic operation (such as SSL/TLS encryption, SSH or PKI) relies fundamentally on random numbers.

As every program on a computer is completely deterministic, how can you generate random numbers? Well, that's where the entropy pool comes in. Every operating system (we'll focus on Linux here) maintains a pool of really high entropy random bits. This pool is fed from random sources like your mouse movement, and the kernel makes sure that these random bits are really random (entropy is high). The pool has space for 4096 random bits with the highest entropy possible.

Entropy matters a lot: if you were to generate a 2048 bit long private key from an entropy pool that only has 20 bits, your 2048 bit private key may be 2048 bits long, but would be "only as good" as a 20 bit key.

Coming back to the entropy pool in Linux... The kernel maintains that pool: and if you draw random bits from it, the entropy goes down, but if the operating system finds good random events (such as mouse movements,...) the pool gets filled up and the entropy level goes up.

You can see for yourself the entropy level in Linux with the command "`cat /proc/sys/kernel/random/entropy_avail`", which gives you a number (in bits) how much estimated entropy is in the pool.

When an application needs random numbers (such as for generating a private key, initiating an SSL/TLS connection,...) it has two options:

- It can read from /dev/random
  - Reading from /dev/random gives you directly data from the entropy pool, which is really high entropy random data, but the problem is that once this entropy pool is empty, there is nothing the system can give you anymore and therefore it will "block" and not return any data until it finds more data.
  - This means in practice that the application that needs random data will wait potentially forever and look like it hangs.

- It can read from /dev/urandom
  - Reading from /dev/urandom will never block as /dev/urandom will use a deterministic random number generator that is seeded from the entropy pool, but it doesn't need the entropy pool to produce then good random numbers afterwards.
  - Different Linux versions use different algorithms how to produce the random numbers.

So, applications that do not want to use this "simple" deterministic random number generator from the Linux kernel have to rely on /dev/random, but as you've seen above: once the entropy pool is empty there won't be any more random data, resulting in performance issues.

How qRand Helps

This is exactly the problem that QuintessenceLabs' qRand solves. qRand is a daemon for Linux systems that's configured to monitor requests for randomness. If entropy falls below a specified limit, qRand corrects this by delivering full-entropy random numbers directly from qStream, our quantum random number generator. This enables applications to generate and use high-quality cryptographic keys, for instance, with no changes needed to the application itself.

qStream is a quantum-powered module that uses quantum tunneling to sample the random movement of electrons across a diode, generating truly unpredictable strings of random numbers. Streaming at up to 1Gbit/s, it provides plenty of random for qRand to feed entropy-starved applications needing that randomness for encryption keys or indeed any other application.

Who Needs This?

As we've seen above, the entropy pool gets filled by the operating system from "random" sources such as mouse movements. When that runs out, it either blocks, or uses deterministic approaches to generate random bits, with lower entropy. There are many situations where the entropy is hard to be filled by the operating system. Especially on virtual machines, this is a big problem as you don't have a monitor or a keyboard or a mouse and no environmental interface to draw randomness from.

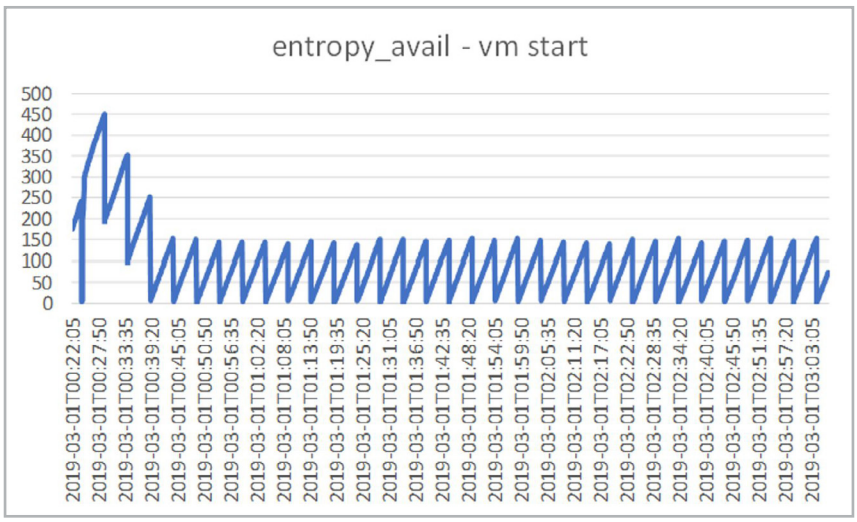
If you don't want to rely on these sources of randomness, and want to avoid the performance issues of blocking, you can now instead use entropy generated from a true quantum source. This is exactly what qRand provides.

The idea for qRand came while working with one of our partners, who had a large number of Linux systems — virtualized and otherwise — struggling to get entropy. Our team recognized the advantages of an entropy daemon for any organization with their own starvation problems, and out of that came today's qRand offering.

Practical Tests: Is This Problem Real?

Well, to answer that question, here is the outcome of a quick experiment. We fired up a virtual machine from Amazon EC2 and looked at what the entropy pool did.

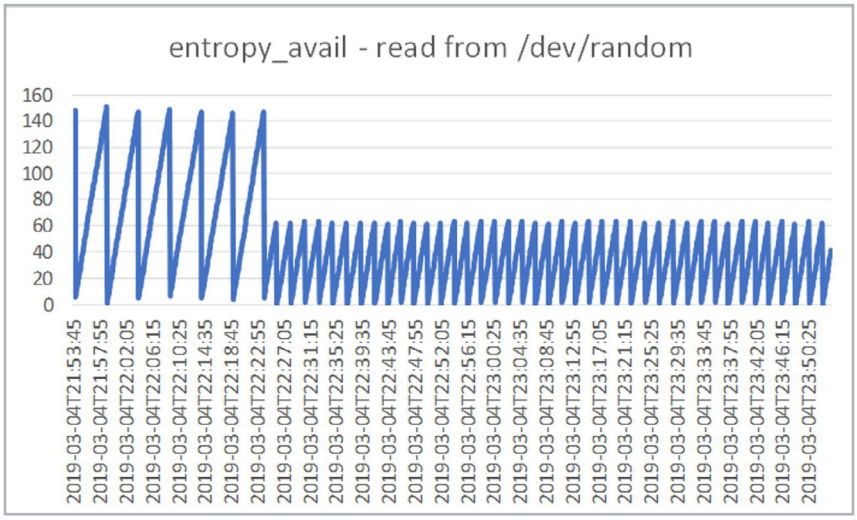
The example here is for an Ubuntu Linux instance T2.micro in us-west-2a, for which we obtained the following entropy level (measured in bits) profile over time:



As seen earlier, the entropy pool can hold 4096 random bits, and it is generally accepted that you are in trouble if entropy_avail values are below 300-400... In our experiment the entropy level on the virtual machine started around 200 and then constantly bounced between 0 and 150. Basically, nothing was happening on this system.

In spite of this, the system appeared to be running fine. As a next step, we tried to read some random bytes from /dev/random, but this immediately blocked and just kept running. You can see when we requested the random numbers below (when the entropy_avail dropped to between – and 60).

From this moment entropy_avail varied between 0 and 60, even though /dev/random didn't actually return any random bytes!



After almost 8h, we decided to put the system out of its misery and fired up qRand. My request for random bytes was then completed immediately.

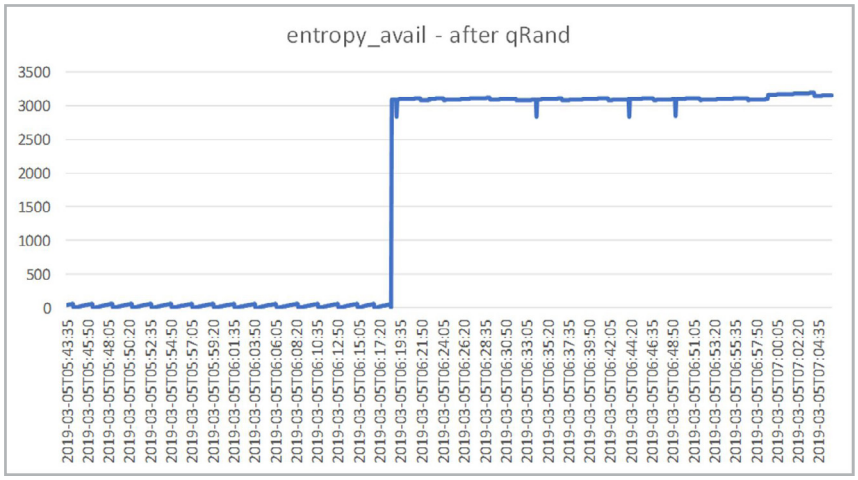
```
ubuntu@ip-172-31-41-147:~$ dd if=/dev/random of=rand bs=1K count=2 iflag=fullblock
2+0 records in
2+0 records out
2048 bytes (2.0 kB, 2.0 KiB) copied, 27620.7 s, 0.0 kB/s
```

As you can see, the full operation took almost 8h (27620 sec). But it only actually succeeded once we started qRand – and at that time it ran almost instantly.

We then re-did this experiment with qRand running from the start:

```
ubuntu@ip-172-31-41-147:~$ dd if=/dev/random of=rand bs=1K count=2 iflag=fullblock
2+0 records in
2+0 records out
2048 bytes (2.0 kB, 2.0 KiB) copied, 0.0317628 s, 64.5 kB/s
ubuntu@ip-172-31-41-147:~$
```

The operation was completed in 0.031 sec, and the chart below shows that the value of entropy_avail that the system reports as this is now nicely filled up and always stays above 3000 (this was the threshold we set for qRand).



So now we have a virtual machine that will behave beautifully and can always produce high quality high-entropy random numbers from /dev/random at any time. Problem solved – thanks to qRand.

qRand currently supports Ubuntu and RHEL Linux distributions, with support for more platforms planned.

Get in Touch

For more information on the qRand product and other cybersecurity solutions from QuintessenceLabs, visit **quintessencelabs.com** or contact **info@quintessencelabs.com**.

Quintessence Labs

AUSTRALIA
Unit 1, Lower Ground
15 Denison St
Deakin, ACT 2600
+61 2 6260 4922

UNITED STATES
175 Bernal Road
Suite 220
San Jose CA 95119
+1 650 870 9920

www.quintessencelabs.com

Document ID: 4234