Random Number Generators: what do you need one for?

Random Number Generators are a vital part of all modern operating systems. Being computers a deterministic system, random numbers are used where a non-deterministic input is required. An obvious use is virtual dice rolling for gambling systems, but if you don’t plan on running an online casino, what do you need one for?

Random numbers are utilized in many different areas, ranging from cryptography (in general) to source port and process ID randomization in some operating systems. Some of their uses in various Operating Systems are:

- Creation of keypairs (SSH, SSL, IPSec, etc.)
- Random DNS query ID’s;
- Random dynamic TCP/UDP source port allocation;
- Random dynamic Process ID allocation;
- Random salts for password algorithms;
- Etc.

So what is exactly a Random Number Generator?

Random number generators are just a source for random numbers. In their most basic form, they are functions that return a random number within a range.

There are two types of random number generators:

- True Random Number Generators (RNG),
- Pseudo Random Number Generators (PRNG).

True Random Number Generators (also known as non-deterministic RNG, or simply RNG) use a source of entropy to generate random numbers. Entropy is a measure of the “disorder” of a system or, in other words, its degree of randomness. As computer algorithms are always predictable, an external source must be used to introduce the entropy. Some of the most common sources of entropy are the keyboard and mouse inputs, the interrupts, the hard disk access latency, etc. The more random the source is, the larger the entropy, and the more unpredictable (random) the output will be.

Pseudo Random Number Generators (also known as deterministic RNG, as their next output can be theoretically determined) are algorithms that generate a large sequence of numbers, apparently random in nature, but that could be predicted. Normally, PRNG's are seeded before using them (some could be regularly re-seeded for additional disturbance). The seed sets the PRNG to a point in the sequence: *if your attacker knows the seed and the algorithm, he can predict the sequence of numbers that will be generated*. Without entering into excessive details, lets say that there are two families of PRNG's: linear and non-linear (the latter being the best).

Sources of entropy are usually scarce in a computer system, and the few ones available are typically not as good as you would expect. Most of the information obtained from the traditional sources of entropy must be discarded during the filtering, unbiasing and conditioning process (which may include hashing with a cryptographically strong algorithm), so true random numbers may not be readily available all the time.

One of the easiest way to test the amount of entropy and the randomness present in a byte stream, is to use the widely known Ent pseudo-random number sequence test program (another possibility is to use the diehard set of tests). Ent takes a file containing presumably random data and executes a battery of tests to determine how random that data is. Ent helps determine how good a Random Number Generator is, by performing several statistical tests that verify that the ouptput is unbiased and random in nature.

A sample output from Ent for a not so good set of random numbers would be:

```
Entropy = 7.683349 bits per byte.

Optimum compression would reduce the size
of this 12216 byte file by 3 percent.

Chi square distribution for 12216 samples is 25053.58, and randomly
would exceed this value 0.01 percent of the times.

Arithmetic mean value of data bytes is 116.7150 (127.5 = random).
Monte Carlo value for Pi is 3.282907662 (error 4.50 percent).
Serial correlation coefficient is - 0.008462 (totally uncorrelated = 0.0).
```

A better set of random numbers:

```
Entropy = 7.994824 bits per byte.

Optimum compression would reduce the size
of this 33984 byte file by 0 percent.

Chi square distribution for 33984 samples is 243.19, and randomly
would exceed this value 50.00 percent of the times.

Arithmetic mean value of data bytes is 126.5938 (127.5 = random).
Monte Carlo value for Pi is 3.180790960 (error 1.25 percent).
Serial correlation coefficient is 0.002392 (totally uncorrelated = 0.0).
```

Linux (and other Unix Operating Systems as well) offers two devices for random numbers:

- /dev/random: True RNG that collects entropy from the system and makes it available to applications. It can be depleted and, if that happens, it just blocks until more entropy is available;

- /dev/urandom: PRNG that utilizes entropy if available, but if there is not enough entropy it defaults to a traditional PRNG. You shouldn't rely on this device if you have strong cryptographic needs, but it may be the only one that can cope with the load.

So you will probably tend to use /dev/random for everything, but the keyword is that *the amount of entropy is limited*, so /dev/random, at best, can generate a few thousand random bytes per minute: not a whole lot if there are several applications consuming them. And remember that, as soon as the amount of entropy in /dev/random gets too low, the device *will block*, so applications using it will block as well (not good at all if one of the applications is waiting for the re-keying of an encrypted tunnel).

Of course, if you have an strong need for random numbers you can use additional entropy sources, or just a hardware based RNG. A true RNG is included with some chipsets (like the late Intel i810), CPU's (like the VIA Padlock enabled Nehemiah based C3 CPU's), and can also be added as an external piece of hardware (not cheap, but not too expensive either). Additional entropy sources that can be used to seed the internal PRNG are the soundcard (see Audio Entropy Daemon), a webcam (see Video Entropy Daemon), a modified camera (see LavaRnd), etc. Even SGI sponsored a project where Lava Lamps were used as entropy sources for random number generation, a few years ago.

Basically, any chaotic system is good as a source of entropy, either by itself or combined with others, as long as you make sure that it's not biased, and that it keeps producing random information consistently. Of course, there are some limitations. One of the main limitations is that you shouldn't use any system that could be externally controlled or biased: if your attacker knows how to influence the next output of your RNG, then there is little point to cryptography. So radio or tv tuners are not good sources of entropy; neither the network activity alone. Electric noise (if properly shielded) or quantum noise in semiconductors are good sources of entropy.

Many users may not need a Random Number Generator yet, but the the ever growing amount of cybercrime and the increasing demand for strong security, are making true RNG's a requirement for any cryptographically serious implementation.