Choudhury Noor

Cardy Wei

ECE 455 - Cyber Security

Professor Gitzel

Generation of Keys with High Entropy

Keys are an extremely fundamental and important part of cybersecurity. Their uses are endless, from creating ciphers that can be impossible to crack to securing even the largest of systems. However, with keys being such an integral portion of the cybersecurity industry, they need to be incredibly secure, and therefore need to be generated in a method that is infeasible or impossible to compute. Number generation in most cases is never fully random, existing in the state called pseudorandomness, meaning that while it does seem random according to statistical analysis, it has been created by a definite mathematical method, and therefore has a flaw. Generating pure randomness is extremely difficult to do so.

That is why many random number generators use methods such as measuring atmospheric conditions to simulate randomness. The Linux kernel provides a device interface to simulate randomness by using environmental variables from the system. We wish to be able to create a random key generator that contains enough entropy from system environment variables that it would be infeasible to crack. It measures user inputs, device IO timing and interrupt timings to generate cryptographically secure random numbers.

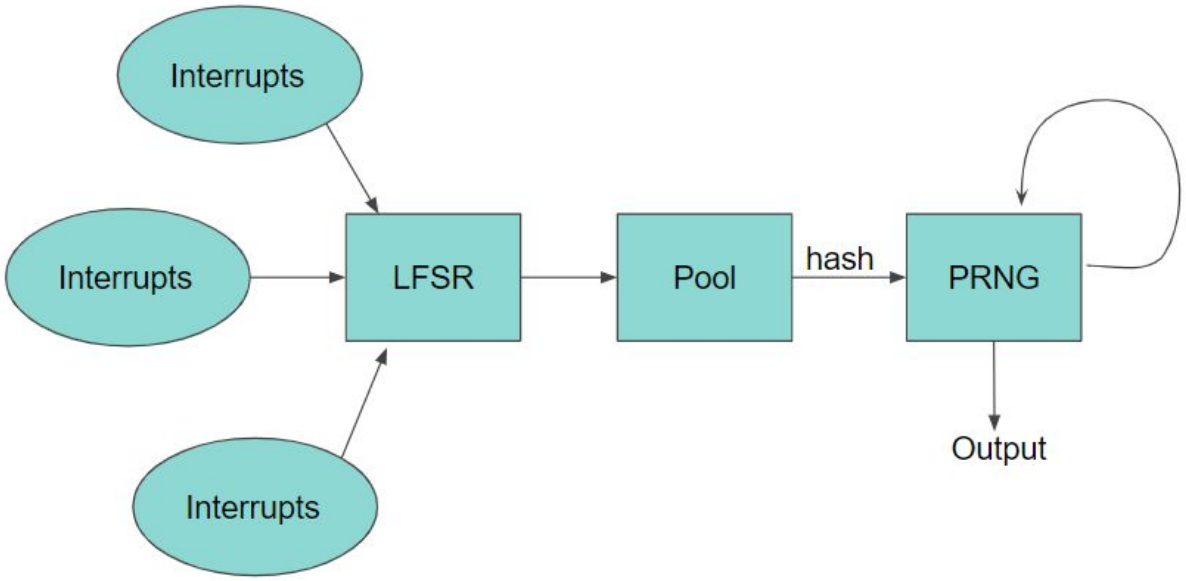
Recently, there have been doubts about the cryptographic security of Linux's RNG. Particularly, since VMs, SSDs and cloud based platforms lack any appreciable source of entropy

from the user inputs and device IO timing, it has been proposed that only using internal interrupt timing along with a deterministic, pseudo random number generator is more cryptographically secure. Furthermore, the entropy collected from user inputs and device IO timing is inconsistent and prone to attack. Another possible vulnerability in the existing implementation is that the entropy is added little by little, as it becomes available. This can allow an attacker to successfully decrease the entropy of the output by draining the entropy pool.

We aim to create our own Linux kernel module that acts as a random number generator. We generate the random numbers from using system interrupts only. Notice, however, that the timing information from user inputs and device IO is also incorporated into system interrupts since they rely on them for their operation. The only information lost is the information relayed through those interrupts. The hypothesis is that the lost information doesn't contribute much entropy to begin with.

Design

The interrupt are caught and their timings measured. Once multiple interrupts are caught, they are fed to a Linear Feedback Shift Register (LFSR), which is then put into the entropy pool. Since the module waits until sufficient number of interrupts are caught, the entropy pool receives and big chunk of entropy at a time. Thus, the attacker can not fully reduce the entropy of the pool by draining. A lot of people have done research on coming up with good tap values for LFSRs. We simply picked well known recommended taps. The pool is then hashed and used to seed a pseudo random number generator (PRNG), which eventually provides output to the user. Since the PRNG is continually reseeded by the LFSR, the outputs remain cryptographically secure.



Analysis

The output of our RNG and Linux's /dev/urandom was analyzed from an information theoretic point of view. The Shannon entropy of the generated outputs were calculated for different chunk sizes and compared to ensure that there isn't much loss of entropy. The Shannon entropy is as follows

Shannon's entropy equation:

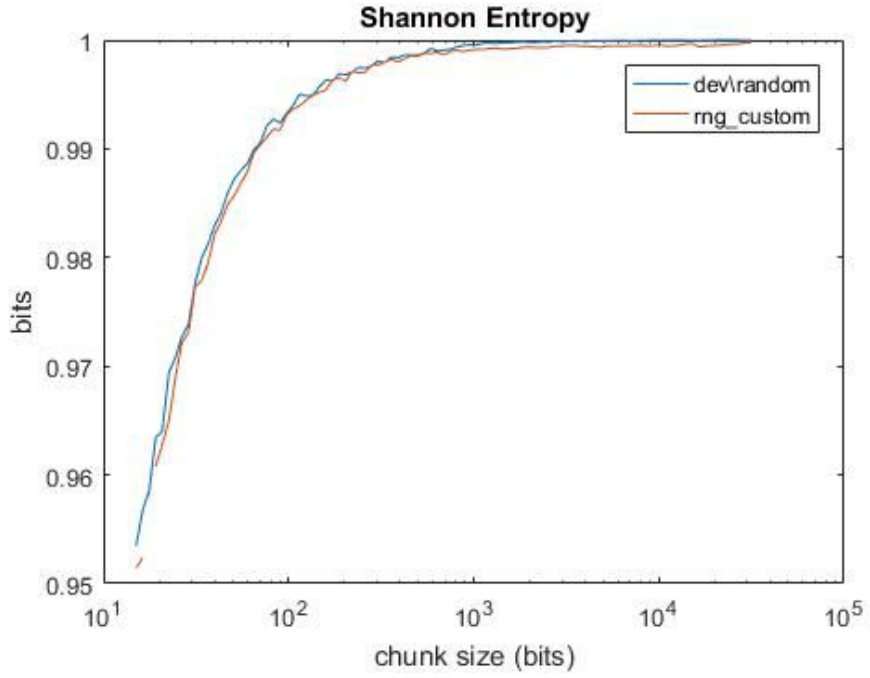
$$H(X) = -\sum_{i=0}^{N-1} p_i \log_2 p_i$$

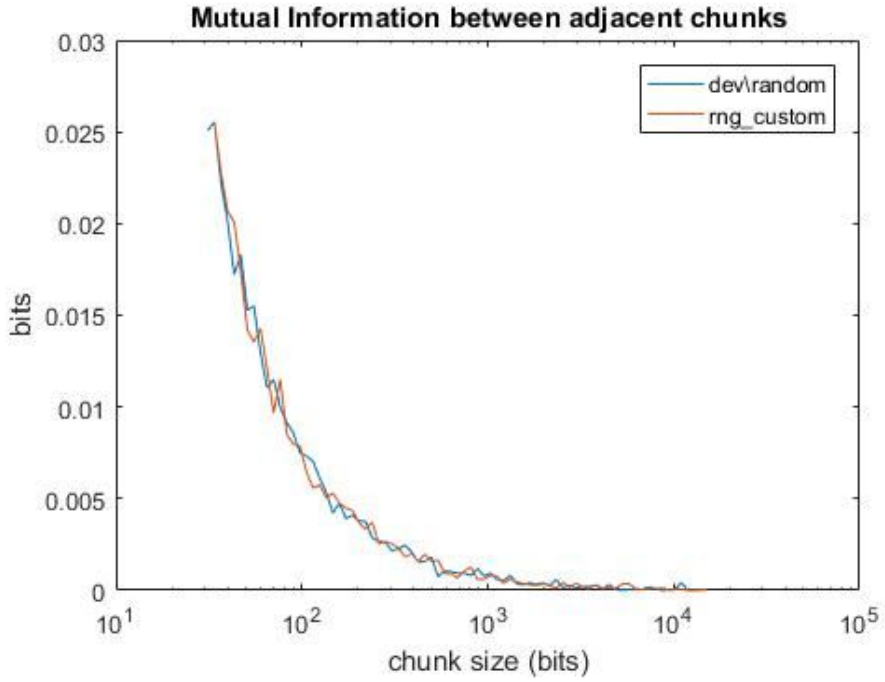
Where the probabilities are estimated by their relative frequencies.

We also looked at the mutual entropy between adjacent chunks for both our and Linux's implementation to measure the predictability of the data. The mutual entropy measures the amount of information gained about a chunk from the previous chunk. The formula is shown below.

$$\mathrm{I}(X;Y) \equiv \mathrm{H}(X) - \mathrm{H}(X|Y)$$

The results are shown below. As you can see, there is slight decrease in asymptotic entropy but no noticeable change in Mutual Information.





References

1. https://static.lwn.net/images/pdf/LDD3/ch11.pdf
2. http://chronox.de/lrng/doc/lrng.pdf
3. https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Studien/ZufallinVMS/Randomness-in-VMs.pdf?__blob=publicationFile&v=3
4. https://lkml.org/lkml/2017/7/18/213