Entropy Loss and Output Predictability in the Libgcrypt PRNG

CVE-2016-6313

Felix Dörre
Karlsruhe Institute of Technology, Germany
felix.doerre@student.kit.edu

Vladimir Klebanov
Karlsruhe Institute of Technology, Germany
klebanov@kit.edu

In the following we describe a design flaw in the mixing function of the Libgcrypt PRNG. Due to the flaw, mixing the full entropy pool reduces the stored entropy amount by at least 20 bytes. Furthermore, the flaw makes a part of the PRNG output completely predictable. This bug exists since 1998 in all GnuPG and Libgcrypt versions and is tracked as CVE-2016-6313. A release fixing the problem is available as of 2016-08-17.

We discovered the flaw with the help of the ENTROPOSCOPE tool developed by us and described in [1].

1. THE MIXING FUNCTION

The mixing function in question is `mix_pool` in `random-csprng.c`. It is supposed to perturb the content of the entropy pool while maintaining its entropy content. The latter requirement (freedom from entropy loss) means formally that the mixing function ought to be injective, i.e., transform distinct pools into distinct pools. For an in-depth discussion of this property, we refer to [1].

It remains to note that the length L of the entropy pool is $30 \cdot 20 = 600$ bytes in the default configuration. A comment in the source states that the Libgcrypt PRNG is modeled after a proposal by Gutmann [2].

Original Proposal by Gutmann.

Figure 1 shows the original proposal for a mixing function presented in [2]. The mixing function operates in cycles. The top rectangle represents the entropy pool at the beginning of the (first) cycle, while the bottom rectangle shows the pool at the end of the cycle. A 20-byte hash of a sliding window first covering bytes [0,84) in the pool is computed and the result is used to overwrite the bytes [20,40). After that, the sliding window is shifted right by 20 bytes and the next cycle commences. If a part of the sliding window extends beyond the end of the pool, it is wrapped around. The function terminates when all bytes in the pool have been rewritten (i.e., after 30 cycles).

Implementation in Libgcrypt.

There are two relevant differences between the proposal in [2] and the Libgcrypt implementation. The more general difference is that the Libgcrypt sliding window has a "hole" (Figure 2b). The hash here is computed from the bytes $[0, 20) \cup [40, 84)$. The bytes [20,40), shown as hatched in the figure, are no longer part of the hash input. The more particular difference is that the first cycle (Figure 2a) deviates from the other cycles (Figure 2b). Here, the hash is computed from the bytes $[L - 20, L) \cup [0, 44)$.

Later on, we show that the hole in the sliding window decreases the security of the PRNG.

Properties of the Libgcrypt Mixing Function.

PROPOSITION 1. *The pool is the only (effective) reservoir of entropy.*

We could identify only two potential threats to this claim in the code:

- There is a small auxiliary entropy buffer (physically trailing the pool) used in the hash calculation, but its content is completely overwritten by data from the pool at the beginning of each cycle.
- The hash context is reused between cycles and contains a 20-byte chaining buffer. Yet, we note that the buffer's content at the end of a cycle is identical with the output of the hash function. The hash function output from cycle i, in its turn, is stored in the pool and fed into the hash function in cycle $i + 1$, after the sliding window shifts right. The chaining buffer, thus, injects no additional entropy into each hash operation beyond what is already contained in the pool.

COROLLARY 2. *Each cycle i, thus, induces a mathematical function f_i transforming a pool into another pool. The whole mixing function is a composition of such cycle functions:*

$$\mathit{mix\_pool} = f_{30} \circ \ldots \circ f_1 \ .$$

2. ENTROPY LOSS

PROPOSITION 3. *Consider an entropy pool containing L bytes of data with an entropy of L bytes. After an application the Libgcrypt mixing function, the entropy content of the pool is at most $L - 20$ bytes.*

PROOF. It is clear that the mixing function can only be injective, if every cycle function f_i is injective.

Yet, for any $2 \leqslant i \leqslant 30$, the corresponding cycle function f_i as implemented in Libgcrypt (Figure 2b) is not injective. For example, any two pools differing (only) in the byte range [20,40) will produce the same pool after applying f_2. $\square$

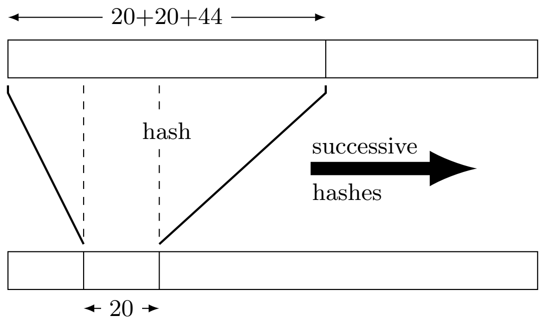
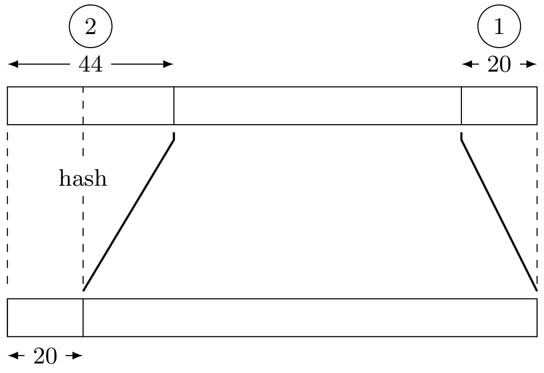
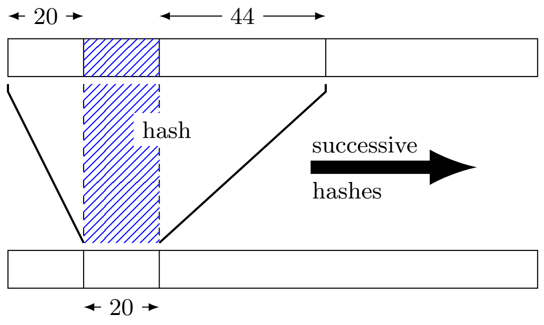


Figure 1: Mixing function proposed in [2]



(a) first cycle



(b) second and following cycles

Figure 2: Mixing function in Libgcrypt

3. OUTPUT PREDICTABILITY

In general, an entropy loss makes it easier for an attacker to brute-force the PRNG state. In this particular case, the entropy loss manifested itself also in trivial partial output predictability.

PROPOSITION 4. *When the Libgcrypt PRNG generates L bytes of output*[1]*, the last 20 bytes are trivially predictable from the first $L - 20$ bytes.*

PROOF. The PRNG produces output by deriving a so-called key pool of length L from the main entropy pool (in a way irrelevant here), mixing it, and returning its content to the client. Due to the design of the mixing function, the bytes $[L-20, L)$ of the key pool (and thus of the output) are obtained by hashing the bytes $[L-40, L-20) \cup [0, 44)$. The hash context chaining buffer at this point coincides with the bytes $[L-40, L-20)$, as explained above. We note that all of the bytes in that range are contained in the first $L - 20$ bytes of the key pool and thus output. $\square$

In other words, by taking the bytes $[L-40, L-20) \cup [0, 44)$ of the output, and hashing them with the hash context chaining buffer set to bytes $[L-40, L-20)$, an attacker can perfectly predict the bytes $[L-20, L)$ of the output. Simple proof-of-concept code confirms our finding.

4. CONCLUSION AND AFTERMATH

We have reported the bug to Werner Koch, the author and maintainer of Libgcrypt. We would like to thank Werner for a productive discussion. Bugfix releases are available as of 2016-08-17[2]. Each mixing cycle now hashes a contiguous 64-byte region of the pool, maintaining injectivity of the mixing function.

Please note that this document makes no claims about the effect of the flaw on the security of generated keys or other artifacts.

5. REFERENCES

[1] F. Dörre and V. Klebanov. Practical detection of entropy loss in pseudo-random number generators. In *Proceedings, CCS*, 2016. To appear.

[2] P. Gutmann. Software generation of practically strong random numbers. In *USENIX Security*, 1998.

[1]For simplicity, we assume that an L-multiple of bytes were generated previously.

[2]https://lists.gnupg.org/pipermail/gnupg-announce/2016q3/000395.html