16 November 2015

Introduction

On behalf of the German Federal Office for Information Security (BSI), Sirrix AG analyzed cryptographically relevant aspects of the OpenSSL library (version 1.0.1g). BSI and Sirrix AG would like to impart the most important findings from this analysis to the OpenSSL Software Foundation, so that the developers as well as the users of the OpenSSL library may benefit from the results of the study.

Sirrix AG complied a comprehensive report [2] in German language, but the most important findings are listed below in short form and in English language.

Results and Recommendations concerning the RNG

1. The functions RAND_write_file(), RAND_load_file(), and RAND_file_name() are used to load and save entropy in order to speed up, for example, seeding when starting up an application. We recommend to save an entropy estimation value together with the entropy file, because when loading the file, the data will be considered having 100% entropy, but the actual entropy could be much smaller. We also recommend to delete the entropy file after reading it to prevent loading and using the same entropy twice by accident.
2. In a worst-case scenario, where there is no /dev/random, where the current entropy estimation of /dev/urandom is too low and where the entropy estimation of the loaded entropy file is incorrect, the RNG might not get seeded well enough to produce unpredictable random numbers. In this case, it is possible to utilize the RNG without proper seeding.
3. It is possible to use RAND_pseudo_bytes() even if there is not enough entropy in its pool. It should be ensured that there is a minimum of entropy in the pool before using this function.
4. In a multi-threaded application, the RNG provides weak entropy when not re-seeded after an invocation of the fork() function, because the only process-specific information that goes into the RNG state is the PID, which lies in a limited range (for example 0 .. 32768 on Linux) and which repeats. Therefore, we recommend that when using OpenSSL in a multi-threaded application, after each invocation of the fork() function, the RNG should be re-seeded and the entropy counter should be reset.
5. The RNG state is implemented as a ring buffer. When multiple threads concurrently access the RNG state, different ranges of the RNG state are provided to them for reading (extraction) and writing (seeding). Due to the size limit of the pool of 1023 bytes, the ranges

will certainly begin to overlap if a certain number of threads is reached. Due to the concurrent access of the RNG state, the state is not deterministic anymore and therefore it may be possible that the same random data be output multiple times. This is justified by performance reasons. Therefore, we recommend locking the RNG state when operating on the RNG state, thereby serializing access by multiple threads. In case this results in inacceptable performance penalties for certain applications, other measures should be taken, such as using different RNG instances with different seeds.

6. We found that requesting a negative amount of entropy yields a successful return code.
7. /dev/urandom should not be the first entropy source of the RNG.
8. The values of getpid(), getuid(), and time() are of different size depending on the platform. Therefore, the maximum entropy that these functions can contribute depends on the platform. This should be taken into account.
9. To a programmer, it is not transparent which entropy sources are used during implicit seeding. Therefore, a programmer should add entropy from a good source using RAND_add() instead of relying on the implicit seeding.
10. In crypto/dsa/dsa_gen.c, RAND_bytes() should be used instead of RAND_pseudo_bytes() when generating a DSA key.
11. In the DSA context, when seeding the RNG with the hash of the message to be signed, the entropy counter will be increased by 32 bytes when using SHA-256. This entropy estimation is too optimistic.
12. For some hardware RNGs, there is no implementation of the seeding function.

Further Aspects

1. There is no warning when using ./config with non-existing compiler flags.
2. Some compiler flags are not working, for example: *no-tls* and *no-tls1*.
3. There seems to be no way to build OpenSSL with support for TLS 1.1 and TLS 1.2, but no support for TLS 1.0.
4. It seems that there is no compiler flag to turn TLS session renegotiation on or off.
5. The compiler flag *no-rdrand* has no effect when building OpenSSL within a virtual machine.
6. Turning off AES-NI is not possible using a compiler flag.
7. We got a compile error when using the flag *no-sha1*.
8. In crypto/bn/bn_prime.c, the number of Miller-Rabin iterations should be increased when generating a prime number.
9. When verifying a X509 certificate, name constraints seem not to be checked properly. When running the x509test [1] test suite, the ValidNameConstraint test failed.
10. In crypto/dh/dh_gen.c:dh_builtin_genparams(), there is no RNG re-seeding during Diffie-Hellman parameter generation.
11. When using ECDH together with non-standard elliptic curves in for example crypto/ecdh/ech_ossl.c:ecdh_compute_key(), there is no default check whether a point is actually on the curve. This could allow for an attack if the twist of the curve has a small subgroup.
12. In ssl/s3_srvr.c: ssl3_send_server_key_exchange and crypto/dh/dh_key.c: generate_key, there will only be Diffie-Hellman ephemeral keys if the SSL_OP_SINGLE_DH_USE option is used, which is not the default case.
13. Same as above for elliptic curves when using ECDH: SSL_OP_SINGLE_ECDH_USE flag has to be set for ephemeral keys even if the configured cipher suite explicitly demands ephemeral keys.
14. In the RSA self-test (test/rsa_test), there is no test for the widely used exponent 65537.

References

[1] https://github.com/yymax/x509test

[2] https://www.bsi.bund.de/DE/Publikationen/Studien/OpenSSL-Bibliothek/opensslbibliothek.html