Audit of OpenSSL's randomness generation

Jean-Philippe Aumasson, Antony Vennard

13/07/18

Summary

We review the OpenSSL PRNG mainly implemented in openssl/crypto/rand, focusing on the new features discussed in https://www.openssl.org/blog/blog/2017/08/12/random/ and implemented in openssl/crypto/rand/ (approximately 3000 lines of code in this directory). The audit mostly consisted in source code review.

We point out potential security issues as well as possible improvements. We worked for approximately 18 hours on this audit.

Disclaimer: We probably missed bugs, the findings reported may or may not be relevant, we're not responsible for any kind of damage caused our recommendations, etc. etc.

Acknowledgments: This audit work was funded by OSTIF and sponsored by Private Internet Access.

Security issues

1. Insufficient privileges check

Severity

Low

Description

In `crypto/rand/randfile.c` the function:

```
if (OPENSSL_issetugid() != 0) {
    use_randfile = 0;
} else if { ... }
```

is used to disable randfile if setuid/getuid has been set. This is to distrust the environment variable `RANDFILE` where the process linking to `libssl` has additional privileges.

On Linux, posix capabilities are seeing increasing usage (at least Fedora community packaging guidelines have banned the use of setuid/setgid binaries in favour of capabilities for some time now). This check does not account for capabilities.

Recommandation

It is tempting to try to check for specific capabilities, however, this list is subject to change.

However, if a process is started with file system supplied capabilities, the ELF auxiliary variable `AT_SECURE` is set. This *also* checks whether setuid or setgid attributes are present on the file and have given the process privileges.

We therefore recommend that on Linux, all such checks on environment variables (including outside the scope of this audit) use this check:

```
#include <sys/auxv.h>

getauxval(AT_SECURE)
```

which checks for the presence of `AT_SECURE` being set, before accessing environment variables.

2. Entropy bytes discarded

Severity

?

Description

In `rand_pool_acquire_entropy()` in rand_unix.c the following code discards the random bytes collected if `syscall_random()` didn't generate exactly as many bytes as requested:

```
#   ifdef OPENSSL_RAND_SEED_GETRANDOM
    bytes_needed = rand_pool_bytes_needed(pool, 1 /*entropy_factor*/);
    buffer = rand_pool_add_begin(pool, bytes_needed);
    if (buffer != NULL) {
        size_t bytes = 0;

        if (syscall_random(buffer, bytes_needed) == (int)bytes_needed)
            bytes = bytes_needed;

        rand_pool_add_end(pool, bytes, 8 * bytes);
        entropy_available = rand_pool_entropy_available(pool);
    }
    if (entropy_available > 0)
        return entropy_available;
#   endif
```

Indeed, `rand_pool_add_end()` will not increment the length index, and any random bytes will not be used. As a consequence, the random pool will have less seed material than it could.

Another interesting behavior of this code is that even if `bytes_needed` is zero, then the `getrandom()` syscall will still be done, though with a count of zero bytes requested. We first thought that this call with a count of zero bytes was useless, however even then the kernel function `crng_backtrack_protect()` will be called. This function will update the pool state and thereby providing a higher level of forward security (a.k.a. backtracking resistance), though not of prediction resistance. Is this the reason why the syscall is still performed with zero bytes?

Recommendation

Do not discard the randomly generated bytes, since adding more seed material cannot degrade security.

Verify that the behavior observed when the pool is full is the intended behavior.

3. Potentially insufficient size comparison

Severity

Informational

Description

In `ctr_update()` the check for `keylen` "longer than 128 bits" actually only checks that `keylen` is not 128 bits.

```
/* If keylen longer than 128 bits need extra encrypt */
if (ctr->keylen != 16) {
    inc_128(ctr);
    if (!EVP_CipherUpdate(ctr->ctx, ctr->K+16, &outlen, ctr->V,
                          AES_BLOCK_SIZE)
        || outlen != AES_BLOCK_SIZE)
        return 0;
```

In the current configuration this should not be problem, but it'd be safer to replace `!=` with `>` nonetheless.

4. Missing null pointer checks in API functions

Severity

Low

Description

`RAND_drbg_*` functions such as `RAND_DRBG_instantiate()` or `RAND_DRBG_reseed()` don't check that `drbg` isn't null (`RAND_DRBG_free()` does the check though).

Adding this check would make the API safer against user misuse.

Observations and possible improvements

A. Ordering of seed sources

The ordering of seed sources does not seem random, with e.g. for Linux `getrandom()` prioritized over RDSEED/RDRAND, itself prioritized over EGD.

In the current setting only the first source is used if it works, even if more sources are specified. It would not hurt to return as a seed the xor of the data collected from the multiple sources specified. This behavior may also be more intuitive for the users.

B. Support of NONE entropy source

We wondered why `none` is supported as an argument for `--with-rand-seed`, as it will obviously be insecure compared to the other options. As this is risk-prone, maybe user warnings should be added when building with `none` only.

C. Role of additional data

Additional data or "personalization" seems useless for a (secure) DRBG of OpenSSL: if the seeding is reliable then injecting public, low-entropy data does not provide extra value; if the seeding is not reliable then low-entropy data is unlikely to help.

For example, the comment below states that the addtional data "does not need to contain entropy" (sic), in other words that it's fine if said data never changes, while arguing that entropy is "useful" (to be fair, the NIST comments on this don't make much sense either).

```
/* Generate additional data that can be used for the drbg. The data does
 * not need to contain entropy, but it's useful if it contains at least
 * some bits that are unpredictable.
 *
 * Returns 0 on failure.
 *
 * On success it allocates a buffer at |*pout| and returns the length of
 * the data. The buffer should get freed using OPENSSL_secure_clear_free().
 */
size_t rand_drbg_get_additional_data(unsigned char **pout, size_t max_len)
{
    size_t ret = 0;
```

D. Confusing terminology

The term "entropy" is often used confusingly, for example it sometimes refer to pseudorandomly generated data, and notions like "entropy length" (via variables `entropylen`) are introduced, which are unconventional.

E. UEFI randomness generation

In `rand_unix.c` the following defines are made:

```
    # error "UEFI and VXWorks only support seeding NONE"
```

The UEFI specification from 2.4 upwards supports generating random numbers using the protocol `EFI_RNG_PROTOCOL`. The implementation details are in a driver called rngdxe: source of RngDxe and can be summarized as "uses `RDRAND` plus various DRBGs as specified in SP800-90.

We have not audited this code ourselves. We mention this as one of two possible alternatives to "none" on UEFI platforms:

1. Use it.
2. Since it is based on `RDRAND` anyway, use that to collect randomness for the seed value.

F. Windows randomness generation

On pre-Windows 7 platforms, three calls to the CryptoAPI are used:

```
        if (CryptAcquireContextW(&hProvider, NULL, NULL, PROV_RSA_FULL,
            CRYPT_VERIFYCONTEXT | CRYPT_SILENT) != 0) {
            if (CryptGenRandom(hProvider, bytes_needed, buffer) != 0)
                bytes = bytes_needed;

            CryptReleaseContext(hProvider, 0);
        }
```

On Windows XP or above, the CryptoAPI can be bypassed in favor of `RtlGenRandom()`, which is for example used by BoringSSL, Chromium, Ring, Firefox.

BoringSSL does the following in in crypto/rand/windows.c:

```
/* #define needed to link in RtlGenRandom(), a.k.a. SystemFunction036.  See the
 * "Community Additions" comment on MSDN here:
 * http://msdn.microsoft.com/en-us/library/windows/desktop/aa387694.aspx */
#define SystemFunction036 NTAPI SystemFunction036
#include <NTSecAPI.h>
#undef SystemFunction036
void RAND_cleanup(void) {
}
int RAND_bytes(uint8_t *out, size_t requested) {
  while (requested > 0) {
    ULONG output_bytes_this_pass = ULONG_MAX;
    if (requested < output_bytes_this_pass) {
      output_bytes_this_pass = requested;
    }
    if (RtlGenRandom(out, output_bytes_this_pass) == FALSE) {
      abort();
      return 0;
    }
    ...
```

G. rand/ scan-build-safe

The clang analyzer didn't report any issue in rand/. It report 158 issues on other components of OpenSSL, most of which seem minor or false positives (we presume that these have already been investigated, we report it for completeness).