Security advisory: Weak protection of sensitive flash pages allows extraction of AES keys from TI dongles (CC2544)

Author: Marcus Mengs

Last change: March 14, 2019

Disclaimer: Although I work as InfoSec, the research parts presented here have been done in my spare time and are not related to my employer.

Short summary

An attacker is able to extract raw per device key material from dongles with TI SoC, using undocumented HID++ registers.

External link video PoC (youtube): https://youtu.be/5z_PEZ5PyeA

This report adds up to the two reports already handed in. To be precise: issues already mentioned in those reports will not be repeated in great detail in this report.

Devices used for testing

- Logitech K400+ Fw: 063.002.00016 (link encryption for Keyboard reports, up to date hardware, combines mouse and keyboard in single device)
- K360: not notes on firmware version, only used for verification
- Dongle CU0012 Fw: RQR24.07 build 0x0030, Bootloader 03.09 (vulnerable)
- Dongle CU0008: I have currently no access to the dongle, thus I can't provide the fw/bootloader version in this report without further delays (the dongle is vulnerable)
- Dongle CU0007 Fw RQR12.01 (used during research phase, not vulnerable)
- nRF24LU1+ (CrazyRadio PA with customized firmware, Only for PoC to demonstrate live decryption with dumped AES key)

Introduction

After handing in the first to reports, I started cleaning my notes and noticed that I forgot to investigate some things. Amongst those things, there where undocumented HID++ 1.0 registers, which could be accessed using HID++ 1.0 GET_REGISTER and SET_REGISTER. After some fuzzing (test device was the CU0007 dongle) the register 0xD4 caught my eye.

In contrast to former reports, this report has walk-through style, as I think that it helps to develop ideas for counter measures which could stop/slow down an attacker in early analysis/information gathering phase.

Relevant information from previous research, which wasn't mentioned in other reports

The following facts about the flash layout of CU0007 on RQR12.01 have been known from research on previous issues:

- 0x0000.. dongle firmware
- 0x6c00 / 0x6e00 flash page for device and dongle info, each table entry has 16 bytes, first byte is index (index is used for requests to pairing information registers), active flash page is marked with 0x3f (dirty 0x7f ?? not in use 0xff)
- 0x7000 / 0x7200 flash page for key material, 17 byte entries, first byte is device index
- 0x7400.. bootloader

For previous research, flash memory from 0x0000 to 0x7fff has been dumped to a file. This was done using the firmware update mode (bootloader) (see tools provided by Bastille: https://github.com/BastilleResearch/nrf-research-firmware/blob/master/prog/usb-flasher/logitech-usb-flash.py). Instead of the flash write command (0x20) the read command (0x10) was used for dumping. It didn't take much effort to find the proper commands, as they are documenetd in open source projects like fwupd (https://github.com/hughsie/fwupd/blob/master/plugins/unifying/fu-unifying-bootloader.h#L25)

It is of importance, that attempts to read flash pages 0x7000/0x7c00 (using the bootloader commands) return mirrors of pages 0x6c00/0x6e00 in response. This is an effective method for hiding key material residing in those pages. As highlighted in previous

reports, an attacker would have to patch the firmware, in order to dump the real content of pages 0x7000/0x7200. This can not be done easily, if a bootloader with signed DFU is deployed to the dongle. Thus, signed DFU has to be assured.

Investigation of register 0xD4 on CU0007 (Nordic nRF24LU1+ - not vulnerable)

It turned out that register 0xD4 could be accessed using "HID++ GET SHORT REGISTER" requests (SubID 0x81). For my first attempts, handcrafted requests have been issued with `r1=0x00 r2=0x01`, `r1=0x01 r2=0x01` and so on. It turned out that some requests lead to responses with arbitrary bytes, while other requests resulted in errors.

Searching the occurrences of the bytes resulting from 0xD4 register reads in the flash dump, revealed that parameter values from r1=0x00 to r1=0x0f map to data in the active flash page 0x6c00/0x6e00 at offset 0x30..0x3f. Attempts to use values other than 0x00 for r2 resulted in errors, but 0xff worked. Thus I iterated over all possible combinations. It turned out, that r2 serves as LSB and r1 as MSB of a memory address with unknown mapping and the byte at this memory address gets delivered in result. Essentially, this is an undocumented "byte read" register.

The mapping turned out to be something like this, after some investigation:

```
0x0000..0x000f     maps to active flashpage + 0x30..0x3f
0x0010..0x6bff     error
0x6c00..0x6fff     one to one mapping
0x7000..0xfdff     error
0xfe00..0xffff     maps to flash info page 0x00..0x1ff (see table 117 of nRF24LU1+ specs)
```

The mapping for 0xfe00..0xffff wasn't clear to me, at first. Thus, I revisit the firmware dump, which revealed that flash info page access was enabled/disabled for those request parameters.

Again, the regions with data for the AES keys (starting at 0x7000) are not accessible, using reads to register 0xD4. Single records from flash pages 0x6c00/0x6e00, which are not accessible using the register 0xB5 (pairing info), could be read using register 0xd4 (namely entries starting with 0x03).

Testing reads from register 0xD4 on a TI dongle, revealed different result.

Investigation of register 0xD4 on CU0012 (TI CC2544 - vulnerable)

Following the same approach for reading bytes using register 0xD4, the CU0012 revealed this:

```
... snip ...
0xe7e0: ffffffffffffffffffffffffffffffff
0xe7f0: ffffffffffffffffffffffffffffffff
0xe800: 3fffffff02ffffff 2407003088080401
0xe810: 0000000000000000 03ffffffe2c794f2
0xe820: 0106400000000000 000000007fffffff
0xe830: 00180724d02a8da2 1100000000000000
0xe840: 60ffffffe2c794f2 404d88089393ffcb
0xe850: 273b052e03ffffff e2c794f202064100
0xe860: 0000000000000000 20ffffff4108404d
0xe870: 0402014700000000 0000000030ffffff
0xe880: 2d9a9fe11e400000 0900000000000000
0xe890: 40ffffff094b3430 3020506c75730000
0xe8a0: 00000000ffffffffffffffffffffffff
0xe8b0: ffffffffffffffffffffffffffffffff
0xe8c0: ffffffffffffffffffffffffffffffff
0xe8d0: ffffffffffffffffffffffffffffffff
... snip ...
```

Obviously the pairing information follows a different layout:

The index consumes 32 bit, where the most significant byte holds the actual table entry index (padded with 0xff 0xff 0xff) and is followed by a 16 byte table entry (in contrast to 15 bytes on Nordic-based dongles) padded with 0x00 bytes. In contrast to the Nordic dongle, the TI dongle never returns an error for requests to 0xD4, but attempts to read "inaccessible" addresses result in 0xFF bytes.

After re-alignment, the dumped data from above looked like this:

```
offset 0xe800:

02ffffff 2407003088080401 0000000000000000
03ffffff e2c794f201064000 0000000000000000
7fffffff 00180724d02a8da2 1100000000000000
60ffffff e2c794f2404d8808 9393ffcb273b052e <-- interesting data, constructed like AES key input data for
    device 1 !!!!!
03ffffff e2c794f202064100 0000000000000000 <-- dev 1 addr
```

```
20ffffff 4108404d04020147 0000000000000000 <-- dev 1 pairing info
30ffffff 2d9a9fe11e400000 0900000000000000 <-- dev 1 ext pairing info
40ffffff 094b34303020506c 7573000000000000 <-- dev 1 name 'K400 Plus'
```

The entry `0x60ffffff` jumped right into my eye, because it holds full 16 bytes, where the first 8 byte looked familiar to me (4 byte dongle serial, 2 byte device WPID of K400+, 2 byte WPID of TI dongle). The fact that the remaining 8 byte looked somehow random, got me really suspicious. I used the PoC from report 1 (sniffing of pairing) to assure that these byte are not representing the two 4-byte nonces, which are exchanged during pairing. Unfortunately, it turned out they are.

A quick test, if the same entries from this flash page are accessible using requests to register `0xb5` (with `0x60..0x65` as parameter) did not succeed. Anyways, the key material could be dumped using register `0xD4`.

To obtain the AES keys from this data, the content of the `Key generation` section from report 1 could be applied. The device key could be derived by swapping some bytes and applying some XORs with 0x55/0xff. The substitution scheme in use does not add any security. Dumping the 0x60..0x65 table entries, is as good as dumping the keys.

Notes on the PoC

The "key dump" part of the video PoC https://youtu.be/5z_PEZ5PyeA is described in pseudo code (go style) in the following snippet:

```
dumpKeyForDevice(deviceIndex byte) (key []byte) {
    //find flash page with device data
    flashPagesToConsider := []uint16{0xe400, 0xe800, 0xec00}

    activePageAddr := uint16(0)
    for _,pageAddr := range flashPagesToConsider {
        flashByte := u.DumpFlashByte(pageAddr)
        if flashByte == 0x3f {
            activePageAddr = pageAddr
            break
        }
    }

    activePageAddr += 0x04 //offset to first entry
    marker := 0x60 + devID
    keyAddr := uint16(0)
    stepsize := 0x14
    maxSteps := 3 + 5*6 //3 entries for dongle, max 5 entries per device
    for step := 0; step < maxSteps; step++ {
        checkAddress := activePageAddr + uint16(stepsize * step)
        flashByte := u.DumpFlashByte(checkAddress)
        if dB == marker {
            keyAddr = checkAddress
            break
        }
    }

    keyAddr += 4
    res = make([]byte, 16)
    for idx,_ := range  res {
        res[idx],_ = u.DumpFlashByte(keyAddr + uint16(idx))
    }

    //swap bytes back to correct position, change to complement / xor with 0x55 where needed
    key = byteSwap(res) //<-- this is neither encryption nor proper key generation

    return key
}
```

The snippet highlights, that key extraction could be done fast, even with slow single-byte-reads, because the markers are taken into account. This allows to deploy an attack in drive-by-fashion, while keys have to be only dumped once. A possible scenario, not demonstrated in the PoC is to use the dumped keys to decrypt RF frames captured in the past.

Mitigation

Mitigations of report 1 apply:

- secure key generation
- unique AES input data per device in addition to unique key
- don't derive key from non-random data, which could be guessed or obtained otherwise (dondgle serial, WPIDs)

Additionally, all counter measures observed on CU0007 have to applied TI dongles, as they protect from this kind of key dumping:

- dedicated flash page for key material (not mixed with readable pairing info)
- bootloader with signe DFU to avoid reading sensitive flash pages using firmware patches
- sanitization of parameters (address) for register 0xd4, better: disabling this register completely
- assure Secure DFU on all firmwares (forced updates or a PR campaign for updates should be considered, as not all dongles ship with current firmware)

Note 1: I haven't tested if this data could be dumped using flash reads via bootloader (like done on CU0007), but it isn't unlikely, as the data is mixed in a single page with the pairing info.

Note 2: Being able to regenerate the RF addresses from accessible device data weakens security further. As shown in the video PoC, this helps an attacker to ultimately start sniffing on the correct RF address, which he has to obtain first otherwise (f.e. using pseudo promiscuous mode proposed by Travis Goodspeed).

Note 3: The PoC has been tested on CU0008, which has been found vulnerable, too. I assume all TI dongles are affected, which is likely due to a high level flash storage library, which allowed storing 16 byte records with indices. Maybe somebody thought, merging keys into the flash pages of pairing data could save some flash writes/erase cycles. To be honest, I have no reasonable explanation for this design decision.

Note 4: Being able to extract key material renders proposed measures for secure key exchange (report 1) useless.

Update with developments up to August, 2019

- CVE-2019-13055 was assigned to the described vulnerability
- It turned out that receivers of presentation clickers R500 and SPOTLIGHT are affected by the same vulnerability (not mentioned by Logitech troughout ongoing communication). Because of this, the issue was handed over to German CERT. For aforementioned presentation clickers, the key extraction vulnerability could be combined with a bypass for the applied keyboard input filter, leading to wireless injection of arbitrary keyboard keys, once the link encryption key is dumped. The user has no easy way to exchange this key, once compromised, as this would involve re-pairing the presentation clicker to the receiver. CVE-2019-13054 was assigned.
- Logitech announced a patch for CVE-2019-13054/13055 for August 2019.
- It turned out that receivers of other Logitech devices, like G-Series LIGHTSPEED are affected, too.
- The patch released in August 2019 could easily be bypassed, by downgrading the firmware (and upgrading it again after AES key extraction, if needed). The original attack took about 1 second, the bypass with firmware downgrade takes additional time (about 23 seconds).
- As most receivers should ship with bootloaders which only accept signed firmwares, a downgrade attack requires a signed firmware. For Unifying receivers and G-Series LIGHTSPEED receivers, vulnerable downgrade firmwares with signature are publicly available.
- For receivers without available downgrade firmware (R500, SPOTLIGHT ...) a downgrade attack is still possible. Instead of the missing signed firmware for those devices, a vulnerable signed Unifying firmware could be used (RQR24.07). Once the AES key is extracted, the intended firmware could be flashed, again.
- To my best knowledge, R500 and SPOTLIGHT clickers should NOT remain vulnerable to keystroke injection, once the latest patches are applied to the receiver. Although a downgrade attack for link-key extraction is possible, the firmware has additionally been updated with a new blacklisting filter for unintended keystrokes. This filter effectively prevents keystroke injection attacks. Bypassing this filter, would require a downgrade with a vulnerable signed firmware, meant to be used with this exact receivers, which isn't publicly available.
- A possible attack scenario for R500 / SPOTLIGHT which remains (I havn't tested) is a cross-grade. If a receiver of such a presentation clicker is flashed with a vulnerable and signed Unifying firmware, it is likely that the device still works (pairing information are preserved, RF protocol for Unifying and those presentation clickers are mostly the same)
- the PoC tool `munifying` has been released, it allows:
  - testing receivers for CVE-2019-13054/13055
  - flashing receivers with different firmwares (upgrade, downgrade, "cross-grade")
  - pairing / unpairing of devices to all supported Logitech receivers
  - source code link: https://github.com/mame82/munifying