Security advisory: Logitech Unifying vulnerabilities allow keystroke injection into encrypted links between device and dongle at latest patch state via RF, without knowledge of the encryption keys

Author: Marcus Mengs

Last change: February 27, 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.

This report adds up to the advisory "Multiple Logitech Unifying vulnerabilities allow eavesdropping of link encrypted RF traffic and live decryption" and thus doesn't go into details on topics already covered in the previous report.

Short summary

An attacker is able to inject arbitrary keystrokes into Unifying dongle with latest patch state, if a device with keyboard capabilities is paired to the respective dongle. The vulnerability applies to dongles with latest patch state (Bastille security issue #2 keystroke injection is addressed with patch).

The attack could be implemented in multiple flavours:

Variant 1) With physical access to the device, in order to allow an attacker to press "trigger" keys, which result in arbitrary USB HID output reports (LED reports). Those reports ultimately end up in unencrypted RF frames directed from the dongle to the device. An attacker could capture those RF frames (with now known plain text), modify them to his needs and play them back to the dongle via RF. This requires one time access to the device for some seconds. After successful capture of arbitrary RF frames, the attacker could inject as many keystrokes as he likes, as often as he wants, even if device/ dongle are powered off and on again. This holds true up to the point encryption keys are re-generated (re-pairing).

Variant 2) The attack could be extended to a brute force based approach, which doesn't require physical access to the device or dongle.

Additionally, the current implementation of link encryption is vulnerable to very simple replay attacks, which don't require any knowledge of the protocols used by Unifying devices or key material.

A possible version of attack Variant 1 has been demonstrated in a video PoC.

External link video PoC (youtube): https://youtu.be/EksyCO0DzYs

For variant 2, I currently can't provide a PoC, due to time constraints. The theoretical approach is described throughout this advisory.

There exist multiple vulnerabilities leading to these attack vectors. The root causes are:

- that AES encrypting isn't applied on the plain payloads, but the cipher text is produce based with a proprietary implementation relying on AES with (insecure) input data and afterwards XOR'ed onto the plaintext. The result is transmitted via RF
- and that the custom implementation of counter mode AES provides insufficient protection against counter re-use, even after patches for the aforementioned vulnerability, reported by Bastille, have been applied.

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)
- Dongle CU0012 Fw: RQR24.07 build 0x0030, Bootloader 03.09 (all issues reported by Bastille patched, signed firmware only; used to verify discovered vulnerabilities)
- nRF24LU1+ (CrazyRadio PA with customized firmware)

Introduction

Taking into account the observations described in the first report, the following questions have arisen:

- How could a Unifying dongle assure, that a counter isn't re-used, while a device should initialize counters to a secure random value after power on (it has been observed, that devices don't store the last used counter persistently).
- How could a Unifying dongle assure, that a counter is the successor to the previous one, while RF frames could get lost and thus the transmitting device could skip counters.

To answer those questions, the knowledge on the pairing vulnerabilities has been applied, like this:

1) Pairing of a valid devices has been sniffed in order to calculate the keys in use (vulnerabilities described in first report)
2) Encrypted RF frames for keystrokes have been generated on both, the real device and in software, utilizing the known algorithm for link encryption (called "virtual device" in this report).
3) Instead of using the successor to the counter of RF frames from the real device (Logitech K400+), the virtual device used a random start counter (unrelated to the counter from the real device) which was incremented for successive RF frames.
4) Keystrokes were send from the real device and the virtual device in alternating fashion, using the different start counters and respective USB HID output reports have been inspected for valid results.

The results of this tests are the basis for the vulnerabilities described in the following sections. All tests have been done with the dongle mentioned in section "Devices used for testing", which has latest firmware applied (patched against all known vulnerabilities).

Vulnerabilities

Vulnerability 1 - counter reuse after overflow of dongles counter storage

Based on the testing scheme described in the introduction, the following observations have been made:

- the dongle successfully prevents usage of non-successive counters in encrypted RF frames (not incremented by one)
- the dongle accepts new, non-successive counters if a fixed amount of reports with successive counters (incremented by one for each new RF frame) is transmitted
- for the dongle under test, this fixed amount of RF frames with a random start counter has been 23

To clarify these points: When the real device has sent encrypted RF frames, follow up RF frames from the virtual device with invalid counters, but otherwise valid encrypted data (report type `0xD3`, valid checksum, valid AES input data based on counter in use, counter in use appended to payload, valid CRC, valid device specific AES key, 8 bytes resulting cipher XOR'ed to payload, last byte of plain payload `0xC9`) don't produce HID output reports. This only holds true, up to the point 23 RF frames have been send to the device. Starting from the 24th RF frame received from the virtual device, the dongle produces HID output reports for successive frames from the virtual device.

Conclusion: This specific dongle stores the last 23 counters used (not persistently, runtime only) and assures the successive frames use an incremented counter. This mechanism is called "counter cache" throughout this report. If a received frame contains an invalid counter (not incremented by one), it is written to the counter cash, anyways, which ultimately drops an already stored counter. When all values in the counter cache are overwritten, with frames using a new start counter, all successive RF frames are considered valid.

An attacker is able to replay keystrokes, based on captured RF frames, as long as more than 23 frames are captured (first 23 RF frames wouldn't reproduce keystrokes, as they are needed to flush "counter cache").

Vulnerability 2 - Information leakage on type of keystroke

This observation is specific to the K400+ device and hasn't been confirmed for other devices, yet. Beside RF reports of type `0xD3`, which represents encyrpted HID keyboard data, there're other reports in ongoing communication. It has been observed, that key down reports are preceded by a "set keep alive" notification frame (`0x4F`) which sets the keep-alive interval to 8 ms. The key down report is followed by keep-alive notifications (`0x40`) in an 8 ms interval, till the respective key up report (`0xD3`) arrives. The key up report is ultimately followed by multiple "set keep alive" reports (`0x4F`), which re-adjust the keep alive interval to an amount greater 8ms.

This allows an attacker to distinguish encrypted key up reports from key down reports, based on the following rule:

- a key down RF frame has to be preceded by a "set keep alive" RF frame

- a key up RF frame follows a key down RF frame (if keep-alive reports are ignored) and has to be succeeded by at least one set keep-alive RF frame

This knowledge helps to distinguish key up from key down reports in most common cases (e.g. not if a key down report is followed by a new key down report, without key up report in between), which again means known plaintext for about 50 percent of the captured reports (a plain key up report consists of 0x00 for modifier, 6-times 0x00 for keys and 0xC9 byte for decryption validation). XOR'ing this known plaintext back to the respective encrypted parts of the RF frame leaks 8 cipher bytes for the specific counter in use (only 0xC9 has to be XOR'ed in this case).

XOR'ing back the key release plaintext to encrypted RF frames isn't much of an issue, but being able to distinguish key down from key up reports is an issue. This is because the attacker now knows which (replayable) RF frames have XOR encoded keystrokes applied. This is a crucial fact for the following sections.

Vulnerability 3 - "Whitening" captured reports for keystroke injection with brute force approach, utilizing unencrypted LED output reports (attack variant 2)

With the issues describe so far, an attacker is able to replay encrypted RF frames and knows which of those frames result in key up reports.

The attack described here has to be considered theoretical, as no PoC has been implemented, yet. If the attack could be carried out successfully, the attacker needs no physical access to device or dongle.

The attack assumes the following preconditions to be met:

- the attacker capture more than 23 encrypted keyboard RF frames (type `0xD3`)
- the attacker knows which frames represent key up and and which frames represent key down reports (see last section)
- the captured sequence doesn't contain key combinations (multiple successive key down reports, without key up report in between)

The attacker captures Rf frames till this preconditions are met. Than he starts replaying the frames, but increments the 2nd of the 8 encrypted payload bytes for each key down frame by one. In result, the "target dongle" produces HID output reports, which aren't known by the attacker, but differ from the HID output reports of the unmodified RF frames. Between the key up and key down reports, the attacker sends keep-alive reports, to assure output reports, which are delivered back via ACK payloads could be received. This step is repeated, till the attacker receives an unencrypted LED output report. Depending on the LED state change (on to off or off to on) the LED report arrives after key up or key down. At this point, the attacker knows that the modified key down report produced an LED output report, which again means, that the plain key report for the given encrypted payload is now known. For example, if a modified encrypted RF frame resulted in a LED report, where the CAPS LED changed (compared to the last received LED report), the attacker knows that byte 2 of the resulting HID report produced by the dongle was 0x39 (CAPS LOCK). The attacker now XORs the respective key byte to the respective encrypted RF frame (0x39 in example), before replaying the whole RF frame sequence again. In result, the former key down report is a key up report, now (all key bytes zeroed out).

These steps are repeated, until all RF frames are whitened.

Now the attacker is able to inject arbitrary keystrokes, by XORing the intended HID key bytes to the whitened RF report sequence (for example 0x04 for KEY_A).

Note: The described attack doesn't account for modifier bytes in the captured RF frames, but could be repeated, till keystroke injection works reliably.

Note 2: This attack, obviously, produces random keystrokes during the brute force phase. The attacker could take additional educated guesses on the state of the target host, to keep the impact low. For example locking down a Windows host is usually done with a key combination (e.g. GUI_KEY + L) without successive key presses. As the attacker is able to detect key combinations (multiple successive key down reports, unless the user is able to press all keys exactly at the same time, with 8ms precision) and is able to detect that no successive keys are pressed (no further key reports, increased keep-alive delays), chances are high, that the attacker knows, at which point in time a Windows host is locked. Brute forcing replayed RF frames with the goal of whitening, has nearly no impact on the target (in terms of producing unpredictable results, due to random keystrokes), while LED reports still work.

Vulnerability 4 - Injecting arbitrary keystrokes without knowledge of encryption key, after physical device access (attack variant 1)

This kind of attack has been proven to work (video PoC) and is a simplification of attack variant 2.

Instead of brute forcing captured encrypted RF keyboard frames with unknown plain text (resulting USB HID report not known), the attacker manually toggles a key which results in a change of the keyboard LED state. For example, pressing CAPSLOCK multiple times, results in (unencrypted) LED output reports, which only have the CAPS LOCK LED changed.

On RF end, the attacker still tries to capture more than 23 RF frames, but a captured sequence is only considered valid, if all key up/key down RF frames received included a LED output report, which has the state of the CAPS LED changed.

Thus, pressing CAPS 12 times on the target (trigger), results in 24 captured RF frames (fully automated).

The attacker knows, that every key down frame has KEY_CAPSLOCK (0x39) encoded and no modifier present, as the keystrokes have been produced by an attacker.

Whitening of the encrypted RF frames captured is now accomplished by XOR'ing 0x39 to the second encrypted byte of every captured key down frame.

In order to inject arbitrary keystrokes, the whitened RF frame sequence is replayed once, in order to flush the dongles counter cash. In the next run of replaying the sequence, the actual HID key codes, which should be injected are XOR'ed onto every key down frame.

This is done in a loop, till all keystrokes are injected successfully.

A video PoC of this attack variant has been made available (without the technical details).

Possible mitigation measures

Instead of XOR'ing cipher text to the plain payloads, real encryption should be applied. As this would result in 16 cypher bytes (where the plaintext has to be part of the AES input data), a way has to be found to transmit the 16 cipher bytes and the 4 byte counter in a single RF frame. This is problematic, as the maximum report length in current implementation is 22 bytes, with 3 bytes already in use (device ID, report type, 8bit CRC).

Validating only some of the last counters isn't secure. Due to memory constraints a different mechanism has to be found. A possible solution would be to persistently store every counter, which overflows the dongle counter cache (every 23th counter) to device and dongle in persistent fashion (flash write). If this goal is achieved, it could be avoided to allow random counters, after power-cycles of the device. Of course, many mor counters would have to be cashed, as storing every 23th counter on flash would be insane.

There is a strong need to encrypt all RF payloads, especially:

- SYSTEM CONTROL reports
- MEDIA KEY reports
- LED output reports

Note: It is hard to provide good mitigations, because the content of this report is based on a pure blackbox approach during R&D. This again means, assumptions on the algorithm used for counter validation couldn't be confirmed, as I did no static firmware analysis to get a full understanding of the full algorithm.

Anyways, it should be considered that a real attacker takes this effort, as the relevant code parts could easily be found with binary diffing of firmware patches.

Final words

Although Logitech got in touch with me quickly after PoC videos of the results of this research have been disclosed (which don't including technical details, which place an additional on customers), I had issues to find a valid way to hand in the security reports to Logitech. Beside the fact, that I had to take a two weeks break after making the video PoC available, there seems to be no proper way to hand in security reports Logitech, a HackerOne program was referred (for reporting and disclosure policy), which unfortunately isn't listed in the HackerOne program dictionary, at time of this writing. So I agreed with *REDACTED* to hand in the report via encrypted mail.

Anyways, there was no clear statement on disclosure policy, thus I want to follow common rules for "responsible disclosure", as there's a broad community of interest on this specific issues and I used a huge amount of my spare time (free of charge) to investigate them. To be precise:

- I plan to publicly disclose the content of this report, as soon as the issues are addressed by Logitech

- I plan to publicly disclose the content of this report in 30 days, if Logitech doesn't consider the issues valid or addresses them
- I plan to confirm requests to present this material throughout security conferences (not before III/2019)
- Valid key material won't be disclosed at any point, as it doesn't add up to improve security or to protect end users
- I reserve the right to share parts of this material with "Chaos Computer Club" and/or "Vulnerability Lab", if needed to protect myself, as the HackerOne program I was pointed to in order to give me this kind of protection during responsible disclosure simply doesn't exist.