Security advisory: Multiple Logitech Unifying vulnerabilities allow eavesdropping of link encrypted RF traffic and live decryption

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.

Short summary

An attacker is able to obtain keys for "AES encrypted" RF traffic between Unifying devices and dongles, with latest firmware patches applied, by monitoring device pairing. This allows decryption of RF traffic to eavesdrop key presses (passive RF monitoring) and injection of arbitrary keystrokes via RF (active RF transmission).

If an attacker gets physical access to a dongle and a paired device (like pre-paired device-dongle-combinations which are shipped) the attack could be extended to a drive-by approach: The attacker could re-pair the device, which leads to regeneration of encryption keys. The relevant data for key derivation, again, could be sniffed from RF. The attack could be fully automated and takes some seconds to execute, as the relevant USB HID commands (HID++ 1.0) are publicly documented. A semi-automated approach has been demonstrated in a video PoC (USB part of pairing/unpairing has been done manually).

External link video PoC (youtube): https://youtu.be/1UEc8K_vwJo

Another scenario, which requires physical access to a Unifying dongle only, is firmware hot-patching with modifications to allow extraction of key material stored on internal flash of the dongle. This attack, again, takes only seconds and allows extraction of keys for all paired devices (even if the aren't physically present). Firmware hot-patching relies on dongles with bootloaders, which don't require a signed firmware during DFU. Although latest firmwares have bootloaders which rely on signed DFU, but brand-new devices have been found to be shipped with dongles with outdated firmware and bootloader.

There exist multiple vulnerabilities leading to these attack vectors, but the root causes are the absence of a cryptographic secure key exchange during pairing and the absence of a secure key generation algorithm.

Devices used for testing

- Logitech K520 (no additional info, as pairing limit was reached during 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)
- Dongle CU0007 Fw RQR12.01 (Used to test firmware based attacks)
- nRF24LU1+ (CrazyRadio PA with customized firmware)

Introduction

Searching for additional material for Security Awareness talks, I reviewed Logitech vulnerabilities on outdated Unifying hardware (mostly reported by Bastille and summed up under the name "MouseJack"). I asked a colleague of mine to test one of the available tools (Jackit) against a Logitech Presenter, widely used throughout our Organisation. To my surprise he immediately succeeded injecting keystrokes. At this point I jumped back in and started some additional research, in order to protect our organisation from damage and provide our users with additional information for self-protection against MouseJack related attacks. The research started out with the initial goal of getting tools like "Jackit" working more reliably in terms of device discovery and successful keystroke injection attacks (live demos of our internal talks have to work immediately). At this point in time, it wasn't clear if we are able to "polish" MouseJack attacks in a way that we can use them for talks, thus I moved all R&D parts to my spare time. So the content of this report is not related to my employer.

The initial research included inspection of RF and the USB communication of the dongles and devices mentioned in the last section.

For RF investigation a nRF24LU1P with a customized nrf-research-firmware and an SDR have been used. Information on the RF protocol are publicly available from prior research (ESB at 2Mbps, 16bit CRC, dynamic packet length). Information on

USB protocol for Unifying dongle (DJ-reports, HID++ 1.0/2.0 reports) have been available from public Logitech drafts. The approach of "Jackit" could quickly be optimized, by investigating channel usage of different device/dongle combinations, channel hop timings, patching issues in the nrf-research-firmware (f.e. fix of flushing of unconsumed RX frames during sniffing).

After this optimizations the following observations have been made:

- The RF part of the pairing process could be passively sniffed from start to finish without much effort (even if RF address and channel changes are involved)
- Selection of RF channels to take into account in order to sniff communication with a dongle in pairing mode or a dongle communicating with a normal device could be gathered/optimized (channel usage, channel order, channel hop timing) to follow communications more reliably.
- RF reports which could carry over arbitrary data to USB HID reports without disrupting device/dongle functionality could be identified (f.e. frames with HID++ reports for device index of dongle)
- USB output reports which could carry over arbitrary data to RF frames could be identified (f.e. DJ reports directed to proper device index)
- The USB part of pairing was already well understood from the Logitech drafts (set LOCK / unpair devices by index)

After the observation of USB reports/RF frames, which are able to carry arbitrary data from USB HID to RF and back, without violating the underlying Unifying protocol, the initial research goal was extended by the idea to implement a covert channel using unmodified Unifying hardware (covert channels utilizing USB HID or 802.11 are already part of my portfolio, a covert channel utilyzing widespread proprietary RF hardware nicely adds up to this). The covert channel research isn't part of my reports, as it doesn't rely on security issues specific only to the Unifying ecosystem itself.

Anyways, in order to deploy such a kind of covert channel, some kind of client-side agent needs to be deployed, which brought me back to keystroke injection vulnerabilities, as this attack vector relies on the same hardware as the covert channel itself. As all publicly known vulnerabilities related to RF keystroke injection should be patched, I refocused R&D on:

- Unifying encryption
- exchange of key (material) for link encryption (closely related to device pairing)
- generation of link encryption keys (assumed to be related to device pairing)
- vulnerabilities leading to keystroke injection into encrypted links, without knowledge of keys

First, I started to rebuild the attacks presented by Bastille from scratch, targeting the CU0007 dongle with outdated firmware as research object. As highlighted above, understanding the pairing process is one of the keys to do further investigation on possible vulnerabilities in link encryption, which could lead to keystroke injection or, even worse, decryption and eavesdropping of link encrypted RF communications between dongle and device.

Emulating a dongle in pairing mode and emulating arbitrary devices which pair to real Unifying dongles could be achieved, after blackbox analysis of sniffed RF pairing traffic and some educated guesses on the meaning of the data exchanged. Additionally, forced pairing could be reproduced, using a device RF address already paired to a dongle, instead of using the global pairing RF address. The findings to this point haven't been considered as vulnerabilities, as patches are available for all issues. What I was wondering about, was, how secure AES keys could be generated and exchanged based on the inspected pairing RF traffic. There just wasn't enough data with high entropy involved.

My understanding of the pairing process, based on the aforementioned blackbox observation, is shown in the next section. It became clear, that only 4 bytes of (pseudo) random data is involved for each peer (dongle and device), which seems not enough for a secure key exchange algorithm.

The remaining sections of this report summarize the issues uncovered during investigation of the pairing process. As stated the mentioned research on the covert channel idea isn't part of this report, neither would it be specific to Logitech Unifying technology (although keystroke injection vulnerabilities could help in deploying client side payloads for such attacks).

Note: I don't consider changing channels or RF addresses during pairing, to be valid security measures, as neither of them reliably could hinder an attacker sniffing a full pairing process. So I don't go into much detail on this.

Note 2: Using burst transmissions, or ESB specifically, isn't considered as security measure, throughout this report.

Interpretation of pairing process, as seen during passive RF monitoring

1 Pairing request, phase 1 - device to dongle

Pairing request 1 is sent on different channels, till the dongle's channel is hit (determined by received ack payload). The dongle in pairing mode is identified, by using the "global pairing" RF address `bb:0a:dc:a5:75`.

```
byte 0:      random sequence, used throughout this pairing phase
byte 1:      0x5f, pairing request, with successive data
byte 2:      0x01, pairing phase (1 in this case)
byte 3..7:   RF address currently used by device
             Note: for older devices, this leaks the RF address of the currently paired dongle, for newer
                 devices randomly chosen
byte 8:      0x08 unknown
byte 9..10:  WPID of device (could be looked up in databases, after visual device inspection)
byte 11:     0x04, likely protocol identifier (0x04 == Unifying)
byte 12:     unknown
byte 13:     device type (visually indicated Unifying software after pairing; mouse, keyboard, numpad etc.)
byte 14:     device capabilities (bit 0 - link encryption, bit 2 - not Unifying compatible if unset)
byte 15:     unknown, likely additional CAPS
byte 16..20: padding
byte 21:     Logitech checksum
```

Note 1: Although the pairing address was mentioned in prior research by Bastille, it could easily be discovered by monitoring RF (f.e. with SDR).

Note 2: If a known device address is used instead of the global pairing address, this leads to forced pairing on an unpatched dongle.

Note 3: As mentioned, changing channels during communication couldn't be regarded as security measure. Even if channels are chosen adaptively by the dongle (based on my observations, this is not the case in pairing mode), finding the right channels to sniff is an optimization problem. This "problem" has been solved with low cost hardware (nRF24LU1+) during R&D, but isn't part of this report, as not directly related to Unifying. Additionally, an attacker utilizing SDR hardware, could monitor relevant multiple/all channels concurrently (SDR equipment with 60MHz bandwith costs less than 400 USD and covers about 87 percent of the relevant Unifying channels). If the pairing process runs without RF interference, it is unlikely that the dongle changes the channel at all, after the initial pairing request, even if the RF address used by the device changes. The video PoC on Twitter shows how the CrazyRadio PA follows a dongle in pairing mode across the channels in use, while the dongle is hopping. Chances to hit the correct channel, before the device (which wants to pair) hits the channel with its pairing request are about 50 percent, for the unoptimized approach in use.

Note 4: Outdated Unifying devices leak the address of their currently paired dongle, with every pairing request: If a device is powered on, while the corresponding dongle is not in range (or not powered), the device sends a pairing request on all channels, which includes the current device RF address. The respective dongle RF address could be derived from this address. Although newer devices seem to chose random addresses in pairing requests, all inspected devices start communication using to the current RF address on channel 5 (2404 MHz). As shown later, sniffing this address means knowledge of 4 out of 16 bytes key data. This is a security vulnerability on its own. This kind of address leakage could hardly be avoided, considering the fact that the dongle does no active TX, unless it receives ESB frames from a device (basic principle in Unifying communication scheme). Because of this fact, IMO this issue could only be addressed by avoiding to use the RF address as part of key generation, at all.

2 Pairing Ack Pulls phase 1

The term "Ack pull" is used, whenever the device needs a RF response from the respective dongle. Due to the nature of the underlying RF protocol (ESB with auto ack, payloads from PRX to PTX only on next ack), the device has to send additional RF frames, in order to receive follow up (response) data from the dongle. This data couldn't be shipped back in the first ack payload, as it has already to reside in the dongle's TX FIFO, when the RF frame from the device arrives. So in order to give the dongle time to process inbound data, the response has to be pulled with additional RF frames, which are called "Ack pulls" here.

```
byte 0:      sequence number from request
byte 1:      0x40 keep alive notification (special kind ??)
byte 2:      0x01, pairing phase (1 in this case)
byte 3:      first octet of device' current RF address
byte 4:      Logitech checksum
```

3 Pairing response phase 1 - dongle to device

Ack pulls are send from device to dongle, till a pairing response is received, which looks like this

```
byte 0:      sequence number from request
byte 1:      0x1f, pairing response, with successive data
byte 2:      0x01, pairing phase (1 in this case)
byte 3..7:   RF address announced to the device (first 4 octets are dongle serial)
             Note: for older devices, this leaks the RF address of the currently paired dongle, for newer
                 devices randomly chosen
byte 8:      0x08 unknown
byte 9..10:  WPID of dongle
byte 11:     0x04, likely protocol identifier (0x04 == Unifying)
byte 12:     unknown
byte 13:     unknown not investigate
byte 14:     unknown not investigate
byte 15:     unknown not investigate
byte 16..20: padding
byte 21:     Logitech checksum
```

After the device received this pairing response, it sends an additional "Ack pull". As soon as it receives an (empty) Ack for this last "Ack pull", it changes its RF address to the one announced by the dongle. The dongle opens an RX pipe for the newly announced RF address, too, and successive communication goes on using the new RF address.

Note: Changing the RF address doesn't necessarily mean the dongle starts channel hopping, again. Even if this would be the case, an attacker could follow the communication, as pointed out earlier.

4 Pairing request, phase 2 - device to dongle

```
byte 0:      random sequence, used throughout this pairing phase
byte 1:      0x5f, pairing request, with successive data
byte 2:      0x02, pairing phase (2 in this case)
byte 3..6:   random nonce of device (all 0x00 for devices without link encryption)
byte 7..10:  serial of device (used to distinguish devices with otherwise same pairing data)
byte 11:     unknown
byte 12..20: not inspected / padding
byte 21:     Logitech checksum
```

5 Pairing Ack Pulls phase 2

```
byte 0:      sequence number from request
byte 1:      0x40 keep alive notification (special kind ??)
byte 2:      0x02, pairing phase (2 in this case)
byte 3:      first octet of newly assigned RF address
byte 4:      Logitech checksum
```

5 Pairing response phase 2 - dongle to device

Ack pulls are send from device to dongle, till a pairing response is received, which looks like this

```
byte 0:      sequence number from request phase 2
byte 1:      0x5f, pairing request, with successive data
byte 2:      0x02, pairing phase (2 in this case)
byte 3..6:   random nonce of dongle
byte 7..10:  serial of device (mirrored from pairing phase 2 request)
byte 11:     unknown, mirrored from request
byte 12..20: not inspected / padding
byte 21:     Logitech checksum
```

6 Final pairing phase

I don't go into detail on the final phase, as it doesn't add much value to the security issues in key exchange or key generation.

Worth mentioning: The final phase exchanges the device name string in a length-value fashion. Due to my observations, the length value isn't checked (no further investigation done on this). If this is true, handing in a name with a non-matching length value could lead to additional exploit vectors.

Sum up, to this point

As already mentioned, the interpretation of the pairing communication from RF lead to the question how the inspected traffic could serve for secure key exchange. The only random data which could not be determined by an attacker (without sniffing the pairing itself) are the two 4-byte nonces.

Observations to this point, already lead to an issue, allowing an attacker to pair non-Unifying Logitech devices to a Unifying dongle, if the device doesn't use link encryption (random nonces not of relevance). This isn't a real security concern and thus not regarded as dedicated security issue.

The workflow for pairing a non-unifying device (like a presenter) could look like this: 1) Emulate the presenter as pairing device against a real dongle in pairing mode. Change byte 14 in the emulated pairing request to have the "Unifying compatibility bit" enabled. 2) Store the address assigned by the dongle 3) Emulate a dongle in pairing mode, turn off and on the presenter (or non

Unifying device), ignore the missing "Unifying compatibility" bit in byte 14 of arriving pairing request, assign the stored RF address in pairing response 1 4) Communication between real dongle and non-Unifying Logitech presenter should work, now, as they use the same RF address and RF report format for keyboard reports

The facts gathered up to this point and the concerns on the key generation algorithm raised the need for further inspection. Thus the firmware of the unpatched dongle (CU0007, RQR12.01) was dumped for statical analysis.

The main results of this inspection are part of the next section

Static analysis of link encryption and key generation on firmware 12.01 for CU0007

Decryption function (same as encryption, as XOR based)

An encrypted RF keyboard report on RF looks like this

```
byte 0:        device Index (0x00, as dongle is addressed and device specific RF address in use)
byte 1:        0xd3, report type - encrypted keyboard report
byte 2..9:     encrypted keyboard payload
byte 10..13:   32 bit counter value, chosen randomly on device power up, incremented by 1 for succesive
    reports
byte 14..20:   padding
byte 21:       logitech checksum
```

The function for decryption of received RF frames is easy to spot in firmware, as it is the only one which writes to the SFR responsible for AESKIN data (as documented for nRF24LU1+, which is known to be used by this dongle).

Inspecting the aforementioned function, reveals the following details to a possible attacker, due missing obfuscation: - AES mode in use: 128bit ECB (was wondering why CTR was not used natively, but there're valid reasons) - storage address of AES input data for encryption in external RAM (0x8311 for this specific firmware) - storage address of AES KEYs for each paired device in external RAM (0x81e0 + 0x10 * devIdx for this specific firmware) - byte 7..11 of (otherwise static) AES input data are substituted by counter (decreases entropy) - only first 8 bytes of created AES cipher are XOR'ed onto the encrypted part of payload (process is reversible with same key material and "algorithm") - payload is validated based on the fact that last byte has to be 0xc9 after decryption (leaks a arbitrary byte of cypher text with every RF frame, if known to attacker)

Further analysis from this point

In order to verify this facts, valid key material has to be extracted from a dongle which has a link encrypted device paired. There are multiple ways to achieve this on a dongle using a bootloader which allowing unsigned firmware updates.

One possible way is, to patch the function which decrypts keyboard RF payloads, in order to include bytes of arbitrary memory addresses ()from external RAM) in the resulting HID output reports, instead of the resulting HID key codes.

After extraction of AES input data (globally shared between all Unifying devices) and the device specific AES keys (generated during pairing) the assumptions of the last section could be validated.

This alone poses multiple security risks on end users, documented in the next section. This section, again, is followed by a section with mitigation measures, before moving on.

Security risks (up to the facts described)

1) An attacker who gets access to a Unifying dongle with paired devices, but with a bootloader which doesn't force signed firmwares updates, could "hot patch" the dongle firmware in order to leak key data of already paired devices.

I assumed this is less of an issue, because bootloaders with signed DFU are available for all dongles AFAIK. Unfortunately, I bought new devices with dongles, which not only shipped a bootloader accepting unsigned firmwares updates, additionally the dongle in question was not patched for known vulnerabilities AND THE UNIFYING SOFTWARE DISABLED THE UPDATE BUTTON (Dongle CU0008, fw 24.01 build 0023, bootloader 01.08)

2) If an attacker is able to obtain "template" AES input data, he knows 50 percent of the key material in use, across all Unifying devices. Beside the method shown, this AES input data template could be extracted with static analysis of public available firmware files for Unifying dongles. Basically, constant values are written to the respective memory locations in external RAM by the function deploying this AES input data template and only the 4 byte counter is substituted afterwards. The implementation suffers from missing obfuscation. Not using the raw unencrypted payload as part of the input to AES

ECB is an issue on its own, as XOR'ing this payload with parts of the resulting cipher couldn't be called AES encryption, anymore.

3) The XOR based "encryption" algorithm brings up additional concerns on the implementations robustness against replay attacks. The PoC for compromise of this encryption scheme is part of another report.

4) Visual inspection of the raw extracted per-device AES keys, alone, emphasises the concern that a too simple key generation algorithm is in use. This is because the AES keys have large common parts amongst different paired devices and obviously contain parts of data exchanged during pairing, even worth, data directly related to device types (WPIDs). At this point it is obvious, that a over-simplistic key generation algorithm is used, if there's anyone at all.

Mitigation for risks described so far

1) It has to be assured that recent dongles ship with bootloaders only accepting signed firmwares. The Unifying software should force firmware updates on outdated dongle firmwares. Firmwares should be encrypted or better obfuscation should be applied to parts relevant to device link encryption. Outdated firmwares should be removed from public resources, if possible (backwards compatibility is fighting security, here), as they contain key material (AES input data) which could be extracted by attackers, without in-depth knowledge on firmware reversing.

2) AES input data should be generated dynamically per device or, even better, the unencrypted payload should serve as part of the AES input data for real encryption. It could be assumed, that an attacker could extract the static part of the AES input data from any given Unifying dongle/device firmware, without much effort (as this data is pre-shared globally between all devices and dongles and thus contained in all firmware). This again means, an attacker knows 16 out of 32 bytes of data, used during link encryption (16 byte AES input template + counter sniffed from RF frame, 16 byte AES key is still unknown at this point)

3) Part of a dedicated report, but XOR encryption using partial results of AES ciphers seems to be insecure, if counter reuse couldn't be avoided reliably (counter re-use is possible for current dongles, even after patches for the reported counter re-use vulnerability reported by bastille are applied).

4) This issue is part of the remaining sections in this report

Key generation

Similar to the other encryption relevant methods, the key generation code of the firmware isn't protected. Due to findings up to thi point (especially the fact of high key similarity), the algorithm has been inspected. The concern, that PLAIN data transmitted during pairing could be used to derive the link keys, has unfortunately been confirmed. Instead of a cryptographic secure key exchange algorithm, the raw pairing data (which could easily be sniffed from RF by an attacker) is used to derive keys in a very simple substitution scheme. Even more concerning, about 50 percent of the raw material used to generate the link key could be guessed by an attacker, without sniffing the pairing communication, if he has some knowledge of the devices involved in the communication.

The raw key data for a paired device is constructed like this:

```
byte 0..3:     first 4 octets of dongle RF address (could be sniffed from encrypted RF traffic or pairing
    response 1)
byte 4..5:     WPID of paired device (could be guessed by visual device inspection or sniffed from pairing
    request 1)
byte 6..7:     WPID of dongle (could be guessed by visual dongle inspection or sniffed from pairing response
    1)
byte 8..11:    device nonce (has to be sniffed from pairing request 2 or brute forced)
byte 12..15:   dongle nonce (has to be sniffed from pairing response 2 or brute forced)
```

The key is derived from this 16 byte raw keydata array, by exchanging some positions, XOR'ing 2 bytes with 0x55 and another 3 bytes with 0xff (complement).

Note: I don't include the full substitution scheme in the report for the same reason I din't include the raw AES input data template: It would pose an additional security risk if leaked and is assumed to be known by Logitech engineers.

The simple key derivation scheme confirms, what was assumed after inspection of raw device keys and there similarities: There is no complex key generation mechanism in place, even worth, there is no secure key exchange algorithm in use, to protect against attackers monitoring the RF part of the pairing process.

I, again, want to emphasize that channel hopping or changing RF addresses couldn't be considered security measures against eavesdropping pairing.

Sum up of vulnerabilities highlighted in this report

Issue 1 - Dumping of key material from dongles with outdated firmwares

An attacker could dump encryption keys of paired devices from a dongle which isn't protected by a bootloader which only accepts signed firmware updates. This could be achieved by "hot patching" a pre-analyzed firmware to leak arbitrary addresses from external RAM. A more generic way, than the one described earlier, would be to manipulate a HID++ command to accept additional parameters (16bit address and memory type: xdata/external ram/IRAM) in order to hand out arbitrary data from memory, which the MCU is able to access. Such a patch could be applied, without disrupting the dongles functionality and without overwriting existing data of paired device. Such an attack would only take a few seconds and requires one-time physical access to the dongle only.

With valid device link keys an attacker is able to decrypt keyboard traffic on the fly and inject encrypted keystrokes (cipher stream calculated based on valid counter values).

Issue 2 - Passive sniffing of key material during pairing

An attacker could sniff a valid pairing up to response of phase 2 (as described in this report) in order to re-calculate the link keys of the newly paired device. This requires no physical access to device or dongle. Various DoS style RF attacks could be applied to force a user into re-pairing of an already paired device (the device has to be unpaired in order to generate new keys).

With valid device link keys an attacker is able to decrypt keyboard traffic on the fly and inject encrypted keystrokes (cipher stream calculated based on valid counter values).

Issue 3 - Passive sniffing of key material, combined with active re-pairing

An attacker with access to a dongle could un-pair and re-pair an already paired device, till he manages to sniff the RF parts of pairing communication which are needed to calculate encryption keys (pairing phase 2). This requires physical access to both, the device and the dongle. This attack, still, is a concern as it only takes seconds if fully automated (USB HID++ 1.0 commands for pairing/unpairing are documented in public Logitech drafts). This means an attack could be implemented in drive-by-fashion or placed in supply chain (targeting pre-paired dongle and device combinations).

As the USB part of pairing and un-pairing could be automated, this could be combined with RF sniffing of pairing communication, to repeat the sniffing procedure till key material could be obtained reliably.

The following video shows a proof of concept of a manual approach (USB partpairing/unpairing is done manually). The success rate of the naive RF implementation used to generate link keys is about 50 percent (about two re-pairing attempts, till key material is sniffed): https://www.youtube.com/watch?v=1UEc8K_vwJo

The actual user of the dongle and device has no chance to recognize that the device has been re-paired to the dongle and the link encryption key is known by an attacker. This manipulation has to be done only one time.

Issue 4 - Low entropy of key material

As pointed out, the AES input data consumed during AES ECB encryption/decryption could be obtained by an attacker in several ways without physical access to dongle or device. Even worse, this part of encryption input data seems to be globally shared by all current Unifying devices.

For the per device link encryption key, 8 out of 16 bytes consist of data which could be obtained by an attacker by doing educated guesses (WPIDs for specific dongles and devices could be looked up in Open Source projects, the 4 byte of the dongle serial could be obtained by monitoring RF traffic).

In summary, the brute force space for valid key material could be reduced to 8 out of 32 byte. Success of a bruteforce attempt for a link encryption key, could be checked by an attacker based on very few sniffed encrypted RF frames, because known plaintext exists (byte 7 of decrypted payload has to be 0xc9; majority of other HID keycodes should be 0x00 in common reports).

I haven't done any comparisons between current bitcoin-hashrate and workload on AES128 ECB brute forcing, but lowering the key entropy by this huge amount could only be considered insecure.

Sum-up of mitigation measures not mentioned so far

Key exchange

The exchange of data used to derive keys has to be replaced by a secure key exchange algorithm, like existing extension of Diffie-Hellman, in order to prevent an attacker from sniffing plain RF data, which could be used for calculation of the link encryption key (passive remote attack). An attacker with valid key material, would ultimately be able to passively decrypt RF frames from link encrypted keyboard (eavesdropping) and inject arbitrary keystrokes.

Firmware protection

Already described in earlier section: - better obfuscation or asymmetric encryption of firmware blobs - trying to remove outdated firmwares from public sources . . .

Key Generation

As shown, there is no real key generation algorithm deployed. It has to be avoided, that data which could easily guessed by an attacker is used to directly derive keys, this includes: - WPIDs od devices/dongles - RF addresses - plain data exchanged during pairing

Indication of re-pairing

A user should get visual indication, if a formerly paired device was paired again (after the pairing process is secured against eavesdropping).

Replacement of XOR encryption

XOR'ing parts of the cipher text, to a plaintext partially known by an attacker (for example: all 0x00 key release reports) couldn't be considered secure and makes hardening against replay attacks nearly impossible.

A possible solution would be, to encrypt the plain HID payload with a per-frame AES key and to include all 16 cipher bytes in the resulting RF payload. This involves some re-design (16 byte cipher and 4 byte counter don't fit in current RF payload) and likely breaks compatibility with older devices (as most other mitigation measures would). The implications of XOR encryption will be described in a follow up report. A keystroke injection PoC, based on the problem, is already shown here (the attacker knows no key material in this case, dongle is patched against counter re-use):

https://www.youtube.com/watch?v=EksyCO0DzYs

Additional issues

Missing encryption for problematic RF reports

It seems only keyboard reports are encrypted. Beside the fact that an attacker could still inject unencrypted reports for SYSTEM CONTROL keys, MEDIA keys and mouse reports, outbound LED reports could leak information and be used for target enumeration.

An example would be to send a keyboard report with NUMLOCK key encoded to an RF address which the attacker wants to test to be a valid keystroke injection target (valid RF addresses could be determined based on reception of ACK payloads). If an Ack payload with an LED report is received in response, the attacker knows that the RF address reliably accepts keystroke injection (even if the actual device isn't connected to the dongle).

Another example of utilizing this, will be part of the next report.

Unintended pairing of non-Unifying Logitech devices to Unifying dongles (no security issue)

Concept already described throughout this report

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.