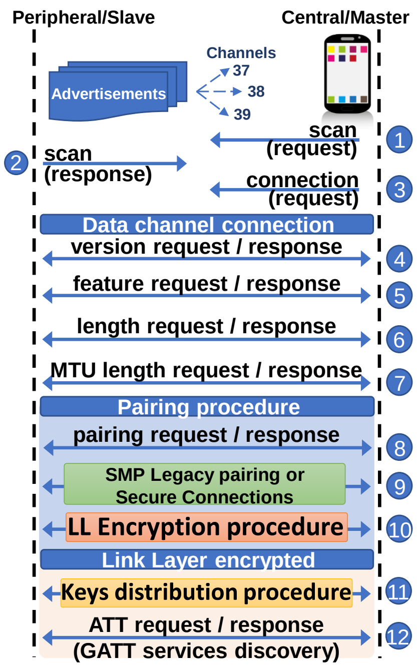
Supplemental Material for SweynTooth: Unleashing Mayhem over Bluetooth Low Energy

1 Introduction

The Bluetooth Low Energy (BLE) is a wireless communication technology specially designed to prolong battery life of devices with different power consumption and usage capabilities. BLE consists of a set of many standardized protocols that provides remote connectivity and security between a simple device (peripheral) and the user's device (central) which is usually a smartphone or a notebook. The relevant interaction between both devices is presented in Figure 1.



Peripheral/Slave
Central/Master
Advertisements
Channels
37
38
39
scan (request)
1
2
scan (response)
connection (request)
3
Data channel connection
version request / response
4
feature request / response
5
length request / response
6
MTU length request / response
7
Pairing procedure
pairing request / response
8
SMP Legacy pairing or Secure Connections
9
LL Encryption procedure
10
Link Layer encrypted
Keys distribution procedure
11
ATT request / response (GATT services discovery)
12

Figure 1: BLE messages diagram

2 Overview

A number of **12 vulnerabilities** were found across different BLE SDK of 6 major system-on-a-chip (SoC) vendors. We've followed responsive disclosure during our discovery, which allowed all SoC vendors to publicly release their respective patches already. However, many IoT products relying in the affected SoCs for BLE connectivity will still need to independently receive patches from their respective vendors if an firmware update mechanism is supported.

The vulnerabilities expose flaws in specific BLE SoC implementations that gives an attacker in radio range the ability to trigger deadlocks, crashes, buffer overflows or completely bypass security depending on the circumstances. The flaws were identified to mostly affect IoT products in appliances such as smart-home, wearables and environmental tracking or sensing. We've also identified some medical and logistics products that could be affected to some extent.

The devices affected by vulnerabilities in their respective BLE SDK are sold by major SoC vendors such as **Texas Instruments**, **NXP**, **Cypress**, **Dialog Semiconductors**, **Microchip**, **STMicroelectronics** and **Telink Semiconductor**.

A proper classification of the vulnerability set is presented in the next section.

2.1 Types of vulnerabilities

We've classified the vulnerabilities according to type and their behavior on the affected BLE devices.

- **Crash** - Vulnerabilities in this category can crash the device by triggering a hard fault due to some incorrect code behavior or memory corruption, i.e., when a buffer overflow on BLE reception buffer occurs. When a device crash occurs, they usually restart if correct hard fault handling mechanism is implemented by the product's vendor.

- **Deadlock** - Deadlocks are vulnerabilities that can affect the availability of the BLE connection without causing a hard fault or memory corruption. Usually they occur due to some de-synchronization between user code and the SDK firmware distributed by the SoC vendor, leaving the user code stuck at some point. Crashes originated from hard faults, if not properly handled, can become a deadlock if the device is not automatically restarted. In most cases, when a deadlock occurs, the user is required to manually power off and power on the device to re-establish proper BLE communication.

- **Security Bypass** - This vulnerability is the most critical one because it allows attackers in radio range to bypass the latest secure pairing mode of BLE, the Secure Connections pairing mode [**?**]. In summary, after the bypass is completed, the attacker has arbitrary read or write to device's functions that are only meant to be accessed by an authorized users.

The summary of our findings and the affected vendors is depicted in Table 1.

Table 1: Vulnerabilities type and affected vendors

Type	***Vulnerability Name***	***Affected Vendors***
Crash	Link Layer Length Overflow	Cypress NXP
	Truncated L2CAP	Dialog Semiconductors
	Silent Length Overflow	Dialog Semiconductors
	Public Key Crash	Texas Instruments
	Invalid L2CAP Fragment	Microchip
	Key Size Overflow	Telink Semiconductor
Deadlock	LLID Deadlock	Cypress NXP
	Sequential ATT Deadlock	STMicroelectronics
	Invalid Connection Request	Telink Semiconductor
Security Bypass	Zero LTK Installation	Telink Semiconductor
	DHCheck Skipping	Texas Instruments

2.2 Attacks on IoT

The exploitation of the vulnerabilities translates to dangerous attack vectors against many IoT products released in 2018-2019. At first glance, most of the vulnerabilities affects product's availability by allowing them to be remotely restarted, deadlocked or having their security bypassed. In order to raise awareness of the threats and risks of potentially vulnerable products already on the market, we've performed attacks on some representative IoT products which uses the affected SoCs as their main processor, and noted their behavior. These products are shown in Figure 2.

(a) FitBit Inspire (b) Eve Energy (c) CubiTag (d) eGee-Touch

Figure 2: An illustration of vulnerable products

- **Wearables: Fitbit Inspire** - Latest 2018 smartwatches line-up from FitBit uses the Cypress PSoC 6 as the main processor, hence they're vulnerable to *Link Layer Overflow* and *LLID Deadlock*. To the end of verifying what happens to the wearable when both issues are exploited, we've send malicious packets against the Fitbit Inspire smartwatch (Figure 2a).

  Once the malicious packets are sent to the device, it's possible to trigger either a buffer overflow in device's memory or deadlock its bluetooth stack temporarily. The former attack can immediately restarts the device whereas the latter disables its bluetooth advertisement for about 27 seconds before the smartwatch is automatically restarted by the firmware.

  In summary, the vulnerabilities only seems to temporarily block the availability of Fitbit. However, the *Link Layer Length Overflow* is a serious threat by itself as it's a potential front door to remote execution once an attacker knows the memory layout of the firmware by means of reverse engineering it. Similar behavior is expected in Fitbit Charge 3 and Ace 2 as their main processor is a Cypress PSoC 6.

- **Smart Home: Eve Systems** - Many smart home products from Eve Systems are vulnerable due to their reliance on Dialog DA14680 as the main processor. For instance, Eve Light Switch, Eve Motion MKII, Eve Aqua, Eve Thermo MKII, Eve Room, Eve Lock, etc, are all prune to *Silent Length Overflow* whereas it's possible to crash the device by sending a specific packet which overflows device's reception buffer. When an attack occurs, the user can immediately experience his smart things restarting or getting unstable. As an example, when performing the *Silent Length Overflow* against the **Eve Energy** (Figure 2b), the power on the smart plug is cut when its processor crashes and restarts. As consequence, we can momentarily cut the power of anything connected in the socket by just sending a malicious packet within radio range of the smart socket. Furthermore, an attacker can use this simple attack to cause real damage to some equipment connected to the plug by intermittently cutting its power.

  The *Silent Length Overflow* is also present on the popular **August Smart Lock**, which remotely controls access to many people's doors. Hence, it's advisable that users update their affected devices as soon as possible to avoid a worst-case exploit which could grant access control to a thief by means of remote execution.

- **Tracking Gadget: CubiTag** - The *CubiTag* Bluetooth tracker is a popular product to track user's belongings (Figure 2c). When the tracker is near an object of interest, it can be found by using a mobile app, which searches for the tracker and starts an alarm on the tracker so the user can hear and find the object. However, *CubiTag* relies on a vulnerable TI CC2640R2 SDK. Out of 2 vulnerabilities on this SDK, only the *Public Key Crash* affects availability. The CubiTag immediately stops advertising itself and is never found by the mobile app again; hence it's deadlocked. The device only works again by manually opening it with a screwdriver and re-inserting its battery so the tracker SoC properly reboot.

- **Smart Locks: eGee Touch** - The eGee Touch, shown in Figure 2d, is a smart luggage lock which can be remotely locked or unlocked through an smartphone app. During our tests, as the device uses the TI CC2540 SoC, we could lead the smart lock in a deadlock state by exploiting the *Invalid Connection Request* vulnerability. When an attack is carried, the device hangs and the user needs to manually press the *power on* button on the smart lock to re-interact with it. This is not a big problem, but the worst-case scenario occurs when the user enables continuous advertisement on the smart lock. This prevents the device to automatically restart after the attack, hence its batteries must be re-inserted to reboot its processor and restore functionalities back to normal.

2.3 Vulnerable chips

Table 2 lists the affected SoCs and the respective SDK version the vulnerabilities were found. The qualification ID of each SoC—attributed to vendors after their SDK is certified—allows to search for products using the SoC connected to such ID on the Bluetooth Listing Search site[1]. A basic search on this site yields about 479 products using the affected SoCs from Table 2.

Table 2: Vulnerabilities and SDK versions of the affected SoCs.* indicates extra affected SoCs reported by the vendor.

Silicon Vendor	BLE SoC	SDK Ver.	Qualification ID
BLE Version 5.0			
Cypress (PSoC 6)	CYBLE-416045	2.10	99158
Texas Instruments	CC2640R2	2.2.3	94079
Telink	TLSR8258	3.4.0	92269, 136037
STMicroelectronics	WB55	1.3.0	111668
STMicroelectroncis	BlueNRG-2	3.1.0	87428, 106700, 94075
Dialog	DA1469X*	3.1.0	100899
Dialog	DA14585/14586*	3.1.0	91436
BLE Version 4.2			
Cypress (PSoC 4)	CYBL11573	3.60	62243, 136808
NXP	KW41Z	2.2.1	84040
Dialog	DA14680	1.0.14.X	87407, 84084, 71309
BLE Version 4.1			
Texas Instruments	CC2540	1.5.0	23454, 127418
Dialog	DA14580	5.0.4	83573
Microchip	ATSAMB11	6.2	73346

3 Detailed description

In this section, we provide a detailed description of each vulnerability, the affected system-on-chip (SoC) models and the SDKs.

[1]https://launchstudio.bluetooth.com/Listings/Search

3.1 Link Layer Length Overflow (CVE-2019-16336, CVE-2019-17519)

The *Link Layer Length Overflow* vulnerability was identified in *Cypress PSoC4/6 BLE Component 3.41/2.60* (CVE-2019-16336) and *NXP KW41Z 3.40* SDK (CVE-2019-17519). Both implementations are susceptible to the same vulnerability. Such a vulnerability allows attackers in radio range to trigger a buffer overflow by manipulating the *LL Length Field*. The overflow occurs when the attacker, acting as the central device, sends a packet to the peripheral, which is padded to include much more bytes than expected from its type (opcode). An example is shown in Figure 3. In this case, a version request is just five bytes of length, but can be falsely extended to 247 bytes when *LL Length field* value is increased. When the underlying BLE stack processed such a packet, more bytes then the expected are allocated in memory, which caused instabilities and ultimately crashing the peripheral.

Impact: This vulnerability initially causes denial of service (DoS). However, due to its characteristic, attackers could reverse engineer products firmware to possibly leverage remote execution. A concrete evidence of this risk is exemplified by the BleedingBit vulnerability [**?**], which allowed remote execution on certain Texas Instruments devices by means of manipulating the same *LL length* field, albeit in a more constrained context. Specifically, this was only possible during advertisement mode [**?**].

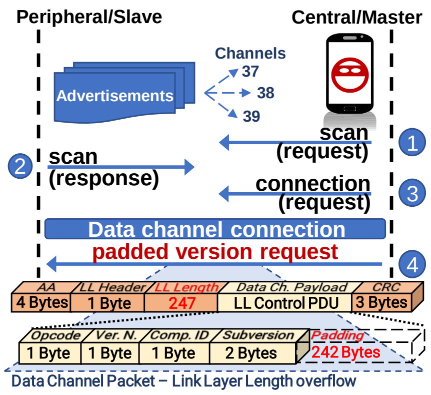


Figure 3: Link Layer Length Overflow vulnerability

3.2 Link Layer LLID deadlock (CVE-2019-17061, CVE-2019-17060)

This critical, yet simple vulnerability can render Cypress (CVE-2019-17061) and NXP devices in a *deadlock* state (CVE-2019-17060). We have discovered that if a *Cypress PSoC4/6* or *NXP KW41Z* device receives a packet with the *LLID field* cleared, then both devices simply enter in a faulty state. Specifically, this state prevents the BLE stack from working correctly in posterior connections. The details of the vulnerability are shown in Figure 4. It turns out that this attack confuses the SDK implementation in a manner that any received packet from the central is handled improperly or simply ignored. For example, NXP KW41Z peripheral may send responses completely out of order to the central. In addition, no hard faults are triggered in the firmware of the device. This, in turn, prevents auto recovery by means of simply employing watchdog timer on the product's firmware.

Impact: The availability of BLE products can be critically impaired, requiring the user to manually perform a power cycle on the product to re-establish BLE communication. Further complication arises, as the peripheral continues to advertise itself after the attack. This makes it difficult for the user to even notice the existence of the problem. The availability issue is only exposed when the user connects to the peripheral, revealing a never ending connection or pairing process.

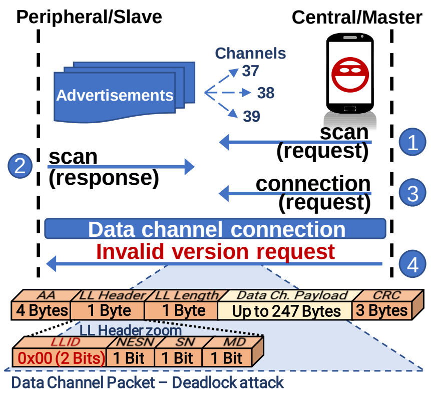


Figure 4: Link Layer LLID deadlock vulnerability

3.3 Truncated L2CAP (CVE-2019-17517)

The *Truncated L2CAP* vulnerability is present on *Dialog DA14580* devices running *SDK 5.0.4* or earlier. The flaw results due to a lack of checks during processing an L2CAP packet. If the total length of the packet (i.e. *LL Length*) has a value lower than *L2CAP Length* + 4 for a valid payload, then the truncated bytes are copied beyond the underlying reception buffer. Figure 5 shows an example of a maximum transmission unit (MTU). The MTU captures a length request, which has *LL Length* of seven bytes and *L2CAP Length* of three bytes. If the peripheral receives a malicious MTU length request with *LL Length* of five bytes instead, the L2CAP reception buffer is overflown by two bytes (i.e. *L2CAP Length* + 4 − *LL Length*). Therefore, the attacker can selectively choose the number of bytes to overflow by sending the correct L2CAP payload and the malformed *LL Length* combination to the peripheral.

Impact: An attacker in radio range can use this attack to perform denial of service and crash the device. However, a careful sequence of packets could be sent by the attacker to force the peripheral into writing certain contents to peripheral memory adjacent to the L2CAP reception buffer. In the worst-case scenario, this attack could be a front door to perform remote execution on products using *Dialog DA14580* SoC.

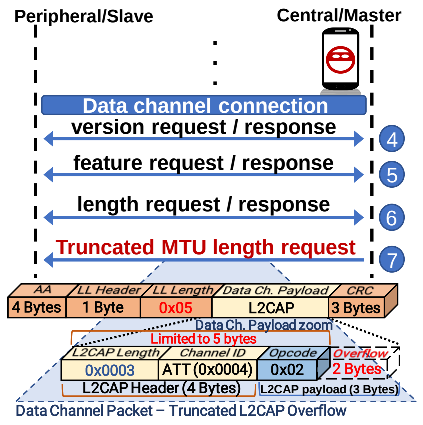


Figure 5: Truncated L2CAP vulnerability

3.4 Silent Length Overflow (CVE-2019-17518)

This attack is similar to Link *Layer Length Overflow* (cf. Section 3.1). In *Dialog DA14680* devices, it was identified that the peripheral responded to packets from the central with unexpectedly large *LL Length*. While this behaviour is not compliant to BLE Core specifications [?], it does not appear to affect the peripheral at first. Nonetheless, when a certain packet payload with higher than expected *LL Length* is sent, the peripheral crashes. This indicates that an overflow on the reception buffer had occurred for certain packet types such as pairing request.

Impact: An attacker in radio range can mostly use this attack to perform denial of service and crash the device. Given that a buffer overflow is being triggered depending on the packet, a remote execution scenario is a possibility. Furthermore, the SoCs affected by this vulnerability are known to be used in a large number of smart home products, which increases the reachability and risk of a more serious exploit of this vulnerability.

3.5 Invalid Connection Request (CVE-2019-19195)

We identified that *Texas Instruments CC2640R2* and *CC2540* SDKs do not properly validate some connection parameters when the central attempts a connection to the peripheral. Instead, the peripheral creates a connection with the central, but fail at later stage and moves the peripheral state to idle (i.e., stops advertisement). If the idle state is not handed correctly in the product's code, the device does not go back to advertisement again.

During the initial phase of a BLE connection, the central device scans for advertisements packets from the peripheral and sends a connection request packet which contains relevant parameters such as connection *interval* and *timeout*. These two parameters control the cadence of packet exchange and timeout between peripheral and the central device, respectively. Their values must represent a non-zero time period in *milliseconds* when multiplied by a factor of **1.25**. However, if the central device sends an invalid connection request with the fields *interval* or *timeout* as zero, then the peripheral stops advertisement and enters in idle state. This state is not the flaw alone, as idle state is a common state in BLE. However, we've discovered that the state change under reception of invalid parameters is not well documented in TI SDK and sample codes, which can lead product developers to not handle such scenario. The mishandling of this state can lead products such as eGeeLock to stop advertisement and hence requiring user intervention. An illustration of this attack is given in Figure 6.

Interestingly, *CC2540* also accepts connection requests with packet length lower than expected (truncated), which triggers the same behaviour (i.e. Figure 6) due to its implementation assuming zero bytes for truncated fields.

Impact: An attacker in radio range can exploit the phenomenon observed in Figure 6 to easily perform DoS in products using the vulnerable SoCs. Furthermore, if the product vendor does not implement mechanisms to detect such a faulty behaviour, the device can be driven into a deadlock state. This, in turn, requires the user to manually restart the device. In general, the impact on such affected products is similar to the *LLID Deadlock* (cf. Section 3.2).

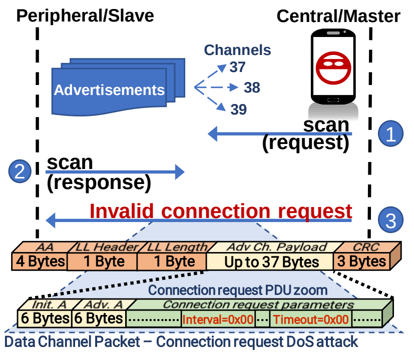


Figure 6: Invalid Connection Deadlock

(CVE-2019-19194)

3.6 Unexpected Public Key Crash (CVE-2019-17520)

This vulnerability was found on Texas Instruments *CC2640R2* SDK. Specifically, the vulnerability is present in the implementation of legacy pairing procedure. When the peripheral performs the legacy pairing procedure, it is possible to cause a hard fault in device's memory by sending an SMP public key packet before the SMP pairing procedure starts (Step 9 in Figure 1). Normally, the peripheral should ignore the reception of a public key if secure connection is not enabled in the *pairing request/response* exchange. During our coordination with the vendor, Texas Instruments informed us that the hard fault is triggered because the peripheral accepts the public key and tries to copy it to a null target address. Normally, this address corresponds to a valid allocated buffer if secure connection is properly indicated during the *pairing request/response* process. We illustrate the vulnerability in Figure 7.

Impact: An attacker in radio range can exploit the aforementioned behaviour to perform DoS and possibly restart products using the *CC2640R2* SoC for the main application. On the bright side, it is not possible to perform a buffer overflow in the peripheral's memory. This is because the unexpected public key is always copied to a null address, which is beyond the control of the attacker. It is worthwhile to mention that this vulnerability can also lead to a deadlock. This is exemplified by our evaluation on the *CubiTag bluetooth tracker*. The product CubiTag does not properly handle hard faults and hence enters a deadlock state. This requires the user of the tracker to manually restart it.

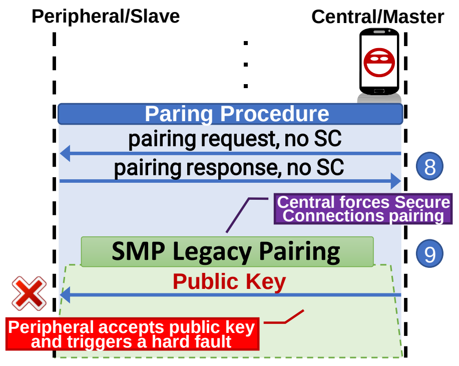


Figure 7: Unexpected Public Key Crash ("SC" means secure connection)

3.7 Sequential ATT Deadlock (CVE-2019-19192)

In *STMicroelectronics WB55 SDK V1.3.0* and earlier, it is possible to deadlock the peripheral by sending just two consecutive ATT request packets in each connection event. Normally, each ATT request sent from a central device is followed by an ATT response from the peripheral. This happens in a time period multiple of the connection interval Δt. However, it is possible that a rogue central device sends multiple and consecutive ATT requests separated by the connection interval Δt (Figure 8). In such a case, the peripheral does not get sufficient time to respond to the first ATT request. The resulting behaviour is a fault in the coprocessor that runs the BLE SDK inside WB55, preventing certain BLE event flags to be cleared. This leads to a deadlock of the WB55 user code. Specifically, the faulty code potentially gets stuck in a `while` loop, which waits for a never finishing BLE event.

Impact: Similar to several other vulnerabilities discussed in this work, the exploitation of this vulnerability can leave the product in a deadlock state if a stability mechanism, such as the watchdog timer, is not employed by the vendor in product's firmware. If such a mechanism is present, the attack is still guaranteed to remotely restart the device.

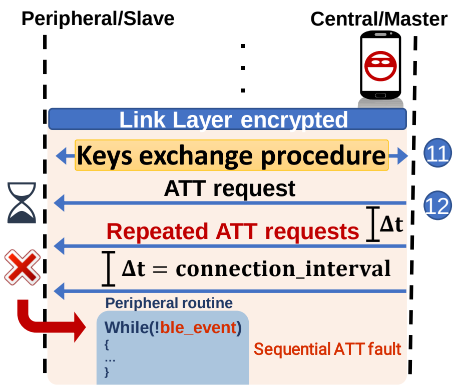


Figure 8: Sequential ATT Deadlock

3.8 Invalid L2CAP fragment (CVE-2019-19195)

During the communication between the central device and peripheral, the Bluetooth 4.0-4.2 Core specification dictates that the minimum and maximum PDU size of the packets should be in the range 2-39 [?]. Packets outside this boundary should be discarded as they are invalid. However, we discovered that such is not the case for devices running *Microchip ATMSAMB11 BluSDK Smart v6.2* and earlier. Following Figure 9, it was discovered that if an L2CAP PDU of length one is sent to the peripheral, then the device simply crashes.

Impact: The watchdog mechanism is enabled by default in the SDK, which reduces the risk for products relying on *ATSAMB11* BLE solution to exhibit deadlock behaviour. Therefore, this vulnerability mostly affects the availability of the device by remotely restarting them.

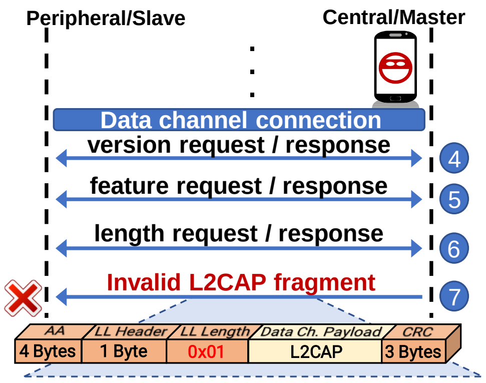


Figure 9: Invalid L2CAP fragment

3.9 Key Size Overflow (CVE-2019-19196)

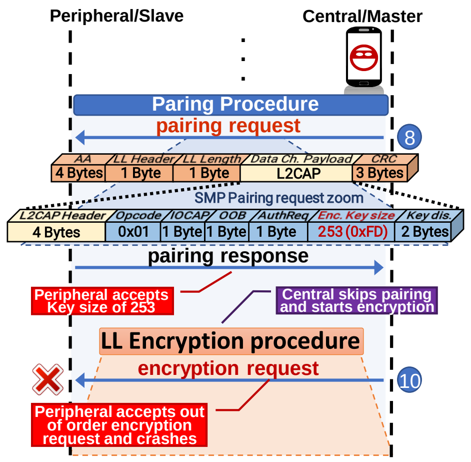


Figure 10: Key Size Overflow

The Key Size Overflow vulnerability was found in all *Telink Semiconductor* BLE SDKs. This causes an overflow in the device memory, resulting in a crash. However, the problem that allows this vulnerability is a combination of multiple issue found during the pairing procedure of devices using Telink SMP implementation.

During the start of the pairing procedure, the central device sends a *Pairing request* packet containing the maximum allowed key size to be negotiated at the end of the pairing procedure. The maximum key size is standardised to be within 7 to 16 bytes and any deviation from that should be rejected with a *pairing failure* response. However, Telink peripheral actually accepts a *maximum encryption key size* up to 255 by answering the central device with a *paring response* instead. Despite this first problem, the peripheral rejects the pairing during at later exchanges of the pairing procedure without abnormal behaviour. The second and final problem that finally triggers the vulnerability, arises because the peripheral accepts the *LL Encryption procedure* to occur before the pairing procedure even starts, albeit failing at later stage.

By combining the two mentioned problems it's possible to force the peripheral into allocating the over sized key buffer length which was negotiated during the *pairing request*. Depicted in Figure 10, the central device sends the invalid *pairing request*, waits for *pairing response* and sends an *encryption request*. The request is accepted and a buffer overflow occurs in peripheral's memory as its firmware tries to allocate the oversized key.

Impact: This vulnerability allows an attacker in radio to perform buffer overflow and crash Telink SoCs products with pairing support enabled, which is a common practice in several BLE products. While this vulnerability doesn't expose easy control over the overflown buffer, an exploit could be carefully constructed to overwrite memory contents adjacent to the key buffer if the the memory layout of the product's firmware is known. In the worst case, it could be possible to overwrite buffers that store encryption nonce would allow the attacker to bypass encryption and leak user information.

3.10 Zero LTK Installation (CVE-2019-19194)

This critical vulnerability is a variation of one of the Key Size Overflow. It affects all products using Telink SMP implementation with support for secure connection enabled. It was verified that when the Telink peripheral accepts an out of order encryption request from the central, the encryption procedure is successful with a LTK which is zero. The LTK size, usually of 16 bytes, is agreed during the *pairing request/response* exchange.

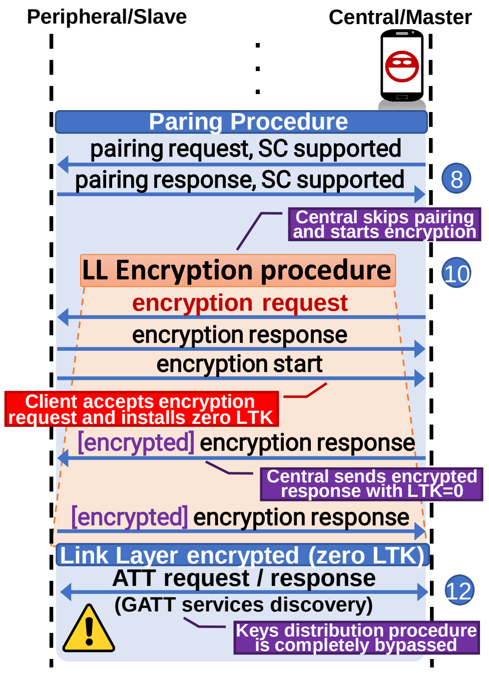

Peripheral/Slave
Central/Master
Paring Procedure
pairing request, SC supported
pairing response, SC supported
8
Central skips pairing and starts encryption
LL Encryption procedure
10
encryption request
encryption response
encryption start
Client accepts encryption request and installs zero LTK
[encrypted] encryption response
Central sends encrypted response with LTK=0
[encrypted] encryption response
Link Layer encrypted (zero LTK)
ATT request / response
(GATT services discovery)
12
Keys distribution procedure is completely bypassed

Figure 11: Zero LTK Installation

Following Figure 9, the rogue central sends a *pairing request* with secure connections pairing indicated and waits for a *pairing response*. Next, the central skips the secure connections pairing procedure and starts the encryption procedure by sending a *encryption request*. Due to the lack of validation in peripheral's implementation, the central receives a *encryption start* from the peripheral and sends an encrypted *encryption response* back. This response is relevant because the peripheral validates it against the session key *SK*, which is derived from a valid LTK. Interestingly, the peripheral's LTK is initialized as zero. This allows the central to easily derive the *SK* to send the correct encrypted *encryption response* to the peripheral, hence successfully completing the encryption procedure.

The *SK* (hiddenly mentioned in Bluetooth Core Specification as *sessionKey* [?]) is generated by the cryptographic function of Equation 1.

$$SK = AES_{ECB}(Key = \mathbf{LTK}, Plaintext = \mathbf{SKD}) \qquad (1)$$

The Session Key Diversifier (*SKD*) is a random 16 byte number obtained via the *encryption request/response* exchange, therefore, guessing the correct LTK allows the central device to send a encrypted *encryption response* with a valid *SK*. As an aggravating factor, the *Keys distribution procedure* is completely bypassed after the attack is performed.

Impact: An attacker in Radio range can abuse this vulnerability to completely bypass security in a BLE products which rely in secure connections pairing to protect user privacy. Furthermore, device's functionalities which were only allowed to be accessed by an authorized user, can be trivially bypassed. In short, this vulnerability allows an attacker full communication control over a protected BLE application.

As a side note, this vulnerability only affects Telink devices that allows secure connection pairing. In reality, affected products that disable secure connections pairing (the currently secure BLE pairing mode) and enable only the insecure legacy pairing mode, are in fact more secure due to this vulnerability.

3.11 DHCheck Skipping

This vulnerability is similar to Zero LTK installation, however, the vulnerability relies in bypassing the last message exchanged during the Secure Connection pairing, the DHCheck from peripheral and central. The objective of DHCheck is to perform a final verification on the nonces exchanged during the pairing process (i.e., Just Works or PIN) by using each device's private key. This way the central and peripheral can verify the DHCheck, based on each other public key and confirm that the exchange was indeed complete.

The flaw found in Texas Instrument CC2640R2 SMP implementation and allows the DHCheck to be skipped by simply starting the `LL Encryption Procedure` earlier (c.f. step 10 of Figure 1. In this case, the peripheral install the LTK being generated in the current unfinished pairing process and allows the central to perform encryption of the link with such LTK while also skipping the Key distribution procedure (c.f. step 11 in Figure 1). It's worthwhile to mention that, in Secure Connection Pairing, the LTK is actually generated before the DHCheck is complete, however such key is normally discarded if something goes wrong during the DHCheck for security reasons.

While the attacker can skip the DHCheck during the pairing process, he still needs the private key associated with the public key exchanged in the beginning of the pairing procedure to generate the LTK of the current pairing session, however as DHCheck is skipped, the random nonces, and peripheral/central addresses are not verified by the peripheral.

Impact: An attacker in radio range can launch a man-in-the middle attack and use this vulnerability to modify the random nonce sent from the legitimate central and force the peripheral to install a fake LTK by starting the `LL_encryption_procedure` before the DHCheck.

References