SweynTooth: Unleashing Mayhem over Bluetooth Low Energy

Matheus E. Garbelini[1]; Sudipta Chattopadhyay[1]; Chundong Wang[1]
[1]*Singapore University of Technology and Design*

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 provide 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.

In this public disclosure, we release the technical details of SweynTooth vulnerabilities. We note that SweynTooth vulnerabilities are released in different batches to respect the responsible disclosure timeline. As of today, we have released 12 new vulnerabilities in the first batch of SweynTooth (released 11th February, 2020) whereas five new vulnerabilities are released in the second batch (released 14th July, 2020).

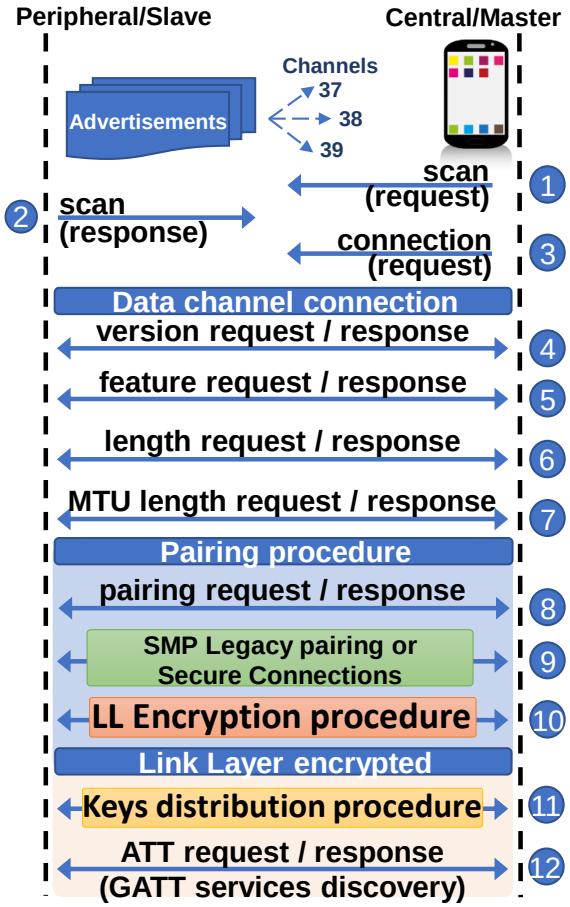

Figure 1: BLE messages exchange diagram

1 Introducing SweynTooth

In the first batch, SweynTooth captures a family of **12 vulnerabilities** across different BLE software development kits (SDKs) of seven major system-on-a-chip (SoC) vendors. The vulnerabilities expose flaws in specific BLE SoC implementations that allow an attacker in radio range to trigger **deadlocks, crashes** and **buffer overflows** or **completely bypass security** depending on the circumstances.

SweynTooth potentially affects IoT products in appliances such as smart-homes, wearables and environmental tracking or sensing. We have also identified several medical and logistics products that could be affected.

For Batch 1, SweynTooth vulnerabilities are found in the BLE SDKs sold by major SoC vendors, such as **Texas Instruments**, **NXP**, **Cypress**, **Dialog Semiconductors**, **Microchip**, **STMicroelectronics** and **Telink Semiconductor**. *By no means, this list of SoC vendors is exhaustive in terms of being affected by* SweynTooth. We have followed responsive disclosure during our discovery, which allowed almost all SoC vendors to publicly release their respective patches already. However, a substantial number of IoT products relying on the affected SoCs for BLE connectivity will still need to independently receive patches from their respective vendors, as long as a firmware update mechanism is supported by the vendor.

1.1 SweynTooth (Batch 2)

In the second batch, SweynTooth captures a family of **five vulnerabilities** across different BLE software development kits (SDKs). The vulnerabilities expose flaws that allow an attacker in radio range to trigger **deadlocks, crashes** or **partially bypass security** depending on the circumstances. For the second batch, SweynTooth vulnerabilities are found in the BLE SDKs sold by major SoC vendors and open-source projects such as **Texas Instruments**, **Espressif**, **Microchip** and **Zephyr Bluetooth Stack**.

Unlike the first batch of SweynTooth vulnerabilities, we did not perform a comprehensive survey of the IoT products affected by the aforementioned affected BLE stacks. IoT product manufacturers are therefore strongly advised to check if their product is using any of the affected BLE stack.

SweynTooth highlights concrete flaws in the BLE stack certification process. We envision substantial amendments to the BLE stack certification to avoid SweynTooth style security flaws. We also urge SoC vendors and IoT product manufacturers to be aware of such security issues and to

initiate focused effort in security testing.

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

1.2 Types of vulnerabilities

We have classified the SWEYNTOOTH vulnerabilities according to their types and their behaviours on the affected BLE devices.

- **Crash**: Vulnerabilities in this category can remotely crash a device by triggering hard faults. This happens due to some incorrect code behaviour or memory corruption, e.g., when a buffer overflow on BLE reception buffer occurs. When a device crash occurs, they usually restart. However, such a restart capability depends on whether a correct hard fault handling mechanism was implemented in the product that uses the vulnerable BLE SoC.
- **Deadlock**: Deadlocks are vulnerabilities that affect the availability of the BLE connection without causing a hard fault or memory corruption. Usually they occur due to some improper synchronisation between user code and the SDK firmware distributed by the SoC vendor, leaving the user code being 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. This is because the vulnerability allows attackers in radio range to bypass the latest secure pairing mode of BLE, i.e., the *Secure Connections* pairing mode [16]. In summary, after the bypass is completed, an attacker in the radio range has arbitrary read or write access to device's functions. These functions, in turn, are only meant to be accessed by authorised users.

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

2 Vulnerable BLE chips

Table 2 lists the affected SoCs and the respective SDK versions where 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 [18]. A basic search on this site yields about **480 products listings using the affected SoCs** from Table 2. Each listing may contain multiple products from the same vendor, which further increases the total number of different products affected. This does not mean, however, that all products are guaranteed to be affected. This is because the impact of SWEYNTOOTH vulnerabilities depends on how the product software handles BLE communication and how much it relies on affected SoCs to operate.

Table 1: Vulnerabilities type and affected vendors.* indicates the vulnerability which exhibited a different behaviour on other vendor.

Type	*Vulnerability Name*	*Affected Vendors*	*CVE*
Crash	Link Layer Length Overflow	Cypress NXP	CVE-2019-16336 (6.1) CVE-2019-17519 (6.1)
	Truncated L2CAP	Dialog Semiconductors	CVE-2019-17517 (6.3)
	Silent Length Overflow	Dialog Semiconductors	CVE-2019-17518 (6.4)
	Public Key Crash	Texas Instruments	CVE-2019-17520 (6.6)
	Invalid L2CAP Fragment	Microchip	CVE-2019-19195 (6.8)
	Key Size Overflow	Telink Semiconductor	CVE-2019-19196 (6.9)
	Invalid Sequence Memory Corruption	Zephyr Project	CVE-2020-10061 (6.13)
	Invalid Channel Map	Zephyr Project Espressif Systems	CVE-2019-19196 (6.14) CVE-2020-13594 (6.14)
Deadlock	LLID Deadlock	Cypress NXP	CVE-2019-17061 (6.2) CVE-2019-17060 (6.2)
	Sequential ATT Deadlock	STMicroelectronics	CVE-2019-19192 (6.7)
	Invalid Connection Request	Texas Instruments	CVE-2019-19193 (6.5)
	HCI Desync	Espressif Systems	CVE-2020-13595 (6.12)
	Invalid Channel Map*	Microchip	CVE-2020-13594 (6.14)
Security Bypass	Zero LTK Installation	Telink Semiconductor	CVE-2019-19194 (6.10)
	DHCheck Skip	Texas Instruments	CVE-2020-13593 (6.11)

Table 2: Vulnerabilities and SDK versions of the affected SoCs.* indicates extra affected SoCs reported by the vendor not tracked by our team.

Vuln.	*SoC Vendor*	*SoC Model*	*SDK Ver.*	*Qualification ID(s)*
	BLE Version 5.0/5.1			
6.13,6.14	Nordic Semiconductor (Zephyr Project Stack)	nRF51/52	2.2.0	135679, 101395
6.12,6.14	Espressif Systems	ESP32	4.2	103833, 147845, 116661, 144495, **many**
6.1,6.2	Cypress (PSoC 6)	CYBLE-416045	2.10	99158
6.5,6.6,6.11	Texas Instruments	CC2640R2	3.30.00.20	94079
6.9,6.10	Telink	TLSR8258	3.4.0	92269, 136037
6.7	STMicroelectronics	WB55	1.3.0	111668
6.7	STMicroelectroncis	BlueNRG-2	3.1.0	87428, 106700, 94075
6.4	Dialog	DA1469X*	10.0.6	100899
6.3	Dialog	DA14585/6*	6.0.12.1020	91436
	BLE Version 4.2			
6.1,6.2	Cypress (PSoC 4)	CYBL11573	3.60	62243, 136808, 79697, 82951, 79480
6.1,6.2	NXP	KW41Z	2.2.1	84040
6.4	Dialog	DA14680	1.0.14.X	87407, 84084, 71309, 75255
	BLE Version 4.1			
6.5	Texas Instruments	CC2540	1.5.0	23454, 127418
6.3	Dialog	DA14580	5.0.4	83573
6.8,6.14	Microchip	ATSAMB11	6.2	73346

2.1 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 affect 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 have performed attacks on five representative IoT products which use the affected SoCs (cf. Table 2) as their main processor. These products are shown in Figure 2.

- **Wearables: Fitbit Inspire** - Latest 2018 smartwatches line-up from FitBit uses the Cypress PSoC 6 as the main processor. Hence they are vulnerable to *Link Layer Overflow* and *LLID Deadlock*. To verify what happens to the wearable when both issues are exploited, we have

Table 3: Products verified to be vulnerable

Product	Category	BLE SoC	Vulnerability	Impact
Eve Energy **August Smart Lock**	Smart Home Smart Home	**DA14680**	**(6.4) Silent Length Overflow**	Crash
Fitbit Inspire	Wearables	CY8C68237	(6.1) LL Length Overflow (6.2) LLID Deadlock	Crash
CubiTag	Gadget Tracking	CC2640R2	(6.6) Public Key Crash	Deadlock
eGeeTouch TSA Lock	Security	CC2540	(6.5) Invalid Connection Request	Deadlock

(a) FitBit Inspire (b) Eve Energy (c) August Smart Lock

(d) CubiTag (e) eGeeTouch

Figure 2: An illustration of some vulnerable products.

sent malicious packets to the Fitbit Inspire smartwatch (Figure 2a) through the BLE communication channel.

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

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

- **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 prone to *Silent Length Overflow*. Specifically, it is possible to crash such devices by sending a specific packet which overflows device's reception buffer. When an attack occurs, the user can immediately experience her smart products 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 off when its processor crashes and restarts. As a consequence, we can momentarily cut the power to 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.

  August Smart Lock - We have verified that the *Silent Length Overflow* also affects the popular **August Smart Lock** (Figure 2c). Such a smart lock remotely controls access to house doors. Hence, it is advisable that users update their affected devices as soon as possible. This is 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 that tracks the belongings of a user (Figure 2d). 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. As a result, the user can hear the alarm and find the object. However, *CubiTag* relies on a vulnerable TI CC2640R2 SDK. Out of two vulnerabilities on this SDK, only the *Public Key Crash* affects availability of CubiTag. The CubiTag immediately stops advertising itself and is never found by the mobile app again; hence it is deadlocked. *The CubiTag device only works again by manually opening it with a screwdriver and re-inserting its battery*. This is for the tracker SoC to reboot properly and to establish normal BLE connection.

- **Smart Locks: eGee Touch** - The eGee Touch, shown in Figure 2e, is a smart luggage lock that can be remotely locked or unlocked through a 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 out, 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 critical problem by itself. However, 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.

2.2 Other potentially vulnerable products

As mentioned in Section 2, the vulnerabilities discovered by our team are likely to affect a substantial number of products

that rely on the affected SoCs. While it is difficult to confirm the reach of such vulnerabilities for every product out in the wild, we provide an overview of the types of products potentially affected by SweynTooth using the Bluetooth Listing Search site [18]. Figure 3 captures the total number of products listings using the affected SoCs as of 8th February, 2020.

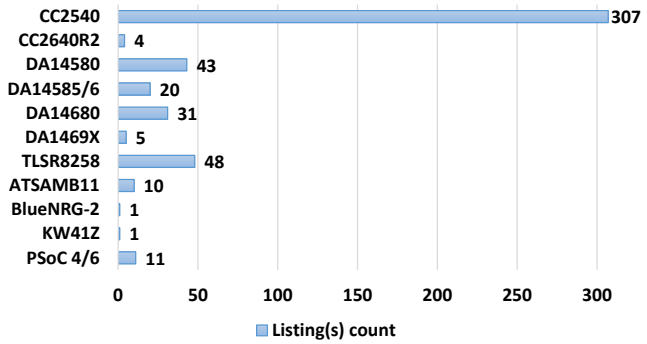

Figure 3: Number of product listings for each SoC affected as of February 8, 2020.

While the majority of products are listed under CC2540, we note several critical BLE products using the other affected SoC vendors. These products are applied to **logistics, medical, consumer electronics, smart home, wearables** and other fields. Although it is impractical for our team to verify such a large number of products using the affected SoCs, we outline some of the most notorious or popular products found on Bluetooth Listing website [18]. These critical products are shown in Table 4.

The most critical devices that could be severely impacted by SweynTooth are the medical products. VivaCheck Laboratories, which manufacture Blood Glucose Meters, has many products listed to use DA14580. Hence all these products are potentially vulnerable to the Truncated L2CAP attack. Even worse, the latest pacemaker related products from Medtronic Inc. are potentially affected. While our team did not verify the extent to which SweynTooth affects such devices (e.g. the impact of remotely restarting such devices or remote code execution in the worst case), it is highly recommended that such companies update their firmware. This is to avoid any situation that could pose life threatening risks to the patients using the respective medical products. Unfortunately, the security issue found in Dialog DA14580 is still unpatched (cf. Table 5). Nonetheless, Dialog is working on an internal security patch for DA14580 and will release it for general public in the next SDK release. More details about patches are mentioned in Section 3.

Another complication arises for the IN180-13 from Swipbox International. This product uses the vulnerable KW41Z SoC from the vendor NXP semiconductor. According to the company page [21], the battery operated product is an automatic parcel locker and has no display. Instead, it relies entirely on a smartphone application communicating via BLE to unlock the parcel sent to the recipient. Similarly to what happens to CubiTag, the KW41Z is vulnerable to a deadlock (LLID Deadlock). The KW41Z LLID deadlock vulnerability is particularly easy to trigger and allows an attacker in radio range to simply block anyone to connect to the parcel locker (unless the the parcel locker is automatically restarted). Fortunately, NXP has already released patches for the two vulnerabilities affecting KW41Z. Other potentially vulnerable popular products include vendors such as August Home, Eve Systems, Samsung and Anhui Huami Information Technology, among others.

It is worthwhile to note that the list in Table 4 is not exhaustive. Thus, we recommend each product vendor to update the SDK firmware of their products to the latest if available or contact their SoC vendor to enquire on the status of the patch.

3 Security Patches

Update 14/07/2020: Zephyr Project, Espressif Systems and Texas Instruments have updated their BLE stack with security fixes [8, 12, 22].
Update 24/04/2020: STMicroelectronics has updated their latest WB55 and BlueNRG-1/2 SDKs [19, 20].
Update 17/03/2020: Table 5 has been updated. Dialog Semiconductors has released hotfixes for DA14680/1/2/3 and DA14585/6 SoCs. Patches for DA14580/1/3 are planned for the end of March. You can read more information in their advisory page [13]. Furthermore, Microchip has kindly self disclosed more devices to be affected by multiple SweynTooth vulnerabilities [23].

Most of the affected vendors have released patches for their respective SoCs. One can get the latest patches by downloading the newest SDK of each vendor referenced in Table 5. Product vendors (who use the affected SoCs), on the other hand, are being independently contacted by each SoC vendor to inform about the security patches. However, we note that some SoCs did not receive a patch from their vendor yet. This is the case for Dialog, Microchip and STMicroelectroncs. We will be updating this section once vendors release the security patches for the affected SoCs.

We urge action from vendors due to the reliance of the BLE IoT market on such unpatched SoCs. For example, **August Home Inc** and **Eve Systems** products rely almost entirely on **DA14680**, which is still *unpatched* even after a responsive disclosure period of more than 90 days.

During our contact with Dialog, they have confirmed that a patch is planned in the next SDK release for the affected SoCs. We were also informed that the reason of such delay is due to the affected code being stored in the read-only-memory (ROM) of such SoCs. Thus, the respective vulnerable BLE stack cannot be modified and it requires complex workarounds for publicly releasing a patch.

Table 4: Some notorious products using SoCs affected by SweynTooth. The declaration ID references each product on the Bluetooth Listing Search [18]. *Syqe Medical Ltd. has clarified that they did buy a BLE license for their product *Syqe Inhaler v01*, but they are not using the BLE technology.

Product Vendor	*Product Model*	*SoC Model*	*Category*	*Declaration ID*
Swipbox International A/S	**IN180-series**	**KW41Z/31Z**	**Logistics**	**D043799**
VivaChek Labortatories, Inc.	**Blood Glucose Meter (many more)**	**DA14580**	**Medical**	**D049092**
Medtronic Inc.	**Azure XT DR MRI (many more)**	**DA14580**	**Medical**	**D047941**
Syqe Medical Ltd*	Syqe Inhaler v01	DA14580	Medical	D046866
Anhui Huami Info. Tech. Co., Ltd.	Amazfit Bip Lite, A1915	DA14580	Wearables	D047941
Anhui Huami Info. Tech. Co., Ltd.	Mi Band 3 NFC AMAZIFT Band 2 (few more)	DA14680	Wearables	D038915
Anhui Huami Info. Tech. Co., Ltd.	Amazfit Verge Lite Amazfit Health Watch (few more)	DA14585/6	Wearables	D043218
August Home Inc	Yale Linus Smart Lock, ASL-06 (many more)	DA14680	Smart Home	D049011
Eve Systems GmbH	Eve Light Switch Eve Window Guard (many more)	DA14680	Smart Home	D035053
Samsung Electronics Co., Ltd.	Galaxy Fit, SM-R370	DA1469X	Wearables	D044091
Fitbit, Inc.	Charge 3 Ace 2 (few more)	PSoC 6	Wearables	D039792
RESIDEO TECHNOLOGIES, INC.	Buoy Device	ATSAMB11	Indoor/Outdoor Monitoring	D029728
Xiaomi Inc.	Mi Wireless Mouse Mi Silent Mouse	TLSR 82xx	Consumer Electronics	D046746
Samsung Electronics Co., Ltd.	Smart Lightining Module, SLM-A-BD4	TLSR 82xx	Modules	D042305
Qingping Technology (Beijing)	Qingping Bluetooth Alarm Clock	TLSR 82xx	Smart Home	D048743

Table 5: Patches available by SoC vendors **(14/07/2020)**.

SoC Vendor	*SoC Model*	*Vendor Patches*
Zephyr Project	nRF51/52	[12] Zephyr v2.3.0 and backports
Espressif Systems	ESP32	[22] Latest ESP32 BT/BLE Stack Libraries
Cypress (PSoC 6)	CYBLE-416045	[5] BLE_PDL 2.2
Cypress (PSoC 4)	CYBL11573	[4] BLE Component 3.63
NXP	KW41Z/31Z	[9] 2.2.1 (2019-11-28)
NXP	KW37/8/9	[10] 2.6.2 (2019-12-20)
NXP	KW34/5/6	[10] 2.2.2 (2019-12-06)
Texas Instruments	CC2640R2	[8] v3.40.00.10
Texas Instruments	CC2640/50	[8] v2.2.4
Texas Instruments	CC13X2/26X2	[8] v3.40.00.02
Texas Instruments	CC13x0	[8] v4.10.xx
Texas Instruments	CC2540/1	[7] v1.5.1
Telink	TLSR8258	[24] v3.4.0 (SMP fix)
Telink	TLSR8232	[26] v1.3.0 (SMP fix)
Telink	TLSR826x	[25] v3.3 (SMP fix)
Dialog	DA1469X	[13] 10.0.6
Dialog	DA14585/6	[13] **Hotfix available**
Dialog	DA14680/1/2/3	[13] **Hotfix available**
Dialog	DA14580/1/3	[13] **Hotfix available**
Microchip	ATSAMB11	[23] **Pending**
Microchip	WINC3400	[23] **Pending**
Microchip	IS1870/1	[23] **Pending**
Microchip	BM70/1	[23] **Pending**
Microchip	RN4870/1	[23] **Pending**
Microchip	BTLC1000	[23] **Pending**
Microchip	IS1677/8	[23] **Pending**
Microchip	BM77/8	[23] **Pending**
Microchip	RN4677/8	[23] **Pending**
Microchip	IS2062/3/4/6	[23] **Pending**
Microchip	BM62/3/4	[23] **Pending**
Microchip	IS2083	[23] **Pending**
Microchip	BM83	[23] **Pending**
STMicroelectronics	WB55	[19] v1.6.0
STMicroelectroncis	BlueNRG-1/2	[20] v3.2.0

4 Non-compliance in the wild!

In recent years, Bluetooth design has been under scrutiny due to several security flaws such as KNOB [1], BlueBorne [14] and Invalid ECC Attack [2]. In contrast, little to no research has been carried out on the security of the diverse Bluetooth implementations out in the wild. The current practice is to leave the implementation tests to the Bluetooth certification process. This is with the mindset that once the design is sound, hardly anything can break in the implementation of the Bluetooth stack.

Our findings expose some fundamental attack vectors against certified and recertified BLE Stacks which are supposed to be “safe" against such flaws. We carefully investigated the reasons that might explain the presence of SweynTooth vulnerabilities on the affected SoCs. We believe this is due to the imposed isolation between the link layer and other Bluetooth protocols, via the Host Controller Interface (HCI) protocol [16]. While such a strategy is reasonable for hardware compatibility, this adds complexity to the implementation. Moreover, it overly complicates the strategies to systematically and comprehensively test Bluetooth protocols. Specifically, during testing, it is complex to send arbitrary Link Layer messages during other protocol message exchanges. Such added complexity is likely the reason for inadequate

security testing of BLE stack implementation.

We carefully read and investigated relevant parts of the Controller and Host volume of the Core Specification, which allow us to understand the main interaction of two devices (c.f., Figure 1). A natural question that arises for anyone looking a sequence diagram of the overall pairing procedure is "*what happens if the LL encryption starts in the middle of the pairing procedure?*". It can be argued that the Zero LTK installation, Key size overflow and Public Key Crash flaws were facilitated due to this question not being answered on the Core Specification itself, leaving it for vendors to decide how to handle such situation. When attempting to answer this question, we have received highly different behaviours across most SoCs we have tested. In addition, several other implementation details, which are explicitly imposed by the Core Specification [17], are also not followed by SoC vendors in reality.

It is worthwhile to mention that every SoC BLE SDK goes through the certification process before going into market. Thus, our findings expose that improvements should be made on the certification process to avoid at least *simple* deviations such as the LLID deadlock: *It takes exactly one field to be zero to lead the device into a deadlock*. Furthermore, devices from Telink responds to version requests multiple times, going against **[Vol 6] Part B, Section 5.1.5** of Core Specification [17], which defines that the peripheral should only respond a version request a single time during the same central-peripheral connection. Similarly, all devices we have tested accept a connection request with the"hopIncrement" field value of less than five. This behaviour goes against **[Vol 6] Part B, Section 2.3.3.1**, which dictates the valid range of such field to be within 5-16. Moreover, all the vulnerabilities we have discovered go against **[Vol 1] Part E, Section 2.7** (Responding to malformed behaviour). This part of the specification provides directives and few examples to handle invalid or malformed packets.

In conclusion, we strongly believe that the Bluetooth SIG should improve and significantly expand Section 2.7 (the section is less than one page!!!) and add more basic tests to the Bluetooth certification to avoid zero-day vulnerabilities such as the ones captured by SweynTooth.

5 Why SweynTooth?

The insight behind the name SweynTooth arrives from Sweyn Forkbeard, the son of King Harald Bluetooth (after whom the Bluetooth Technology was originally named). Sweyn revolted against Harald Bluetooth and this forced King Harald to his exile. The exile lead to the death of King Harald shortly. We envision that if SweynTooth style vulnerabilities are not appropriately handled by BLE vendors, then the technology can become a breeding ground for attackers. This may, in turn, lead the Bluetooth technology to be obsolete.

The SweynTooth logo is designed based on the combination of letter "S" (abbreviating Sweyn) and letter "T" (abbreviating Tooth) from the Elder Futhark alphabet – one of the oldest Runic alphabets.

6 Detailed Description

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

6.1 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 4. 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 [6], 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, BleedingBit exploited the central implementation of the SoC during the advertisement phase.

6.2 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 5. It turns out that this attack confuses the SDK implementation in a manner that any

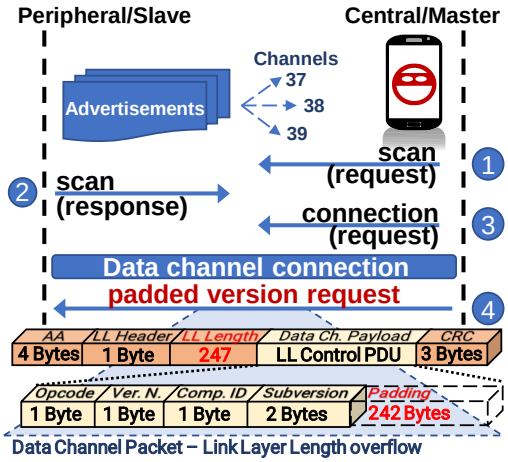

Figure 4: Link Layer Length Overflow vulnerability

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 is 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.

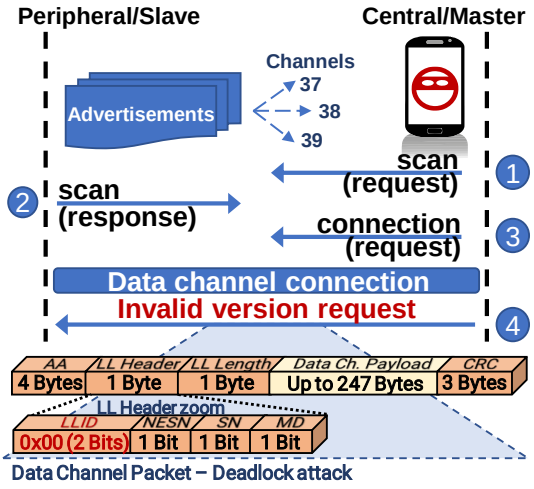

Figure 5: Link Layer LLID deadlock vulnerability

6.3 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 6 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.

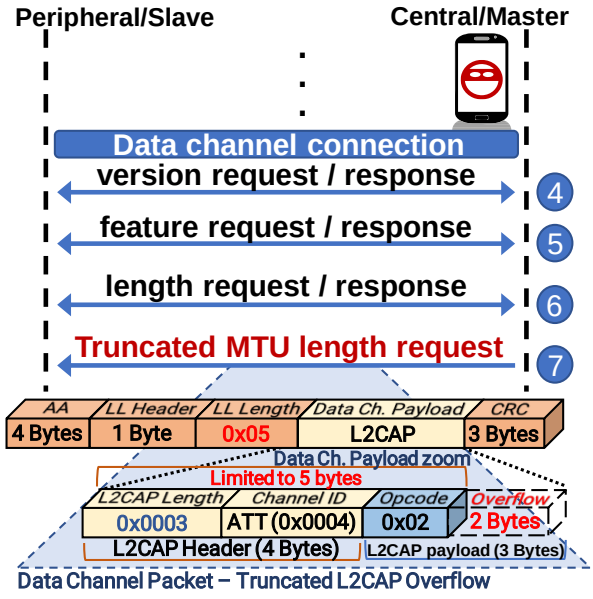

Figure 6: Truncated L2CAP vulnerability

6.4 Silent Length Overflow (CVE-2019-17518)

This attack is similar to Link *Layer Length Overflow* (cf. Section 6.1). 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 [16], 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.

6.5 Invalid Connection Request (CVE-2019-19193)

We identified that sample applications provided in *Texas Instruments CC2640R2 BLE-STACK SDK* (v3.30.00.20 and prior) and *CC2540 SDK* (v1.5.0 and prior) do not properly handle some connection parameters when the central attempts a connection to the peripheral. Instead, the peripheral creates a connection with the central, but fails at a later stage and moves the peripheral state to idle (i.e., stops advertisement). If the idle state is not handled correctly in the product's code, the device does not go back to the advertisement stage 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. During reception of an invalid connection request, the BLE stack sends a connection request fail event to the application code (bleGAPConnNotAcceptable) and upon receiving this fail status, the sample application enters in idle state by default, thus stopping advertisements. This behaviour is not the flaw alone, as idle state is a common state in BLE and should be handled by the application, due to other state transitions that can occur during application operation.

Nevertheless, we have discovered that this state change under reception of invalid parameters is not sufficiently documented in TI SDK, which may lead product developers to not handle idle state. 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 7.

Interestingly, *CC2540* also accepts connection requests with packet length lower than expected (truncated), which triggers the same behaviour (i.e., Figure 7) due to its implementation assuming zero bytes for truncated fields.

Impact: An attacker in radio range can exploit the phenomenon observed in Figure 7 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 6.2).

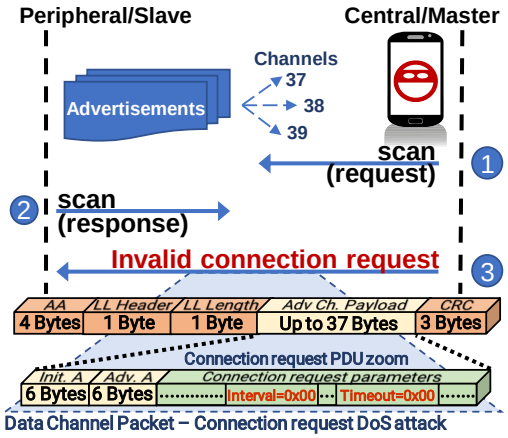

Figure 7: Invalid Connection Deadlock

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

This vulnerability was found on Texas Instruments *CC2640R2* BLE-STACK-SDK (v3.30.00.20 and prior versions). Specifically, the vulnerability is present in the implementation of the legacy pairing procedure which is handled by the Secure Manager Protocol (SMP) implementation. 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 8 (SC means secure connection).

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 exem-

plified 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.

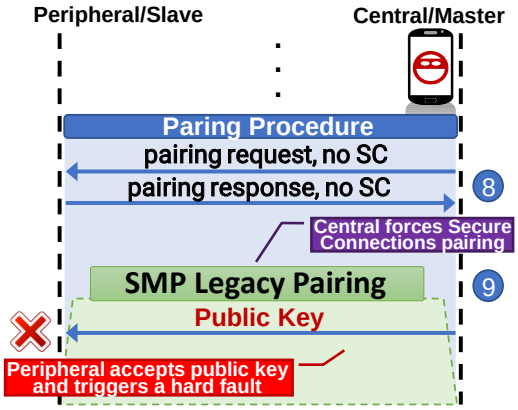

Figure 8: Unexpected Public Key Crash

6.7 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 9). 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.

6.8 Invalid L2CAP fragment (CVE-2019-19195)

During the communication between the central device and peripheral, the Bluetooth 4.0-4.1 Core specification dictates that the minimum and maximum link layer PDU size of L2CAP packets should be in the range 4-31 [15], considering that the

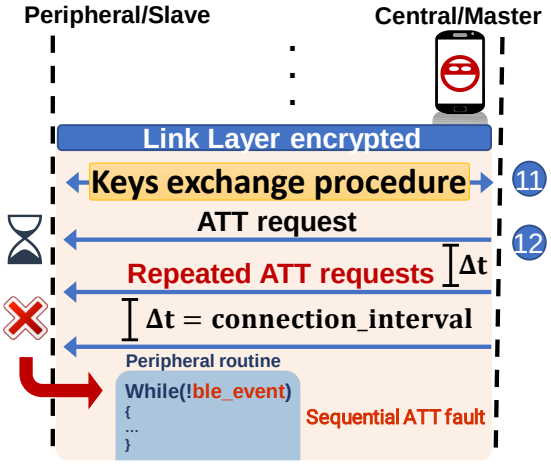

Figure 9: Sequential ATT Deadlock

L2CAP header is four bytes. 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 10, it was discovered that if an link layer PDU of length one is sent to the peripheral, then the device crashes due to the L2CAP header being truncated to just one byte. It also happens that this byte corresponds to the L2CAP length field. Thus, sending a higher value for this byte may lead to a buffer overflow and subsequently, crash the device.

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.

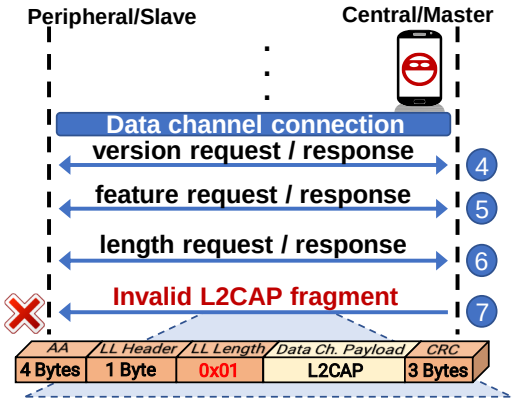

Figure 10: Invalid L2CAP fragment

6.9 Key Size Overflow (CVE-2019-19196)

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 is possible to force the peripheral into allocating the over sized key buffer length which was negotiated during the *pairing request*. Depicted in Figure 11, 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.

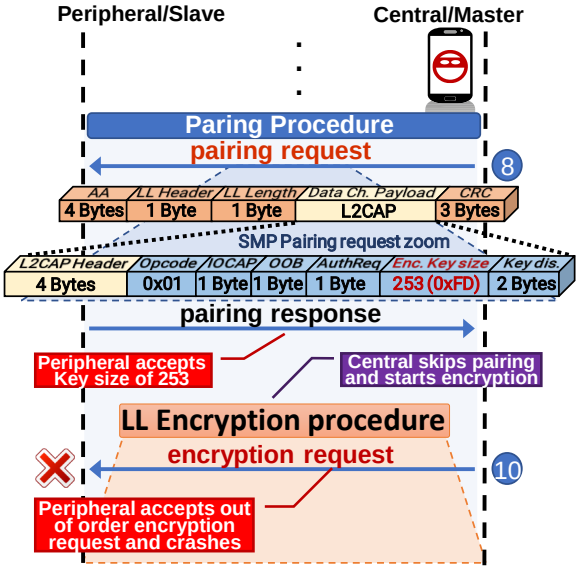

Figure 11: Key Size Overflow

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.

6.10 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.

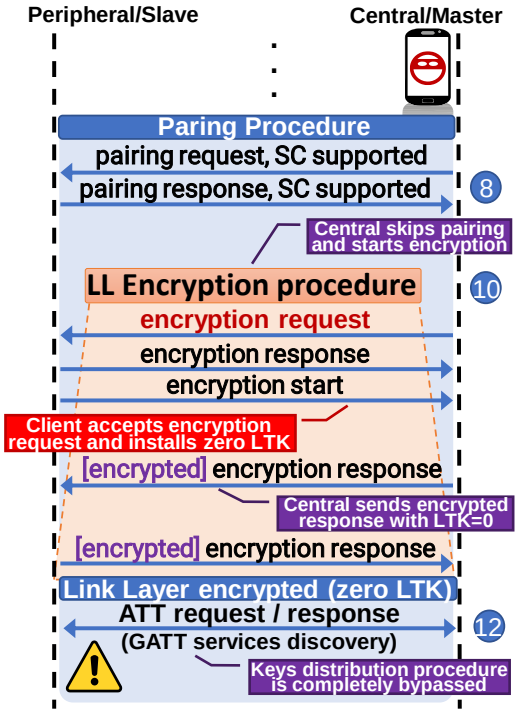

Figure 12: Zero LTK Installation

Following Figure 10, 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

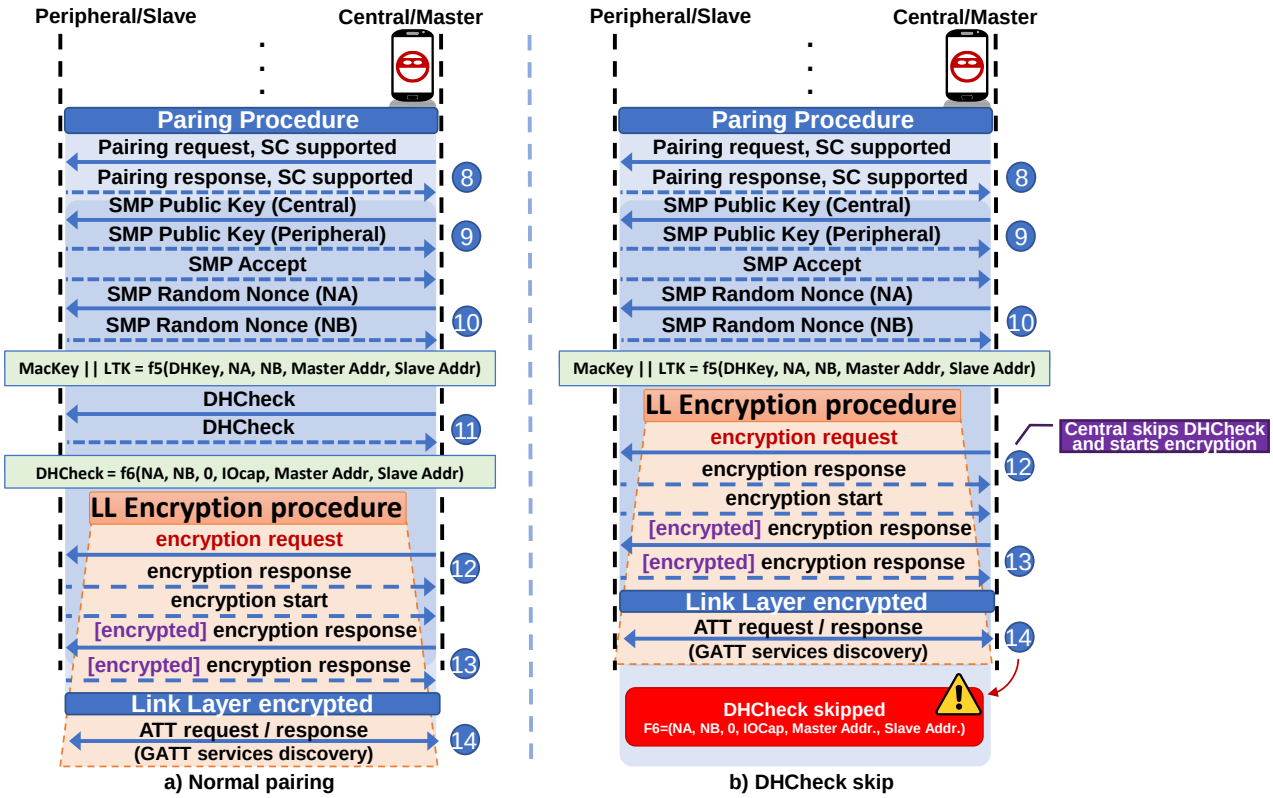

Figure 13: Diffie–Hellman Check Skip

as *sessionKey* [16]) is generated by the cryptographic function of Equation 1.

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

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.

6.11 DHCheck Skip (CVE-2020-13593)

This particular finding allows a partial security bypass. During the BLE secure connection pairing (stage 2), it is possible to bypass the DHCheck of a particular SoC vendor by starting the encryption setup early. The flaw was found in Texas Instrument CC2640R2 SMP implementation and allows the DHCheck to be skipped by starting the `LL Encryption Procedure` earlier (c.f. step 10 of Figure 1).

It is worthwhile to mention that, in Secure Connection Pairing, the LTK is actually generated before the DHCheck is complete. However, such a key (i.e. LTK) is normally discarded if something goes wrong during the DHCheck for security reasons. A normal DHCheck exchange (Figure 13.a) and the DHCheck skip (Figure 13.b) are shown in the figures below.

Following Figure 13.b, the peripheral installs 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). The additional security checks performed by f6 function is not verified and hence the legitimacy of the IO Capabilities (IOCap) is not verified.

Impact: Due to the requirements of the initial pairing procedure to be completed, this vulnerability can be reduced to

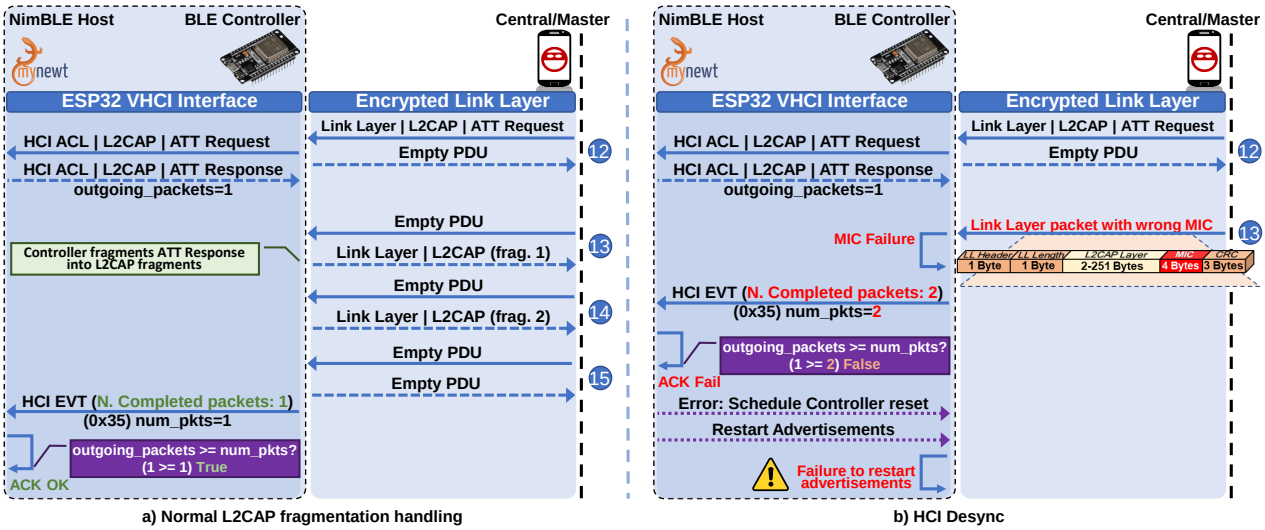

Figure 14: HCI Desync deadlock

a non-compliance as it cannot be exploited over-the-air (as it would require a compromised central BLE implementation to start the attack). Specifically, the attacker needs the private key associated with the public key exchanged in the beginning of the pairing procedure. This was to generate the LTK of the current pairing session to go through the *numeric comparison/just work/pass key* pairing models.

DHCheck Skip is triggered due to mishandling of the peripheral while starting the *Encryption Setup* during the *SMP Pairing*. Such a phenomenon is not currently clarified in the Core Specification [17]. This may explain why the vendor mishandled the scenario depicted in Figure 13. We note that the requirements to trigger DHCheck skipping is similar to Zero LTK installation, as it involves starting encryption setup earlier during the pairing stage, albeit without simply using a zeroed LTK.

6.12 HCI Desync Deadlock (CVE-2020-13595)

This denial of service (DoS) vulnerability explores a desynchronization between the Espressif Systems ESP32 BLE controller and the host running other Bluetooth related protocols. It was found that in a particular case, the HCI event, which indicates completed packets, returns the number of scheduled packet fragments by the controller instead of the number of packets solicited by the host. This is not a big issue by itself (albeit non-compliant to Bluetooth Core Specification 5.2 Vol 4, Part E, Section 7.3.40 [17]). However, when the host runs the Apache mynewt-nimble stack, the default behaviour is to try to recover the host by restarting the stack due to a possible de-synchronisation with the controller. When nimble tries to re-enable advertisements due to the faulty behaviour, the controller fails to do so as it was not really de-synchronized to start with.

The source of this vulnerability is not on the Mynewt-nimble central implementation itself, but on the ESP32 controller implementation which returns the wrong packet number in a corner-case scenario as shown below. The figure to the left shows a normal encrypted connection when the peripheral fragments packets from the peripheral host. The attacker needs to send a precise invalid packet. This causes an MIC error on the same connection event that the first fragment is supposed to be delivered by the peripheral. Normally, the HCI packet completion event returns 1 to indicate that one packet has been acknowledged. When the attack is performed, such HCI event returns 2 and enters a faulty detection code on nimble stack. This disables advertisements at a later stage on the peripheral's default sample code.

Impact: The Impact is denial of service by means of disabling the bluetooth advertisement of ESP32 peripherals which are using Nimble as the BLE host layer implementation. The attack requires encryption and pairing support enabled on the peripheral to be successful. ESP32 peripherals that are *not* using the default *just-works* pairing have reduced risk in terms of an attacker in radio range to directly start the attack. However, triggering an MIC failure can be accomplished while using an active BLE sniffing tool that offers hijacking, such as *btle-jack* [3].

6.13 Invalid Sequence Memory Corruption (CVE-2020-10061)

This vulnerability was found in version 2.2.0 of Zephyr RTOS Bluetooth stack implementation. It allows an attacker within radio range to cause a memory corruption by incorrectly starting a BLE connection with the target SoC employing Zephyr stack (step 4 on Figure 15). During a connection,

the central and the peripheral read and write to the flow control/acknowledgement bits (NESN and SN) on the Link Layer header to acknowledge each other. However, if the central starts a connection by sending an Anchor Point packet with NESN and SN bits set to 1, the Zephyr peripheral does not accept such bits as valid and performs invalid operation on its internal packet buffer. If the central proceeds with the connection by sending further packets, the peripheral *retry-buffer* gets full, which leads to a memory corruption (dangling pointer) and eventual crash of the Zephyr peripheral.

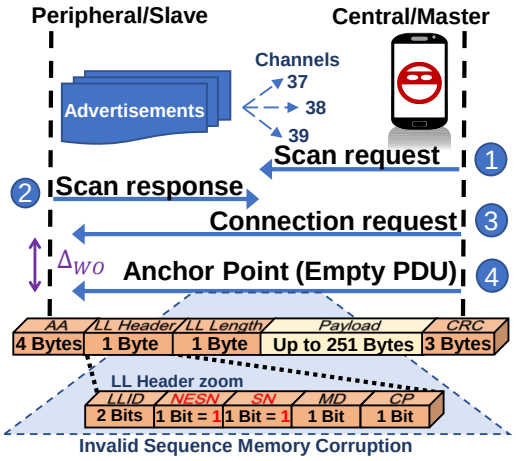

Figure 15: Invalid Sequence Memory Corruption

Impact: The impact is denial of service by means of crashing the peripheral or leaving it in an unstable state. Performing the attack and disconnecting quickly from the peripheral does not always trigger a crash, but makes the peripheral unstable. This makes it possible to lead the peripheral into crashing only when a legitimate central attempts the connection with the affected peripheral.

Furthermore, the attack is simple to trigger since starting a connection to a peripheral does not require any authentication.

6.14 Invalid Channel Map Crash/Deadlock (CVE-2020-10069, CVE-2020-13594)

The Invalid Channel Map Crash/Deadlock was found to affect **Microchip ATSAMB11 BluSDK Smart v6.2**, **ESP32 esp-idf v4.2 (CVE-2020-13594)** and **nRF51/52 Zephyr Bluetooth Stack v2.2.0 (CVE-2020-10069)**. An attacker can trigger the vulnerability by simply sending a connection request with the channel map field cleared (e.g., 0x000000...). The channel map field indicates to the peripheral which physical BLE channels are allowed when performing frequency-hopping with the central.

After the aforementioned invalid connection request to ATSAMB11, the controller informs the peripheral host of the failed connection with an HCI status code 0x3E. However, the host does not correctly enable advertisements again, requiring user intervention.

As for ESP32 and nRF51/52, a hard fault occurs on the former and a reachable assert is triggered on the latter, leading both peripherals to restart immediately. It is worthwhile to mention that differently from Section 6.12, the hard fault/assert is triggered regardless of which BLE host stack is employed by the SoC (e.g., Espressif offers a port of nimble or bluedroid for ESP32). This is because the vulnerability exists in the ESP32 static Bluetooth Library [22] and Zephyr nRF51/52 Link Layer controller implementation [11].

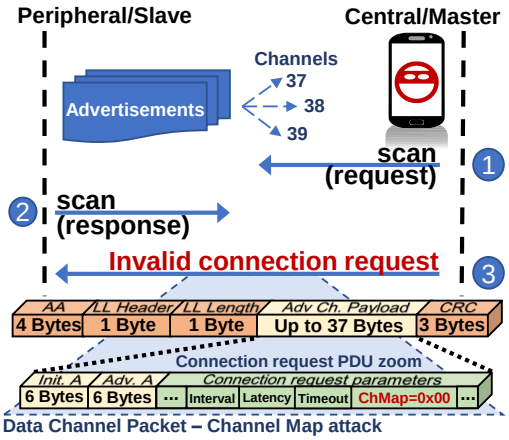

Figure 16: Channel Map Crash/Deadlock

Impact: The impact is denial of service by means of disabling Bluetooth advertisements on Microchip ATSAMB11 or restarting ESP32 or nRF51/52 Zephyr peripherals. Furthermore, triggering the attack is as simple as described in Section 6.13, as starting a connection to a peripheral does not require any authentication. It is important to mention that some Bluetooth Intellectual Property designs are shared and integrated amongst several silicon vendors. As a result, the Channel Map Crash/Deadlock can affect many other SoCs that we have not tested.

Contact

Feel free to reach us by email: sweyntooth@gmail.com

Acknowledgements

This research was partially supported by Keysight Technologies. We also want to thank all the involved vendors for their support during the coordination process and Ezekiel O. Soremekun for coining the term SWEYNTOOTH.

References

[1] Daniele Antonioli, Nils Ole Tippenhauer, and Kasper B. Rasmussen. The KNOB is broken: Exploiting low entropy in the encryption key negotiation of Bluetooth BR/EDR. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 1047–1061, Santa Clara, CA, August 2019. USENIX Association.

[2] Eli Biham and Lior Neumann. Breaking the bluetooth pairing–the fixed coordinate invalid curve attack. In *International Conference on Selected Areas in Cryptography*, pages 250–273. Springer, 2019.

[3] Damien Cauquil. BtleJack: a new Bluetooth Low Energy swiss-army knife, 2018. https://github.com/virtualabs/btlejack.

[4] Cypress. Component Datasheet - Bluetooth Low Energy (BLE), February 2020. https://www.cypress.com/documentation/component-datasheets/psoc-creator-component-datasheet-bluetooth-low-energy-ble.

[5] Cypress. Component Datasheet - Bluetooth Low Energy (BLE_PDL), February 2020. https://www.cypress.com/documentation/component-datasheets/bluetooth-low-energy-blepdl.

[6] Armis Inc. Bleedingbit vulnerability. https://armis.com/bleedingbit/, 2018.

[7] Texas Instruments. Invalid connection request advisory, February 2020. http://e2e.ti.com/support/wireless-connectivity/bluetooth/f/538/t/881881.

[8] Texas Instruments. Public Key Crash advisory, February 2020. https://e2e.ti.com/support/wireless-connectivity/bluetooth/f/538/t/881879.

[9] NXP. MCUXpresso Software Development Kit (SDK), January 2020. https://www.nxp.com/design/software/development-software/mcuxpresso-software-and-tools/mcuxpresso-software-development-kit-sdk:MCUXpresso-SDK?tab=Design_Tools_Tab.

[10] NXP. SweynTooth Advisory, March 2020. https://community.nxp.com/community/wireless-connectivity/security-updates/blog/2020/03/03/bluetooth-low-energy-vulnerabilities-sweyntooth.

[11] Zephyr Project. Zephyr nRF51/52 Bluetooth Controller Implementation, 2020. https://github.com/zephyrproject-rtos/zephyr/tree/master/subsys/bluetooth/controller/ll_sw.

[12] Zephyr Project. Zephyr RTOS Repository, 2020. https://github.com/zephyrproject-rtos/zephyr.

[13] Dialog Semiconductor. SweynTooth Advisory, February 2020. https://www.dialog-semiconductor.com/sweyntooth-bluetooth-low-energy-vulnerability.

[14] Ben Seri and Gregory Vishnepolsky. Blueborne. Armis. https://www.armis.com/blueborne/, 2017.

[15] Bluetooth SIG. Bluetooth Core Specification v4.2, December 2014. https://www.bluetooth.com/specifications/bluetooth-core-specification.

[16] Bluetooth SIG. Bluetooth Core Specification v5.0, December 2016. https://www.bluetooth.com/specifications/bluetooth-core-specification.

[17] Bluetooth SIG. Bluetooth Core Specification v5.1, January 2019. https://www.bluetooth.com/specifications/bluetooth-core-specification.

[18] Bluetooth SIG. View previously qualified designs and declared products, January 2020. https://launchstudio.bluetooth.com/Listings/Search.

[19] STMicroelectronics. STM32Cube MCU Package for STM32WB series, April 2020. https://www.st.com/en/embedded-software/stm32cubewb.html.

[20] STMicroelectronics. STSW-BLUENRG1-DK SW package, April 2020. https://www.st.com/en/embedded-software/stsw-bluenrg1-dk.html.

[21] Swipbox. Swipbox Infinity Product page, January 2019. https://www.swipbox.com/products_infinity.html.

[22] Espressif Systems. ESP32 BT/BLE Stack Libraries, 2020. https://github.com/espressif/esp32-bt-lib.

[23] Microchip Technology. SweynTooth Advisory, March 2020. https://www.microchip.com/design-centers/wireless-connectivity/bluetooth/sweyntooth-ble-vulnerability.

[24] Telink. Telink wiki - Kite BLE SDK V3.4.0, February 2020. http://wiki.telink-semi.cn/tools_and_sdk/BLE_SDK/8x5x_SDK/ble_sdk.zip.

[25] Telink. Telink wiki 826x Ble SDK v3.3, February 2020. http://wiki.telink-semi.cn/tools_and_sdk/BLE_SDK/826x_SDK/ble_sdk.rar.

[26] Telink. Telink wiki TLSR8232 SDK V1.3.0, February 2020. http://wiki.telink-semi.cn/tools_and_sdk/BLE_SDK/823x_SDK/ble_sdk.zip.