FirmXRay: Detecting Bluetooth Link Layer Vulnerabilities From Bare-Metal Firmware

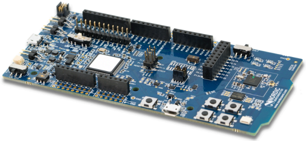
Haohuang Wen, Zhiqiang Lin, and Yinqian Zhang

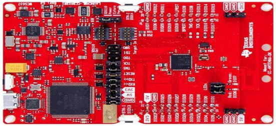
CCS 2020

Bluetooth Low Energy

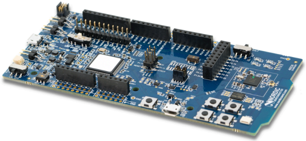
Bluetooth™
4.0
Low Energy

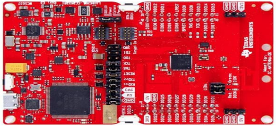
Low Technical Barrier for IoT Development



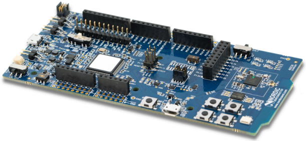


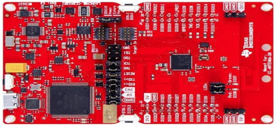
Low Technical Barrier for IoT Development





Low Technical Barrier for IoT Development

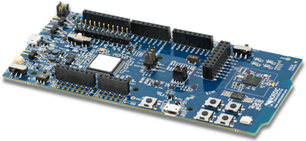


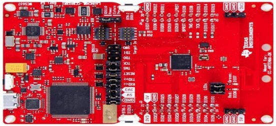


AWS IoT

Azure IoT Hub

Low Technical Barrier for IoT Development

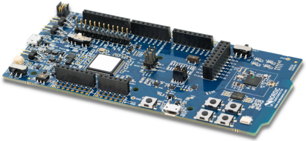


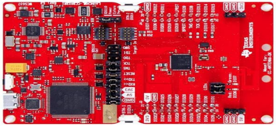


AWS IoT

Azure IoT Hub

Low Technical Barrier for IoT Development



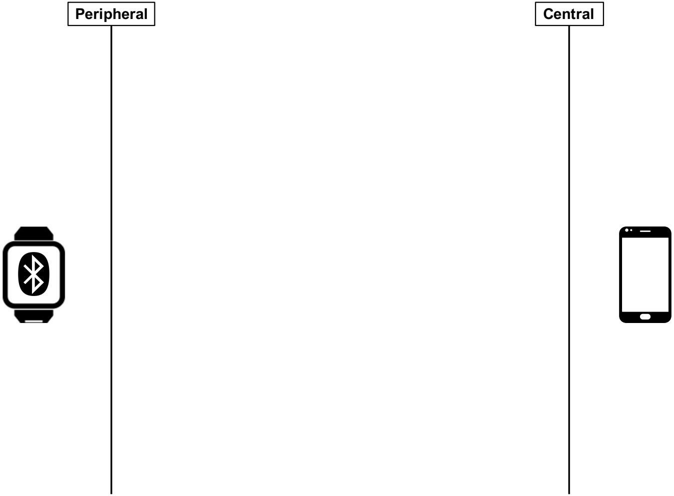


AWS IoT

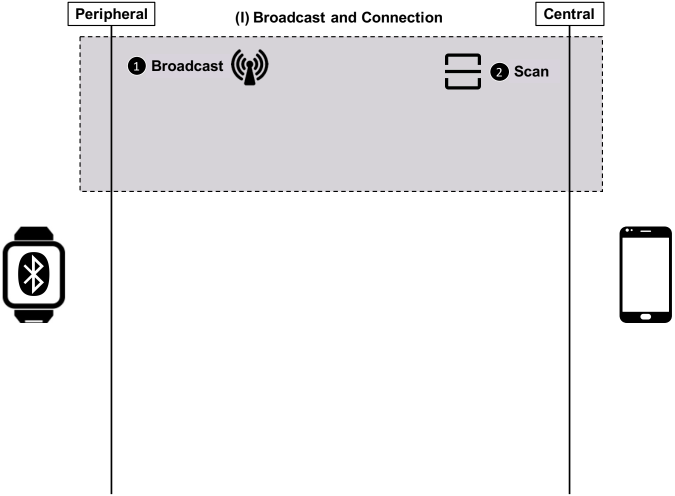
Azure IoT Hub

Are they secure?

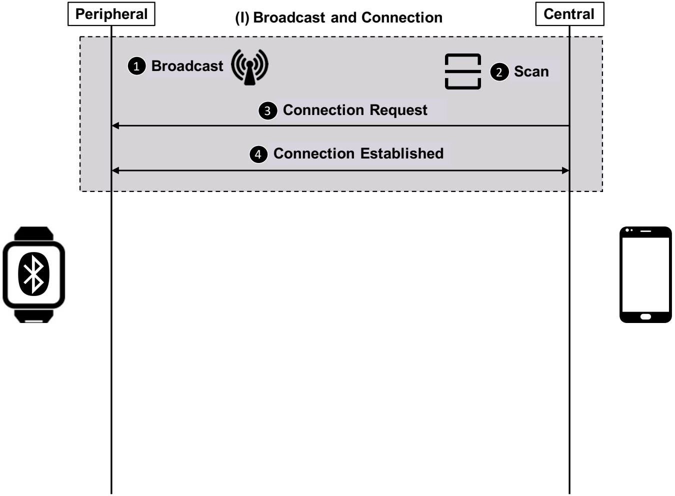
BLE Workflow



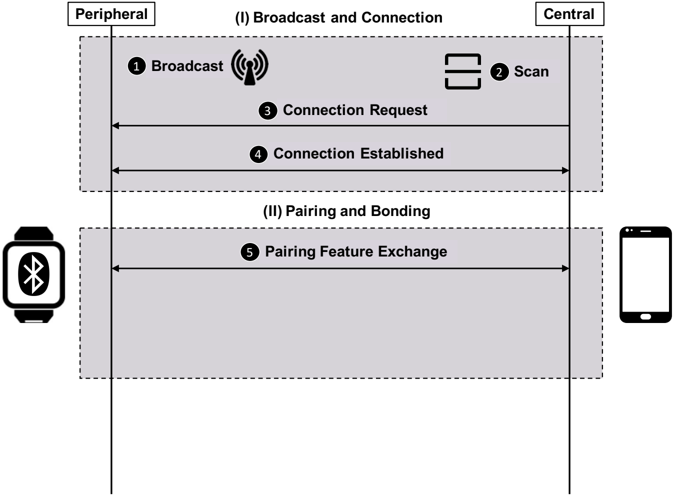
BLE Workflow



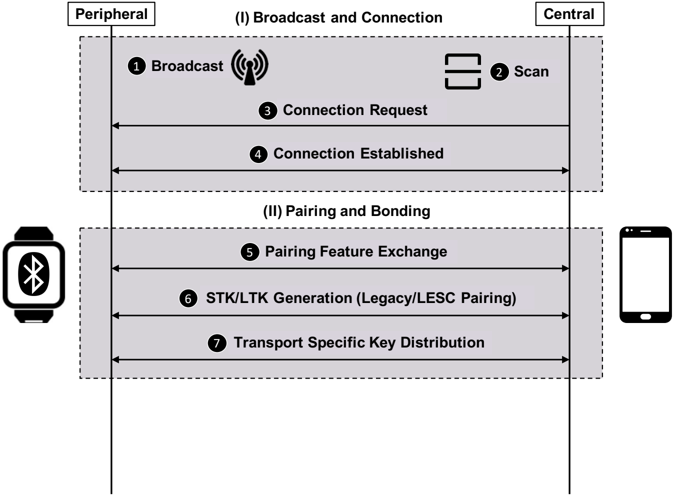
BLE Workflow



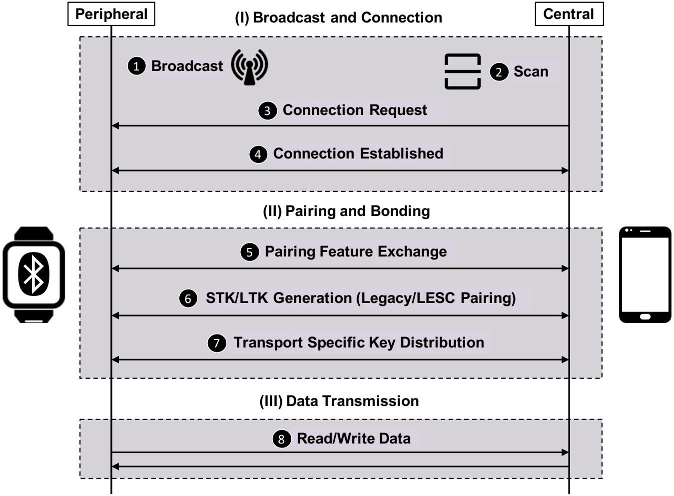
BLE Workflow



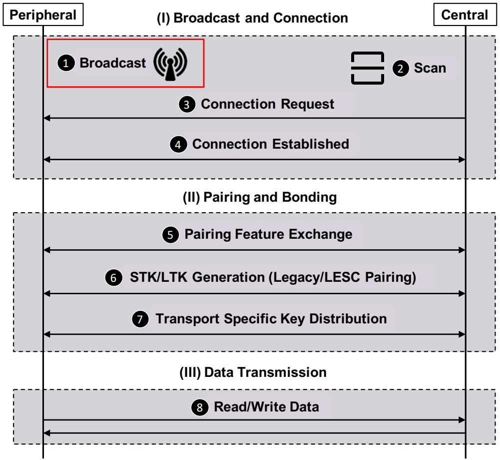
BLE Workflow



BLE Workflow



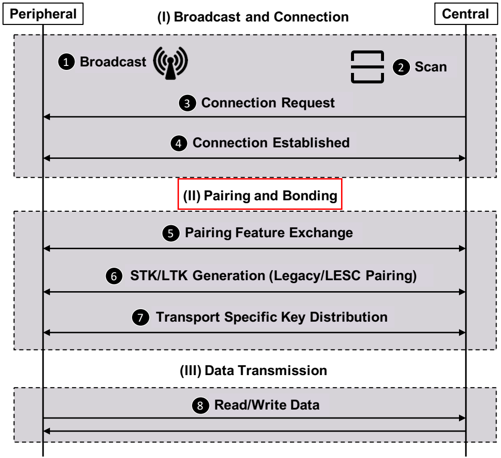
BLE Link Layer Vulnerabilities



Vulnerabilities

1. **Identity Tracking**. Configure static MAC address during broadcast [DPCM16].

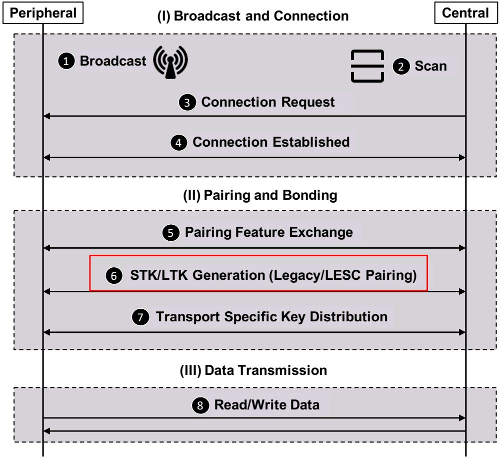
BLE Link Layer Vulnerabilities



Vulnerabilities

1. **Identity Tracking**. Configure static MAC address during broadcast [DPCM16].
2. **Active MITM**. Just Works is adopted as the pairing method.

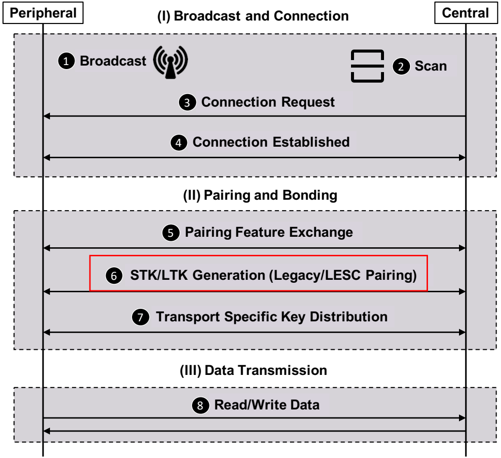
BLE Link Layer Vulnerabilities



Vulnerabilities

1. **Identity Tracking**. Configure static MAC address during broadcast [DPCM16].
2. **Active MITM**. Just Works is adopted as the pairing method.
3. **Passive MITM**. Legacy pairing is used during key exchange [ble14].

BLE Link Layer Vulnerabilities



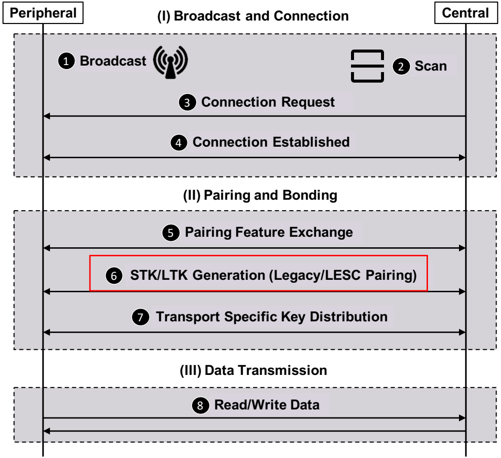
Vulnerabilities

1. **Identity Tracking**. Configure static MAC address during broadcast [DPCM16].
2. **Active MITM**. Just Works is adopted as the pairing method.
3. **Passive MITM**. Legacy pairing is used during key exchange [ble14].

Identification

1. Traffic analysis
2. Mobile app analysis

BLE Link Layer Vulnerabilities



Vulnerabilities

1. **Identity Tracking**. Configure static MAC address during broadcast [DPCM16].
2. **Active MITM**. Just Works is adopted as the pairing method.
3. **Passive MITM**. Legacy pairing is used during key exchange [ble14].

Identification

1. Traffic analysis
2. Mobile app analysis
3. Firmware analysis

An Example of a Just Works Pairing Vulnerability

Read Only Memory

```
243a8  mov  r2, #0x0
243aa  orr  r2, #0x1
243ac  and  r2, #0xe1
243ae  add  r2, #0xc
243b0  and  r2, #0xdf
243b2  ldr  r1, [0x260c8]
243b4  str  r2, [r1,#0x0]
...
25f44  ldr  r2, [0x260c8]
25f46  mov  r1, #0x0
25f48  svc  0x7f
// SD_BLE_GAP_SEC_PARAMS_REPLY
...
260c8  0x20003268
       // ble_gap_sec_parms_t*
```

Register Values

r1 = 0x0
r2 = 0x0

An Example of a Just Works Pairing Vulnerability

Read Only Memory

```
243a8   mov   r2, #0x0
243aa   orr   r2, #0x1
243ac   and   r2, #0xe1
243ae   add   r2, #0xc
243b0   and   r2, #0xdf
243b2   ldr   r1, [0x260c8]
243b4   str   r2, [r1,#0x0]
...
25f44   ldr   r2, [0x260c8]
25f46   mov   r1, #0x0
25f48   svc   0x7f
// SD_BLE_GAP_SEC_PARAMS_REPLY
...
260c8   0x20003268
        // ble_gap_sec_parms_t*
```

Register Values

```
r1 = 0x0
r2 = 0xD
```

An Example of a Just Works Pairing Vulnerability

Read Only Memory

```
243a8   mov   r2, #0x0
243aa   orr   r2, #0x1
243ac   and   r2, #0xe1
243ae   add   r2, #0xc
243b0   and   r2, #0xdf
243b2   ldr   r1, [0x260c8]
243b4   str   r2, [r1,#0x0]
...
25f44   ldr   r2, [0x260c8]
25f46   mov   r1, #0x0
25f48   svc   0x7f
// SD_BLE_GAP_SEC_PARAMS_REPLY
...
260c8   0x20003268
        // ble_gap_sec_parms_t*
```

Random Access Memory

Struct ble_gap_sec_params_t

```
20003268   uint8 pairing_feature

20003269   uint8 min_key_size
20003270   uint8 max_key_size
20003271   ble_gap_sec_kdist_t  kdist_own
20003275   ble_gap_sec_kdist_t  kdist_peer
```

Register Values

```
r1 = 0x20003268
r2 = 0xD
```

An Example of a Just Works Pairing Vulnerability

Read Only Memory

```
243a8    mov    r2, #0x0
243aa    orr    r2, #0x1
243ac    and    r2, #0xe1
243ae    add    r2, #0xc
243b0    and    r2, #0xdf
243b2    ldr    r1, [0x260c8]
243b4    str    r2, [r1,#0x0]
...
25f44    ldr    r2, [0x260c8]
25f46    mov    r1, #0x0
25f48    svc    0x7f
// SD_BLE_GAP_SEC_PARAMS_REPLY
...
260c8    0x20003268
         // ble_gap_sec_parms_t*
```

Random Access Memory

Struct ble_gap_sec_params_t

```
20003268   uint8 pairing_feature  = 0xD

20003269   uint8 min_key_size
20003270   uint8 max_key_size
20003271   ble_gap_sec_kdist_t  kdist_own
20003275   ble_gap_sec_kdist_t  kdist_peer
```

Register Values

```
r1 = 0x20003268
r2 = 0xD
```

An Example of a Just Works Pairing Vulnerability

Read Only Memory

```
243a8  mov  r2, #0x0
243aa  orr  r2, #0x1
243ac  and  r2, #0xe1
243ae  add  r2, #0xc
243b0  and  r2, #0xdf
243b2  ldr  r1, [0x260c8]
243b4  str  r2, [r1,#0x0]
...
25f44  ldr  r2, [0x260c8]
25f46  mov  r1, #0x0
25f48  svc  0x7f
// SD_BLE_GAP_SEC_PARAMS_REPLY
...
260c8       0x20003268
            // ble_gap_sec_parms_t*
```

Random Access Memory

Struct ble_gap_sec_params_t

```
20003268  uint8 pairing_feature = 0xD

20003269  uint8 min_key_size
20003270  uint8 max_key_size
20003271  ble_gap_sec_kdist_t kdist_own
20003275  ble_gap_sec_kdist_t kdist_peer
```

Register Values

```
r1 = 0x0
r2 = 0x20003268
```

An Example of a Just Works Pairing Vulnerability

Read Only Memory

```
243a8   mov   r2, #0x0
243aa   orr   r2, #0x1
243ac   and   r2, #0xe1
243ae   add   r2, #0xc
243b0   and   r2, #0xdf
243b2   ldr   r1, [0x260c8]
243b4   str   r2, [r1,#0x0]
...
25f44   ldr   r2, [0x260c8]
25f46   mov   r1, #0x0
25f48   svc   0x7f
// SD_BLE_GAP_SEC_PARAMS_REPLY
...
260c8         0x20003268
              // ble_gap_sec_parms_t*
```

Random Access Memory

Struct ble_gap_sec_params_t

```
20003268   uint8 pairing_feature  = 0xD
```

BOND	MITM	IO	OOB

```
           // BOND  = 1, MITM  = 0
           // IO    = 3, OOB   = 0
20003269   uint8 min_key_size
20003270   uint8 max_key_size
20003271   ble_gap_sec_kdist_t  kdist_own
20003275   ble_gap_sec_kdist_t  kdist_peer
```

Register Values

r1 = 0x0
r2 = 0x20003268

An Example of a Just Works Pairing Vulnerability

Correct Firmware Disassembling

Read Only Memory

```
243a8   mov   r2, #0x0
243aa   orr   r2, #0x1
243ac   and   r2, #0xe1
243ae   add   r2, #0xc
243b0   and   r2, #0xdf
243b2   ldr   r1, [0x260c8]
243b4   str   r2, [r1,#0x0]
...
25f44   ldr   r2, [0x260c8]
25f46   mov   r1, #0x0
25f48   svc   0x7f
// SD_BLE_GAP_SEC_PARAMS_REPLY
...
260c8   0x20003268
        // ble_gap_sec_parms_t*
```

Random Access Memory

Struct ble_gap_sec_params_t

```
20003268   uint8 pairing_feature  = 0xD
```

BOND	MITM	IO	OOB

```
           // BOND = 1, MITM = 0
           // IO   = 3, OOB  = 0
20003269   uint8 min_key_size
20003270   uint8 max_key_size
20003271   ble_gap_sec_kdist_t  kdist_own
20003275   ble_gap_sec_kdist_t  kdist_peer
```

Register Values

```
r1 = 0x0
r2 = 0x20003268
```

An Example of a Just Works Pairing Vulnerability

Correct Firmware Disassembling → **Recognize data structures**

Read Only Memory

```
243a8   mov    r2, #0x0
243aa   orr    r2, #0x1
243ac   and    r2, #0xe1
243ae   add    r2, #0xc
243b0   and    r2, #0xdf
243b2   ldr    r1, [0x260c8]
243b4   str    r2, [r1,#0x0]
...
25f44   ldr    r2, [0x260c8]
25f46   mov    r1, #0x0
25f48   svc    0x7f
// SD_BLE_GAP_SEC_PARAMS_REPLY
...
260c8   0x20003268
        // ble_gap_sec_parms_t*
```

Random Access Memory

Struct ble_gap_sec_params_t

```
20003268   uint8 pairing_feature  = 0xD
```

BOND	MITM	IO	OOB

```
           // BOND  = 1,  MITM  = 0
           // IO    = 3,  OOB   = 0
20003269   uint8 min_key_size
20003270   uint8 max_key_size
20003271   ble_gap_sec_kdist_t  kdist_own
20003275   ble_gap_sec_kdist_t  kdist_peer
```

Register Values

```
r1 = 0x0
r2 = 0x20003268
```

An Example of a Just Works Pairing Vulnerability

Correct Firmware Disassembling → **Read Only Memory**

```
243a8   mov   r2, #0x0
243aa   orr   r2, #0x1
243ac   and   r2, #0xe1
243ae   add   r2, #0xc
243b0   and   r2, #0xdf
243b2   ldr   r1, [0x260c8]
243b4   str   r2, [r1,#0x0]
...
25f44   ldr   r2, [0x260c8]
25f46   mov   r1, #0x0
25f48   svc   0x7f
// SD_BLE_GAP_SEC_PARAMS_REPLY
...
260c8   0x20003268
        // ble_gap_sec_parms_t*
```

Recognize data structures → **Random Access Memory**

Struct **ble_gap_sec_params_t**

```
20003268   uint8 pairing_feature  = 0xD
```

BOND	MITM	IO	OOB

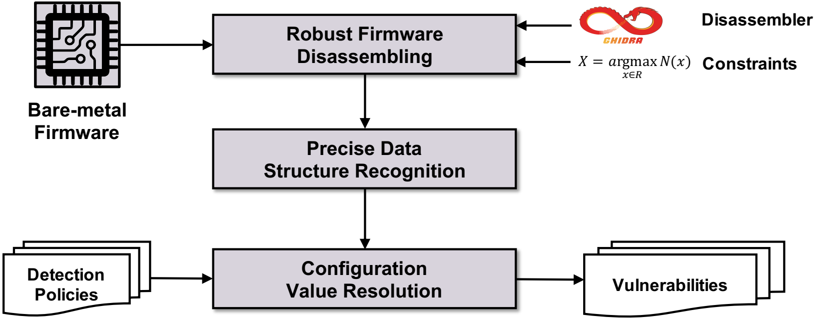
```
           // BOND = 1, MITM = 0
           // IO   = 3, OOB  = 0
20003269   uint8 min_key_size
20003270   uint8 max_key_size
20003271   ble_gap_sec_kdist_t  kdist_own
20003275   ble_gap_sec_kdist_t  kdist_peer
```

Value computation →

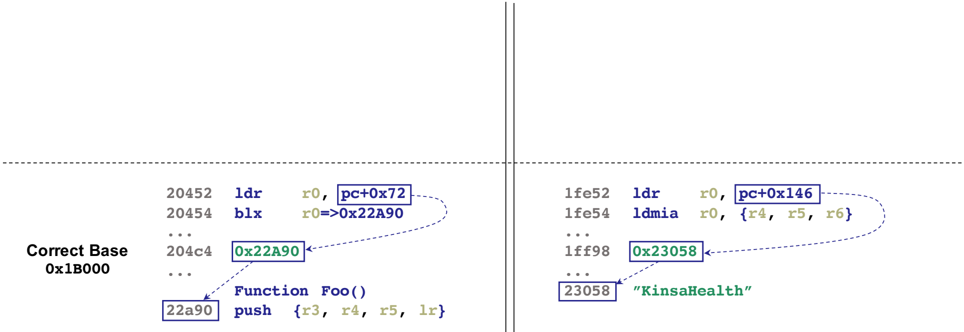
Register Values

```
r1 = 0x0
r2 = 0x20003268
```

FirmXRay Overview



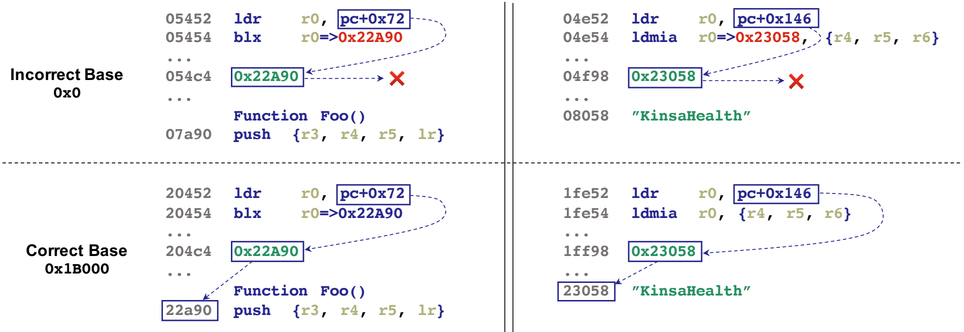
Robust Firmware Disassembling



(1) Absolute Function Pointer

(2) Absolute String Pointer

Robust Firmware Disassembling



(1) Absolute Function Pointer

(2) Absolute String Pointer

Robust Firmware Disassembling

Base
0x0

```
05452   ldr     r0, pc+0x72
05454   blx     r0=>0x22A90
...
054c4   0x22A90
...
        Function Foo()
07a90   push   {r3, r4, r5, lr}
```

```
04e52   ldr     r0, pc+0x146
04e54   ldmia   r0=>0x23058, {r4, r5, r6}
...
04f98   0x23058
...
08058   "KinsaHealth"
```

Robust Firmware Disassembling

Base
0x0

```
05452  ldr    r0, pc+0x72
05454  blx    r0=>0x22A90
...
054c4  0x22A90
...
       Function Foo()
07a90  push   {r3, r4, r5, lr}
```

```
04e52  ldr    r0, pc+0x146
04e54  ldmia  r0=>0x23058, {r4, r5, r6}
...
04f98  0x23058
...
08058  "KinsaHealth"
```

Absolute Pointers: 0x22A90, 0x23058

Gadgets: 0x07A90, 0x08058

Robust Firmware Disassembling

Base 0x0

```
05452  ldr    r0, pc+0x72
05454  blx    r0=>0x22A90
...
054c4  0x22A90
...
       Function Foo()
07a90  push   {r3, r4, r5, lr}
```

```
04e52  ldr    r0, pc+0x146
04e54  ldmia  r0=>0x23058, {r4, r5, r6}
...
04f98  0x23058
...
08058  "KinsaHealth"
```

Absolute Pointers: 0x22A90, 0x23058

Gadgets: 0x07A90, 0x08058

N(0x1B000) = 2

Robust Firmware Disassembling

Base
0x0

```
05452  ldr    r0, pc+0x72
05454  blx    r0=>0x22A90
...
054c4  0x22A90
...
       Function Foo()
07a90  push   {r3, r4, r5, lr}
```

```
04e52  ldr    r0, pc+0x146
04e54  ldmia  r0=>0x23058, {r4, r5, r6}
...
04f98  0x23058
...
08058  "KinsaHealth"
```

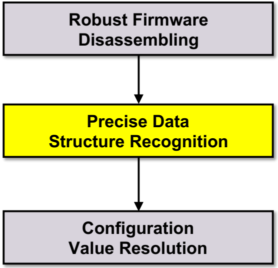
Absolute Pointers: 0x22A90, 0x23058

Gadgets: 0x07A90, 0x08058

0x1B000 = 0x22A90 − 0x07A90
= 0x23058 − 0x08058

N(0x1B000) = 2

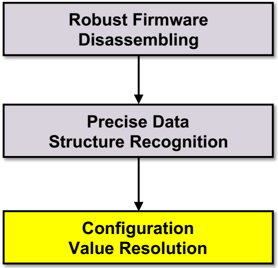
Precise Data Structure Recognition



Read Only Memory

```
243a8    mov    r2, #0x0
243aa    orr    r2, #0x1
243ac    and    r2, #0xe1
243ae    add    r2, #0xc
243b0    and    r2, #0xdf
243b2    ldr    r1, [0x260c8]
243b4    str    r2, [r1,#0x0]
...
25f44    ldr    r2, [0x260c8]
25f46    mov    r1, #0x0
25f48    svc    0x7f
// SD_BLE_GAP_SEC_PARAMS_REPLY(r0, r1, r2)
...
260c8    0x20003268
         // ble_gap_sec_parms_t*
```

Configuration Value Resolution

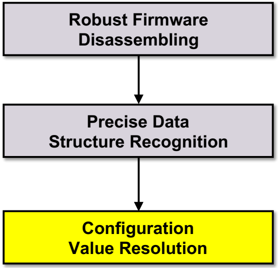


Read Only Memory

```
243a8  mov  r2, #0x0
243aa  orr  r2, #0x1
243ac  and  r2, #0xe1
243ae  add  r2, #0xc
243b0  and  r2, #0xdf
243b2  ldr  r1, [0x260c8]
243b4  str  r2, [r1,#0x0]
...
25f44  ldr  r2, [0x260c8]
25f46  mov  r1, #0x0
25f48  svc  0x7f
// SD_BLE_GAP_SEC_PARAMS_REPLY
...
260c8  0x20003268
       // ble_gap_sec_parms_t*
```

Program Path

Configuration Value Resolution



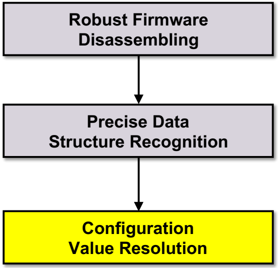
Read Only Memory

```
243a8   mov   r2, #0x0
243aa   orr   r2, #0x1
243ac   and   r2, #0xe1
243ae   add   r2, #0xc
243b0   and   r2, #0xdf
243b2   ldr   r1, [0x260c8]
243b4   str   r2, [r1,#0x0]
...
25f44   ldr   r2, [0x260c8]
25f46   mov   r1, #0x0
25f48   svc   0x7f
// SD_BLE_GAP_SEC_PARAMS_REPLY
...
260c8   0x20003268
        // ble_gap_sec_parms_t*
```

Program Path

```
ldr r2, [0x260c8]
str r2, [r1, #0x0]
```

Configuration Value Resolution



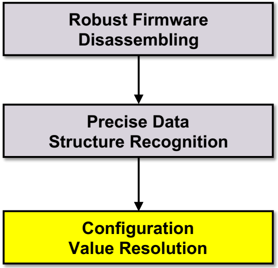
Read Only Memory

```
243a8   mov   r2, #0x0
243aa   orr   r2, #0x1
243ac   and   r2, #0xe1
243ae   add   r2, #0xc
243b0   and   r2, #0xdf
243b2   ldr   r1, [0x260c8]
243b4   str   r2, [r1,#0x0]
...
25f44   ldr   r2, [0x260c8]
25f46   mov   r1, #0x0
25f48   svc   0x7f
// SD_BLE_GAP_SEC_PARAMS_REPLY
...
260c8   0x20003268
        // ble_gap_sec_parms_t*
```

Program Path

```
ldr r2, [0x260c8]
str r2, [r1, #0x0]
ldr r1, [0x260c8]
and r2, #0xdf
add r2, #0xc
and r2, #0xe1
orr r2, #0x1
mov r2, #0x0
```

Configuration Value Resolution



Read Only Memory

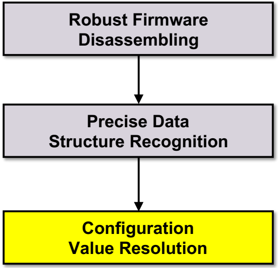
```
243a8  mov  r2, #0x0
243aa  orr  r2, #0x1
243ac  and  r2, #0xe1
243ae  add  r2, #0xc
243b0  and  r2, #0xdf
243b2  ldr  r1, [0x260c8]
243b4  str  r2, [r1,#0x0]
...
25f44  ldr  r2, [0x260c8]
25f46  mov  r1, #0x0
25f48  svc  0x7f
// SD_BLE_GAP_SEC_PARAMS_REPLY
...
260c8  0x20003268
       // ble_gap_sec_parms_t*
```

Program Path

```
ldr r2, [0x260c8]
str r2, [r1, #0x0]
ldr r1, [0x260c8]
and r2, #0xdf
add r2, #0xc
and r2, #0xe1
orr r2, #0x1
mov r2, #0x0
```

r2 = 0x20003268

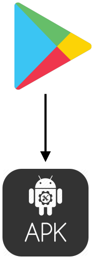
Configuration Value Resolution



Policy	SDK Function Name	Reg. Index	Description
(i)	SD_BLE_GAP_ADDR_SET	0	Configure the MAC address
	SD_BLE_GAP_APPEARANCE_SET	0	Set device description
	SD_BLE_GATTS_SERVICE_ADD	0, 1	Add a BLE GATT service
	SD_BLE_GATTS_CHARACTERISTIC_ADD	2	Add a BLE GATT characteristic
	SD_BLE_UUID_VS_ADD	0	Specify the UUID base
	GAP_ConfigDeviceAddr*	0	Setup the address type
	GATTServApp_RegisterService*	0	Register BLE GATT service
(ii)	SD_BLE_GAP_SEC_PARAMS_REPLY	2	Reply peripheral pairing features
	SD_BLE_GAP_AUTH	1	Reply central pairing features
	SD_BLE_GAP_AUTH_KEY_REPLY	1, 2	Reply with an authentication key
	SD_BLE_GATTS_CHARACTERISTIC_ADD	2	Add a BLE GATT characteristic
	GAPBondMgr_SetParameter*	2	Setup pairing parameters
	GATTServApp_RegisterService*	0	Register BLE GATT service
(iii)	SD_BLE_GAP_LESC_DHKEY_REPLY	0	Reply with a DH key
	GAPBondMgr_SetParameter*	2	Setup pairing parameters

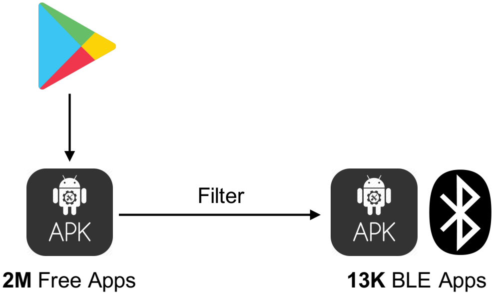
Firmware Collection

Firmware Collection

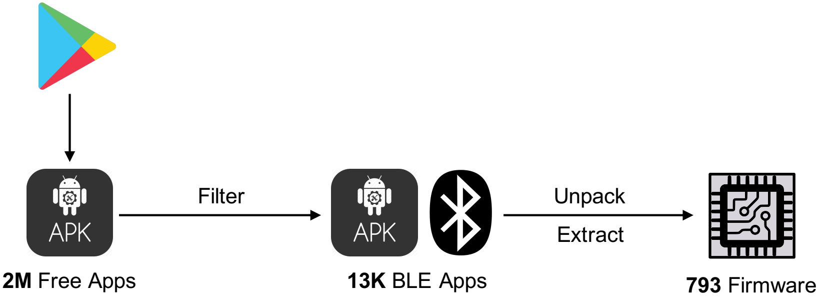


2M Free Apps

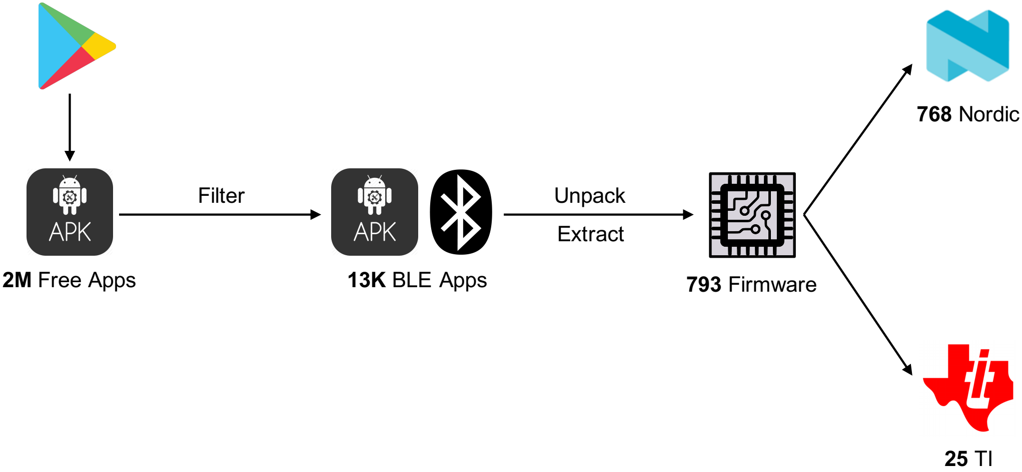
Firmware Collection



Firmware Collection



Firmware Collection



Firmware Categorization

- Firmware categorization

Firmware Categorization

- Firmware categorization
  - Descriptive APIs (e.g., SD_BLE_GAP_APPEARANCE_SET)

Firmware Categorization

- Firmware categorization
  - Descriptive APIs (e.g., SD_BLE_GAP_APPEARANCE_SET)
  - Mobile app descriptions

Firmware Categorization

- Firmware categorization
  - Descriptive APIs (e.g., SD_BLE_GAP_APPEARANCE_SET)
  - Mobile app descriptions

Category	# Firmware	# Device	Avg. Size (KB)
Nordic-based Firmware			
Wearable	204	138	98.2
Others	76	22	223.5
Sensor	67	51	80.9
Tag (Tracker)	58	41	84.2
Robot	41	21	117.7
Medical Devices	41	21	138.6
TI-based Firmware			
Sensor	19	19	132.9
Smart Lock	2	2	46.3
Smart Toy	2	2	47.8
Medical Devices	1	1	70.2
Others	1	1	76.7
Total	793	538	102.7

Table: Top categories of firmware.

Firmware Categorization

- Firmware categorization
  - Descriptive APIs (e.g., SD_BLE_GAP_APPEARANCE_SET)
  - Mobile app descriptions
- Firmware aggregation
  - Aggregate different versions of firmware of the same device
  - The 793 firmware represent 538 real devices

Category	# Firmware	# Device	Avg. Size (KB)
Nordic-based Firmware			
Wearable	204	138	98.2
Others	76	22	223.5
Sensor	67	51	80.9
Tag (Tracker)	58	41	84.2
Robot	41	21	117.7
Medical Devices	41	21	138.6
TI-based Firmware			
Sensor	19	19	132.9
Smart Lock	2	2	46.3
Smart Toy	2	2	47.8
Medical Devices	1	1	70.2
Others	1	1	76.7
Total	793	538	102.7

Table: Top categories of firmware.

Experiment Results

Identity Tracking Vulnerability Identification

Among the 538 devices, nearly all of them (98.1%) have configured random static addresses that do not change periodically.

Experiment Results

Identity Tracking Vulnerability Identification

Among the 538 devices, nearly all of them (98.1%) have configured random static addresses that do not change periodically.

Firmware Name	Mobile App	Category	# Device
cogobeacon	com.aegismobility.guardian	Car Accessory	4
sd_bl	fr.solem.solemwf	Agricultural Equip.	2
LRFL_nRF52	fr.solem.solemwf	Agricultural Equip.	2
orb	one.shade.app	Smart Light	1
sd_bl	com.rainbird	Agricultural Equip.	1

Table: Firmware using private MAC address.

Experiment Results

Active MITM Vulnerability Identification

385 (71.5%) devices use Just Works pairing, which essentially does not provide any protection against active MITM attacks at the BLE link layer.

Experiment Results

Active MITM Vulnerability Identification

385 (71.5%) devices use Just Works pairing, which essentially does not provide any protection against active MITM attacks at the BLE link layer.

Item	N	T	Total	%
# Total Device	513	25	538	100
# Device w/ active MITM vulnerability	384	1	385	71.5
# Device w/ Just Works pairing only	317	1	318	59.1
# Device w/ flawed Passkey implementation	37	0	37	6.9
# Device w/ flawed OOB implementation	30	0	30	5.6
# Device w/ secure pairing	6	24	30	3.8
# Device w/ correct Passkey implementation	3	24	27	3.4
# Device w/ correct OOB implementation	3	0	3	0.4

Table: Pairing configurations of devices (N:Nordic, T:TI).

Experiment Results

Passive MITM Vulnerability Identification

98.5% of the devices fail to enforce LESC pairing, and thus they can be vulnerable to passive MITM attacks if there is no application-layer encryption.

Experiment Results

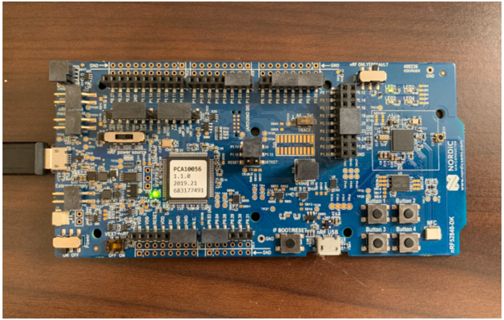
Passive MITM Vulnerability Identification

98.5% of the devices fail to enforce LESC pairing, and thus they can be vulnerable to passive MITM attacks if there is no application-layer encryption.

Firmware Name	Mobile App	Category	# Version
DogBodyBoard	com.wowwee.chip	Robot	16
BW_Pro	com.ecomm.smart_panel	Tag	1
Smart_Handle	com.exitec.smartlock	Smart Lock	1
Sma05	com.smalife.watch	Wearable	1
CPRmeter	com.laerdal.cprmeter2	Medical Device	4
WiJumpLE	com.wesssrl.wijumple	Sensor	1
nRF Beacon	no.nordicsemi.android.nrfbeacon	Beacon	1
Hoot Bank	com.qvivr.hoot	Debit Card	1

Table: Firmware that enforce LESC pairing.

Attack Case Studies



nRF52840 DK

Vulnerable BLE Devices

Attack Case Studies

Device Name	Category	Attacks		
		A1	A2	A3
Nuband Activ+	Wearable	✓		✓
Kinsa Smart	Thermometer			✓
Chipolo ONE	Tag	✓		
SwitchBot Button Pusher	Smart Home		✓	
XOSS Cycling Computer	Sensor	✓		✓

A1: User Tracking

Attack Case Studies

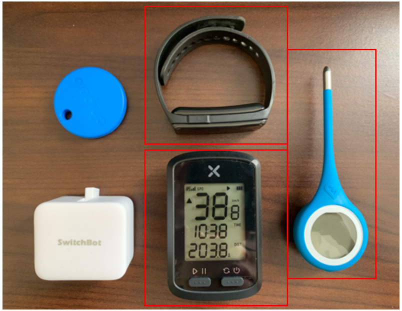
Device Name	Category	Attacks		
		A1	A2	A3
Nuband Activ+	Wearable	✓		✓
Kinsa Smart	Thermometer			✓
Chipolo ONE	Tag	✓		
SwitchBot Button Pusher	Smart Home		✓	
XOSS Cycling Computer	Sensor	✓		✓

A2: Unauthorized Control

Attack Case Studies

Device Name	Category	Attacks		
		A1	A2	A3
Nuband Activ+	Wearable	✓		✓
Kinsa Smart	Thermometer			✓
Chipolo ONE	Tag	✓		
SwitchBot Button Pusher	Smart Home		✓	
XOSS Cycling Computer	Sensor	✓		✓

A3: Sensitive Data Eavesdropping



Discussion

- **Effectiveness**. Source of FP/FN: incorrect base address and fundamental limitations of program analysis.

Discussion

- **Effectiveness**. Source of FP/FN: incorrect base address and fundamental limitations of program analysis.
- **Exploitation**. Not all the vulnerabilities can be exploited in practice.

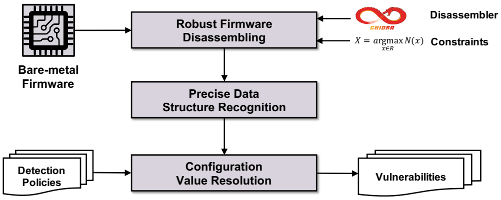
Discussion

- **Effectiveness**. Source of FP/FN: incorrect base address and fundamental limitations of program analysis.
- **Exploitation**. Not all the vulnerabilities can be exploited in practice.
- **Root Cause**. Lack of hardware capabilities and misconfiguration by the developers are the two major root causes.

Discussion

- **Effectiveness**. Source of FP/FN: incorrect base address and fundamental limitations of program analysis.
- **Exploitation**. Not all the vulnerabilities can be exploited in practice.
- **Root Cause**. Lack of hardware capabilities and misconfiguration by the developers are the two major root causes.
- **Future Work**.
  - Extract more embedded firmware from apps (e.g., those downloaded from server).
  - Adapt FirmXRay to other SDKs and architectures.
  - Enable dynamic analysis and firmware emulation [CGS+20] [CWBE16] [FML20].

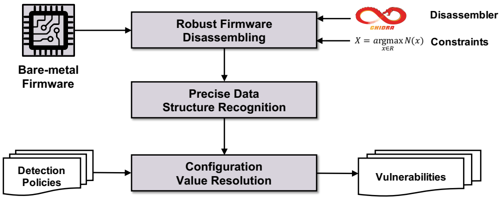
Takeaway



FirmXRay

- A static analysis tool based on Ghidra for detecting BLE link layer vulnerabilities from bare-metal firmware.
- A scalable approach to efficiently collect bare-metal firmware images from only mobile apps.
- Vulnerability discovery and attack case studies.

Takeaway



FIRMXRAY

- A static analysis tool based on Ghidra for detecting BLE link layer vulnerabilities from bare-metal firmware.
- A scalable approach to efficiently collect bare-metal firmware images from only mobile apps.
- Vulnerability discovery and attack case studies.

The source code is available at `https://github.com/OSUSecLab/FirmXRay`.

References I

- *Bluetooth specification version 4.2*, https://www.bluetooth.org/DocMan/handlers/DownloadDoc.ashx?doc_id=286439, 2014.
- Abraham Clements, Eric Gustafson, Tobias Scharnowski, Paul Grosen, David Fritz, Christopher Kruegel, Giovanni Vigna, Saurabh Bagchi, and Mathias Payer, *Halucinator: Firmware re-hosting through abstraction layer emulation*, Proceedings of the 29th USENIX Security Symposium (USENIX Security 20), USENIX Association, 2020.
- Daming D Chen, Maverick Woo, David Brumley, and Manuel Egele, *Towards automated dynamic analysis for linux-based embedded firmware.*, 2016 Network and Distributed Systems Security Symposium (NDSS), vol. 16, 2016, pp. 1–16.
- Aveek K Das, Parth H Pathak, Chen-Nee Chuah, and Prasant Mohapatra, *Uncovering privacy leakage in ble network traffic of wearable fitness trackers*, Proceedings of the 17th International Workshop on Mobile Computing Systems and Applications, ACM, 2016, pp. 99–104.
- Bo Feng, Alejandro Mera, and Long Lu, *P2im: Scalable and hardware-independent firmware testing via automatic peripheral interface modeling*, Proceedings of the 29th USENIX Security Symposium (USENIX Security 20), USENIX Association, 2020.