# Automatically Uncovering Vulnerabilities in Bluetooth Applications from Binary Code

Zhiqiang Lin
zlin@cse.ohio-state.edu

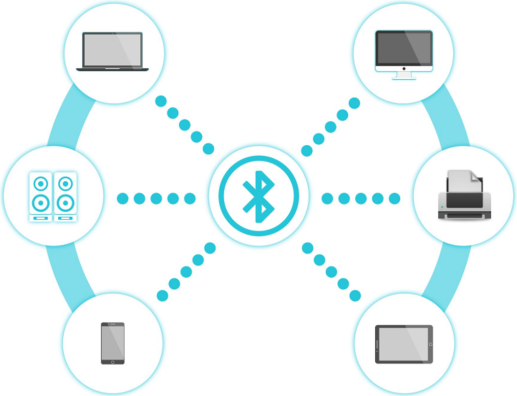
Joint work w/ Haohuang Wen, Yinqian Zhang, and Chaoshun Zuo

12/04/2020

# What is Bluetooth

**Bluetooth wireless technology**

- Low-cost, low-power
- Short-range radio
- For ad-hoc wireless communication
- For voice and data transmission



# What is Bluetooth





Google

Contact tracing


# Why Named Bluetooth

## Harald "Bluetooth" Gormsson

- King of Denmark 940-981.
- He was also known for his bad **tooth**, which had a very dark **blue-grey** shade.
- He united the Tribes of Denmark.

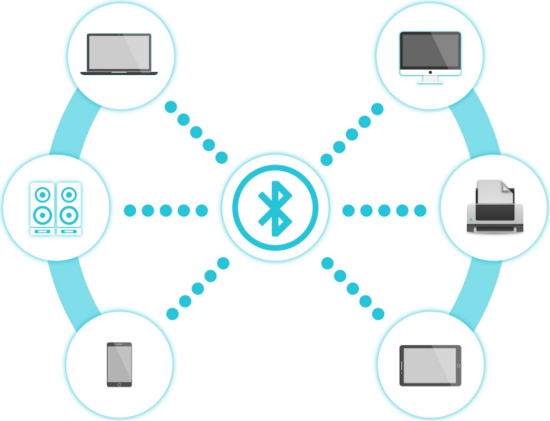
The Bluetooth wireless specification design was named after the king in 1997, based on an analogy **that the technology would unite devices the way Harald Bluetooth united the tribes of Denmark into a single kingdom**.


# Why Named Bluetooth

## Harald "Bluetooth" Gormsson

- King of Denmark 940-981.
- He was also known for his bad **tooth**, which had a very dark **blue-grey** shade.
- He united the Tribes of Denmark.

The Bluetooth wireless specification design was named after the king in 1997, based on an analogy **that the technology would unite devices the way Harald Bluetooth united the tribes of Denmark into a single kingdom**.



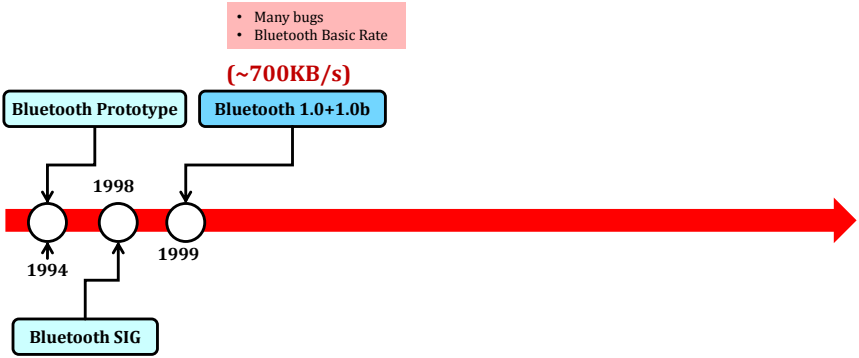
# History of Bluetooth

# History of Bluetooth

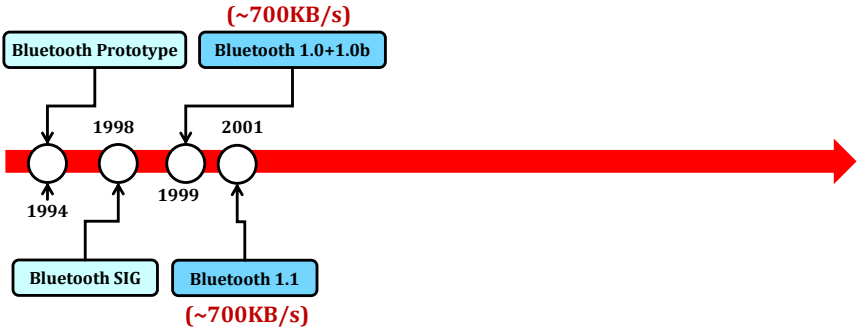
Dr. Jaap Haartsen started a project named Bluetooth.

# History of Bluetooth

# History of Bluetooth



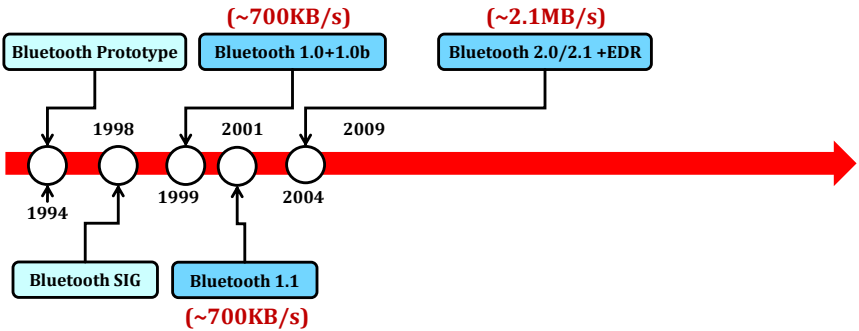
# History of Bluetooth



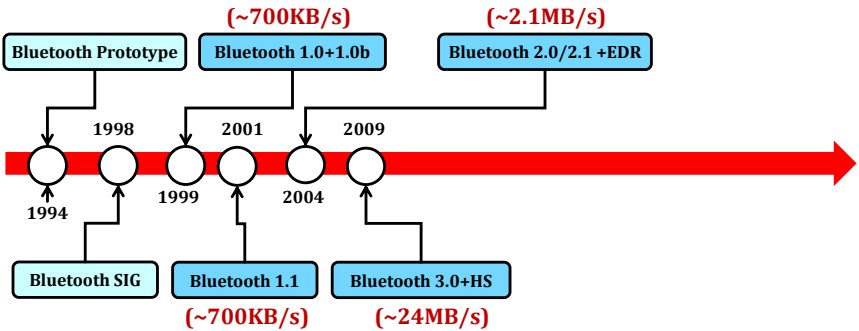

- Fixed security issues.
- First marketable product version.

# History of Bluetooth

- Introduced EDR (Enhanced Data Rate)
- Secure simple pairing (SSP) is introduced

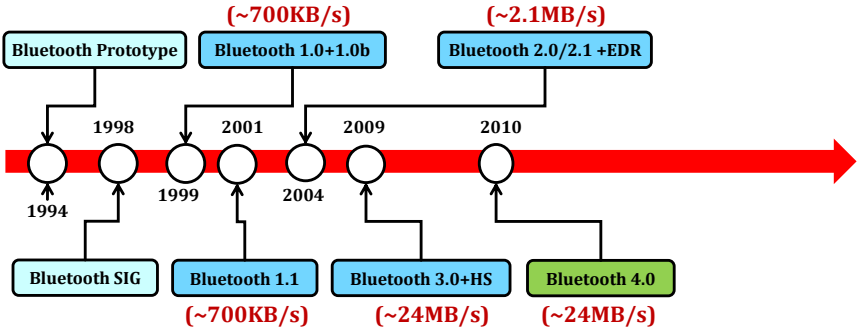


# History of Bluetooth



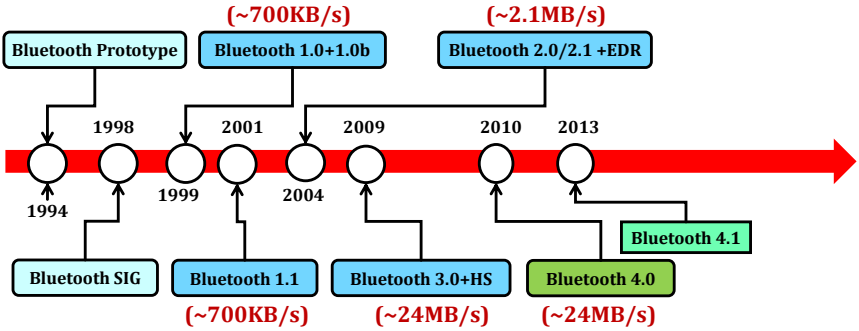

Introduced High-speed channel

# History of Bluetooth



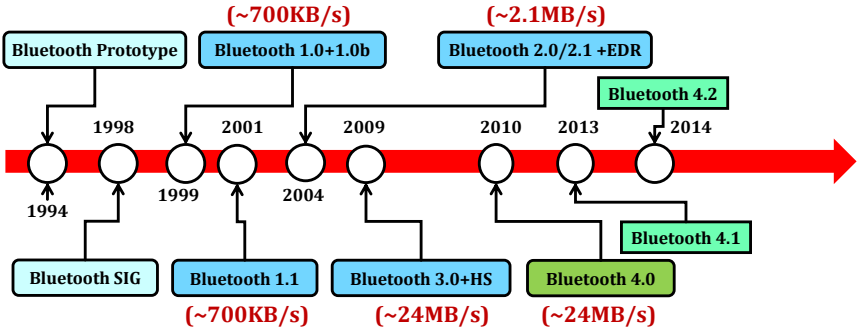

- Low energy (LE) protocol for IoT;
- 128-bit encryption/LE Privacy and Whitelisting

# History of Bluetooth



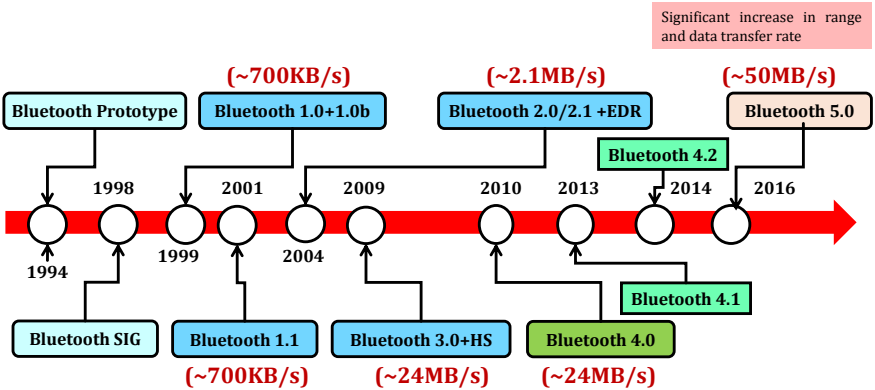

- Low energy (LE) protocol for IoT;
- 128-bit encryption/LE Privacy and Whitelisting

# History of Bluetooth

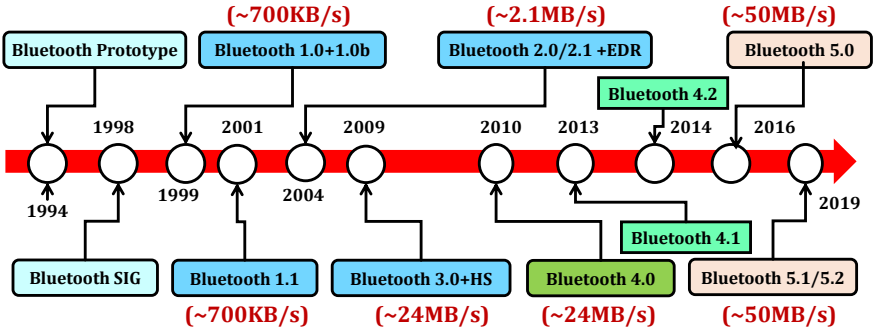



- Low energy (LE) protocol for IoT;
- 128-bit encryption/LE Privacy and Whitelisting

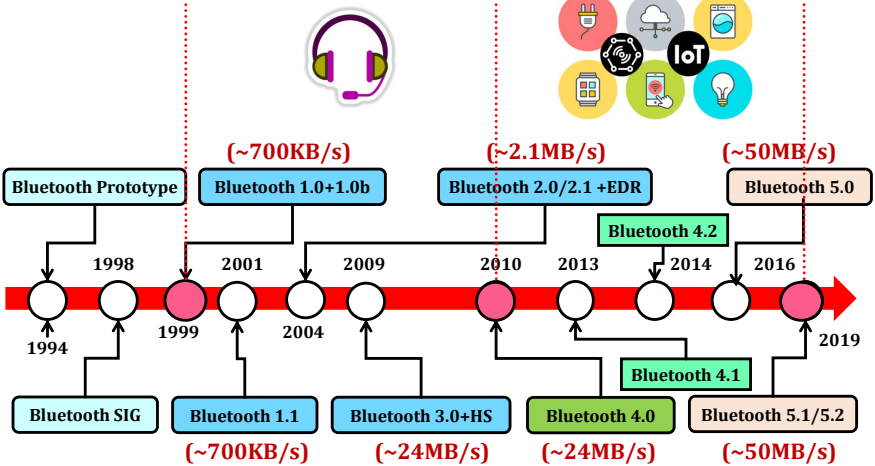
# History of Bluetooth



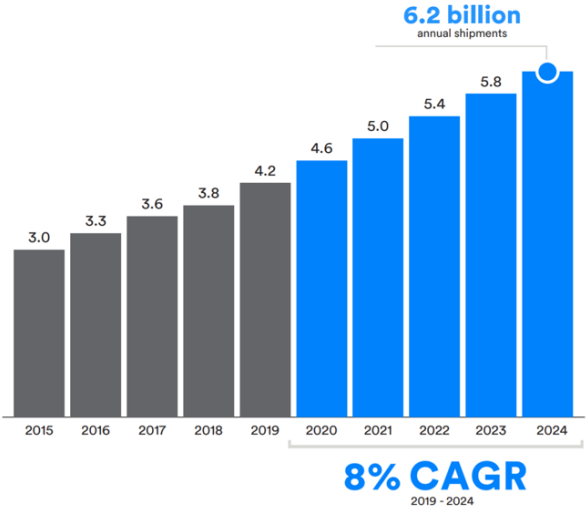
# History of Bluetooth



# History of Bluetooth

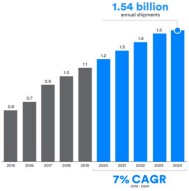


# Total Annual Bluetooth Device Shipments [SIG20]



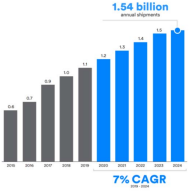
# Total Annual Bluetooth Device Shipments [SIG20]

**Annual Bluetooth Audio Streaming Device Shipments**

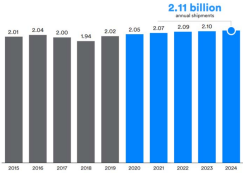


# Total Annual Bluetooth Device Shipments [SIG20]

**Annual Bluetooth Audio Streaming Device Shipments**

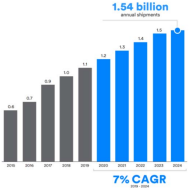



**Annual Bluetooth Phone, Tablet & PC Shipments**

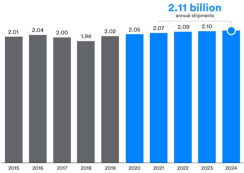


# Total Annual Bluetooth Device Shipments [SIG20]

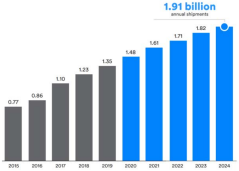
**Annual Bluetooth Audio Streaming Device Shipments**




**Annual Bluetooth Phone, Tablet & PC Shipments**

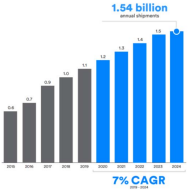



**Annual Bluetooth Entertainments Shipments**

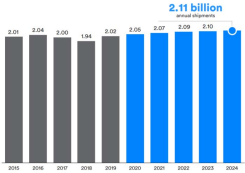


# Total Annual Bluetooth Device Shipments [SIG20]

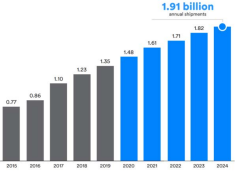
**Annual Bluetooth Audio Streaming Device Shipments**




**Annual Bluetooth Phone, Tablet & PC Shipments**

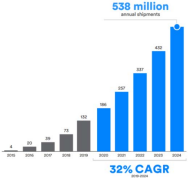



**Annual Bluetooth Entertainments Shipments**





**Annual Bluetooth Location Service Device Shipments**



# Bluetooth IoT Devices and Companion Apps

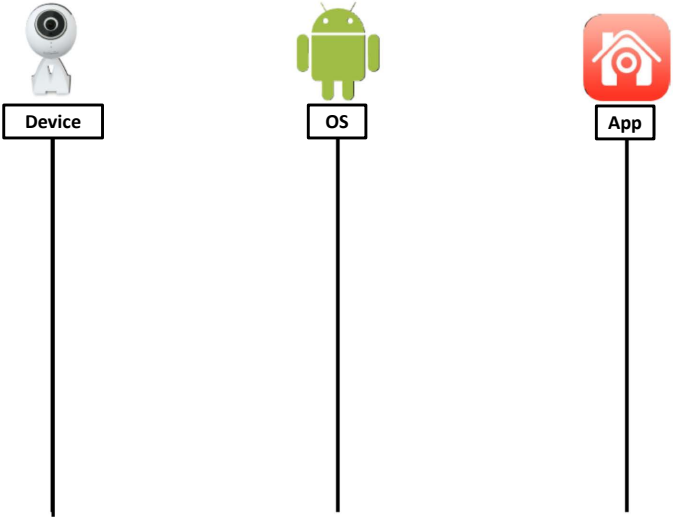

BLE IoT Devices

# Bluetooth IoT Devices and Companion Apps

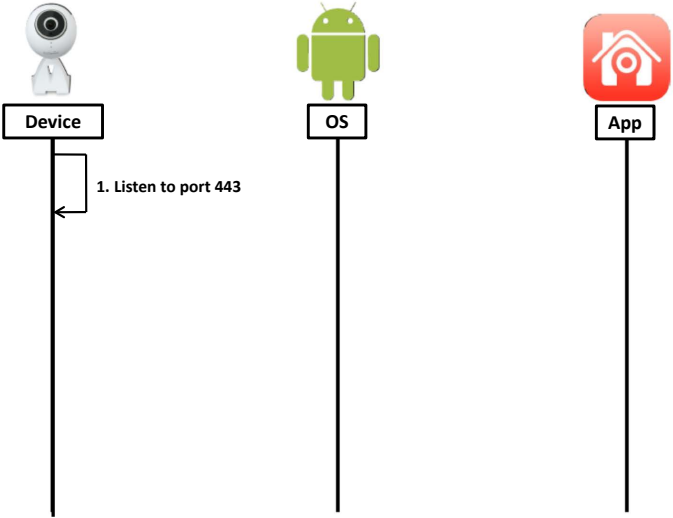
BLE IoT Devices

Companion Mobile Apps

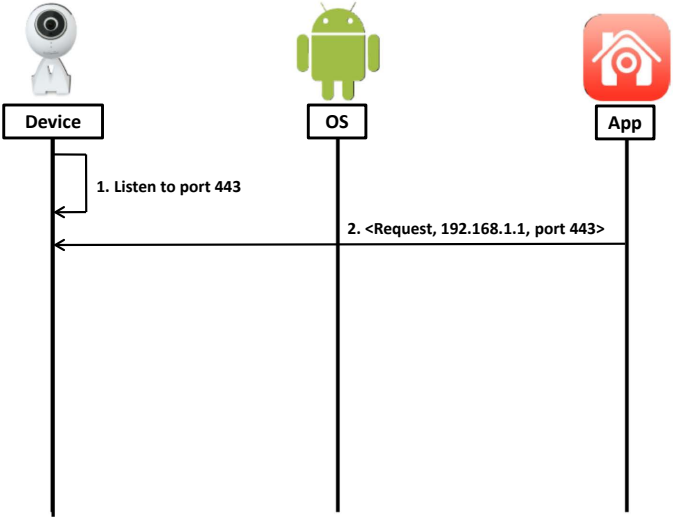
# The General Workflow of Device Communication in TCP/IP Setting



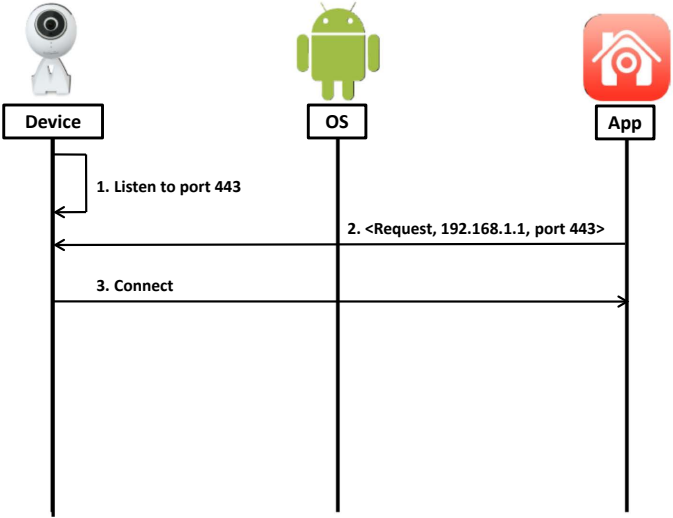
# The General Workflow of Device Communication in TCP/IP Setting



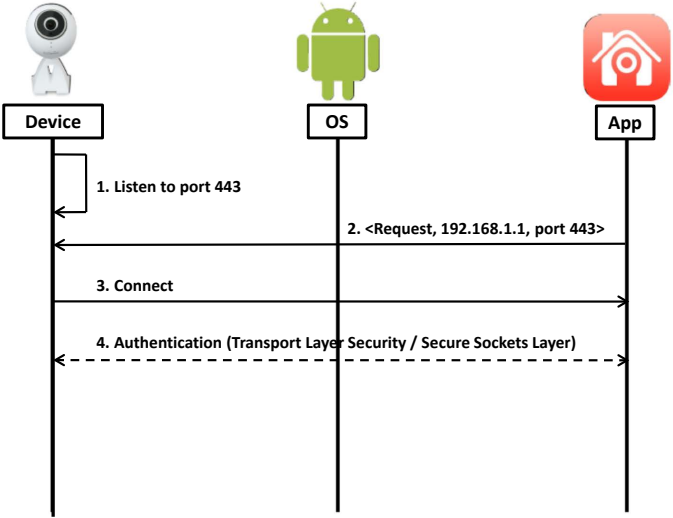
# The General Workflow of Device Communication in TCP/IP Setting



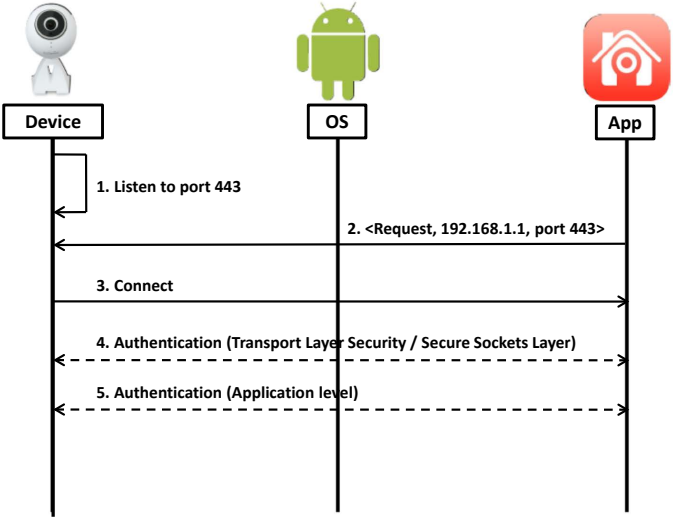
# The General Workflow of Device Communication in TCP/IP Setting



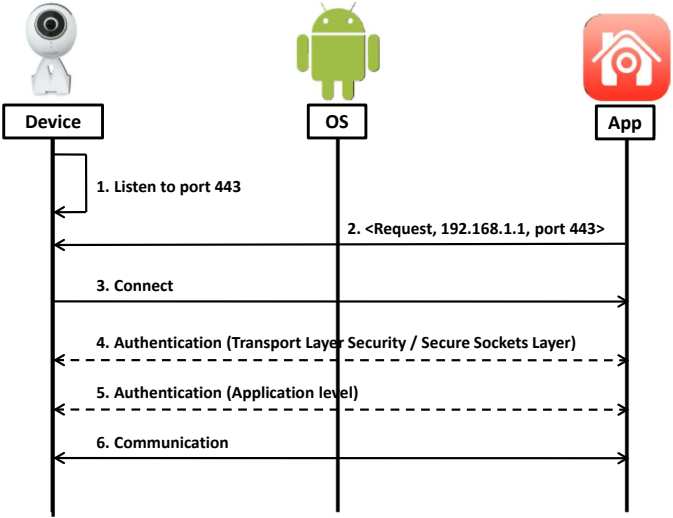
# The General Workflow of Device Communication in TCP/IP Setting



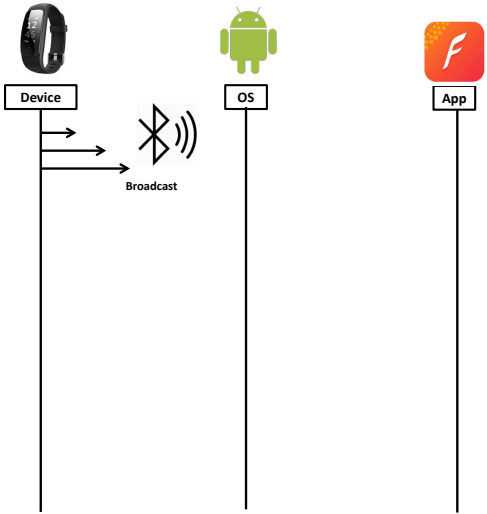
# The General Workflow of Device Communication in TCP/IP Setting



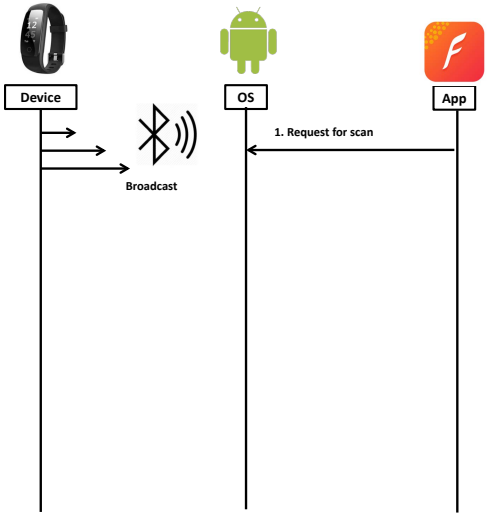
# The General Workflow of Device Communication in TCP/IP Setting



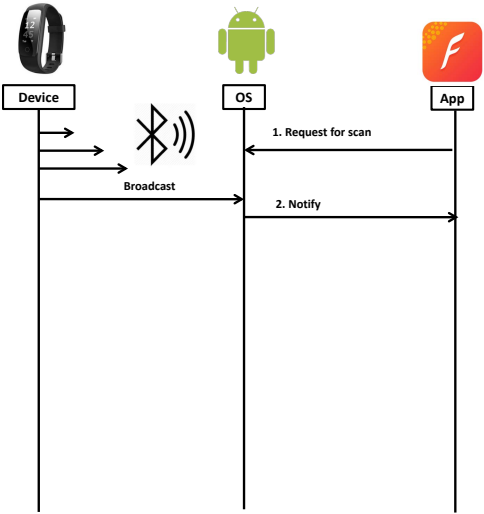
# The General Workflow of BLE IoT Devices and Companion Apps



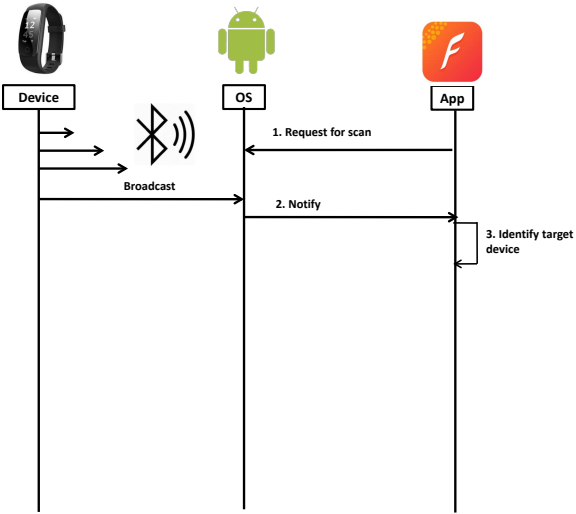
# The General Workflow of BLE IoT Devices and Companion Apps



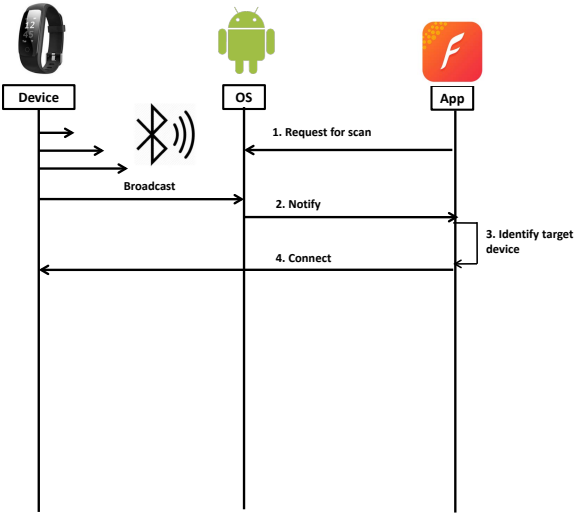
# The General Workflow of BLE IoT Devices and Companion Apps



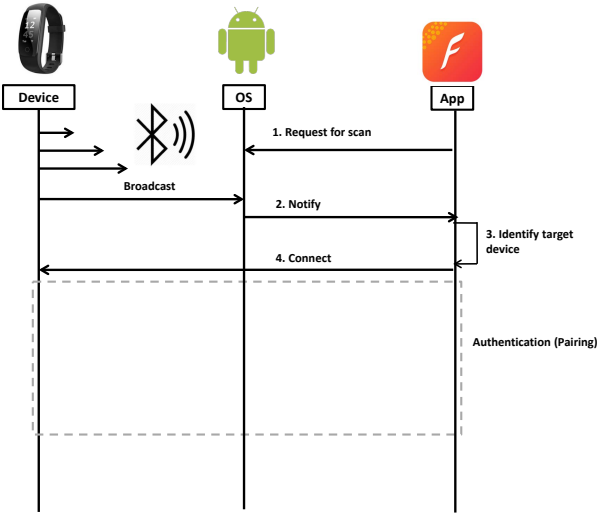
# The General Workflow of BLE IoT Devices and Companion Apps



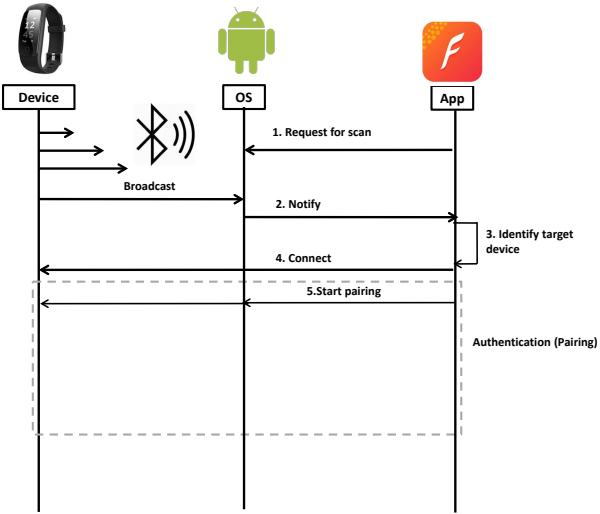
# The General Workflow of BLE IoT Devices and Companion Apps



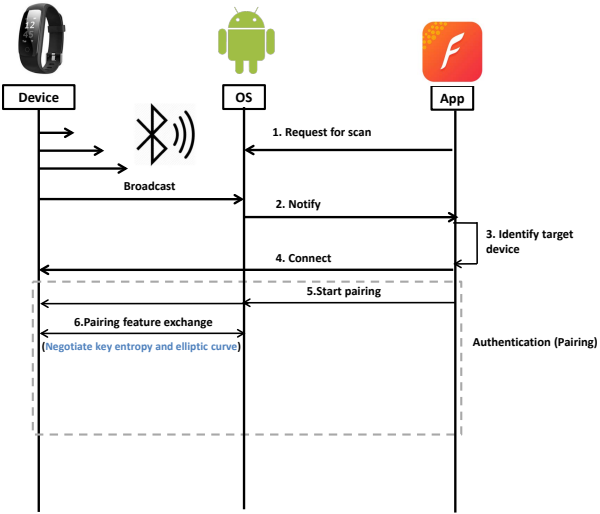
# The General Workflow of BLE IoT Devices and Companion Apps



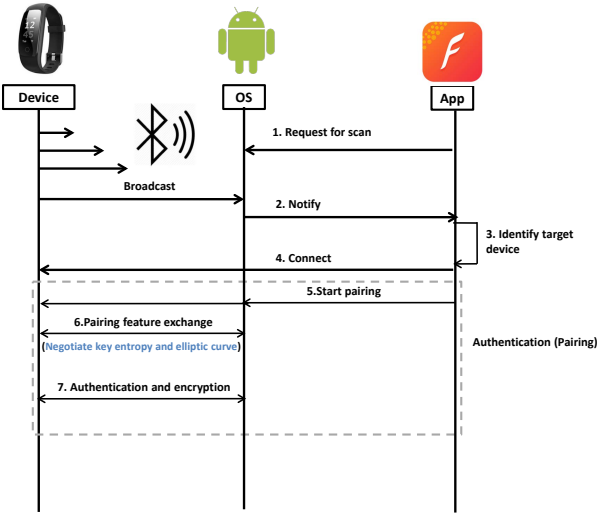
# The General Workflow of BLE IoT Devices and Companion Apps



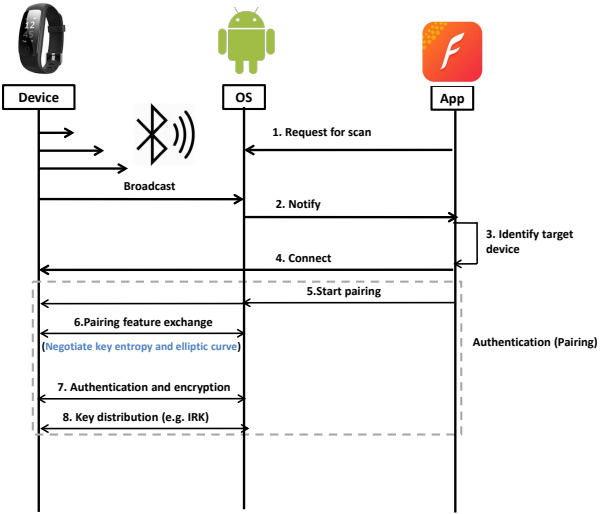
# The General Workflow of BLE IoT Devices and Companion Apps



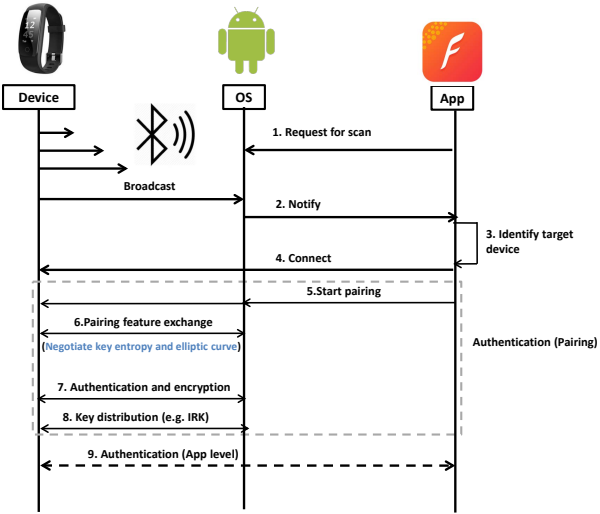
# The General Workflow of BLE IoT Devices and Companion Apps



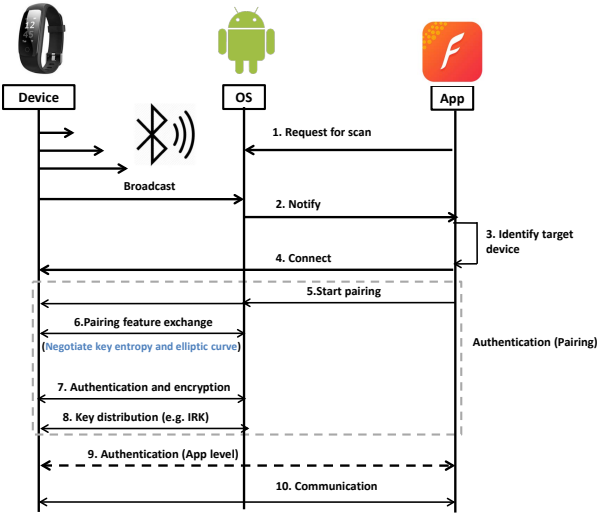
# The General Workflow of BLE IoT Devices and Companion Apps



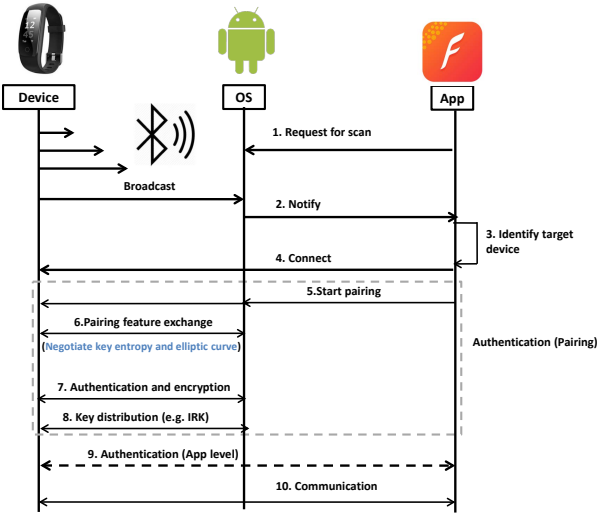
# The General Workflow of BLE IoT Devices and Companion Apps



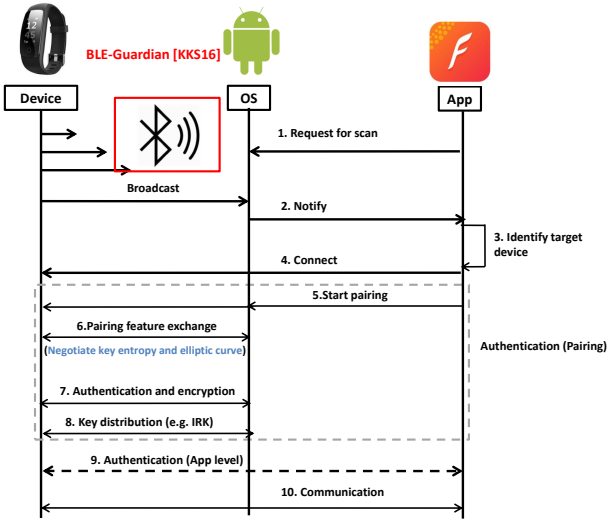
# The General Workflow of BLE IoT Devices and Companion Apps



# The General Workflow of BLE IoT Devices and Companion Apps



# State-of-the-Art

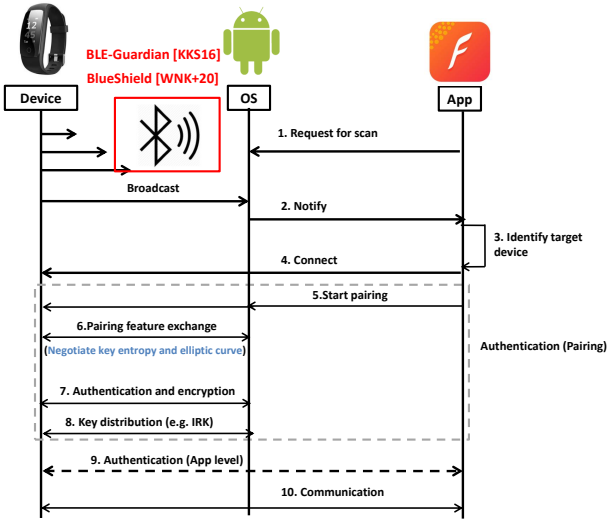



## BLE-Guardian [KKS16]

**Protecting Privacy of BLE Device Users**. In USENIX Security 2016.

- Defending against sensitive information leakage during broadcasting

# State-of-the-Art

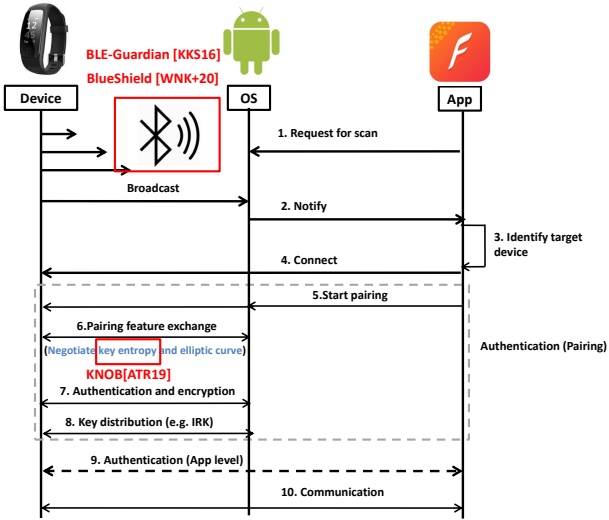



## BlueShield [WNK+20a]

**BlueShield: Detecting Spoofing Attacks in Bluetooth Low Energy Networks**. In RAID 2020.

- Detecting spoofing BLE devices during broadcasting.

# State-of-the-Art

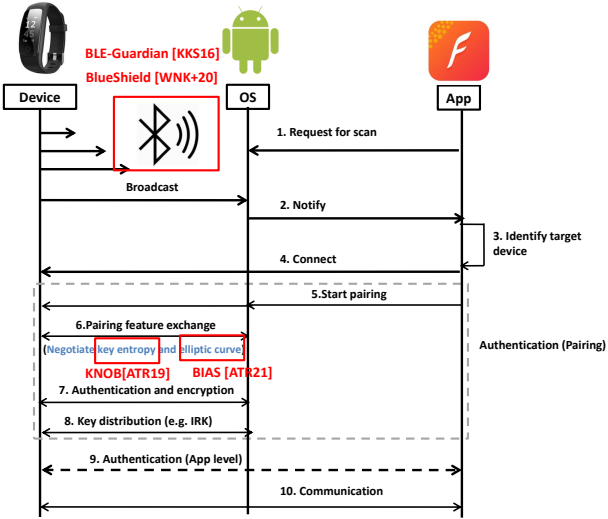



## KNOB [ATR19]

**The KNOB is Broken: Exploiting Low Entropy in the Encryption Key Negotiation Of Bluetooth BR/EDR**. In USENIX Security 2019.

- An attacker forces victims to agree on an encryption key with only one byte of entropy.
- Windows/iOS have fixed it

# State-of-the-Art

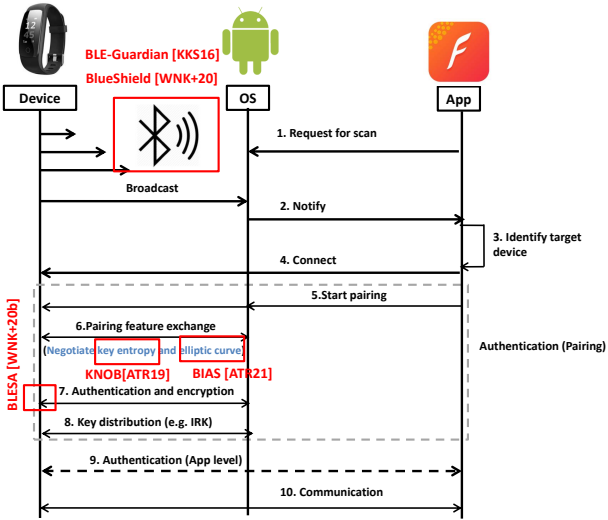



## BIAS [ATR20]

**BIAS: Bluetooth Impersonation AttackS**. In Oakland 2020.

- An attacker forces victims to use P-192 curve instead of using P-256 curve.

# State-of-the-Art

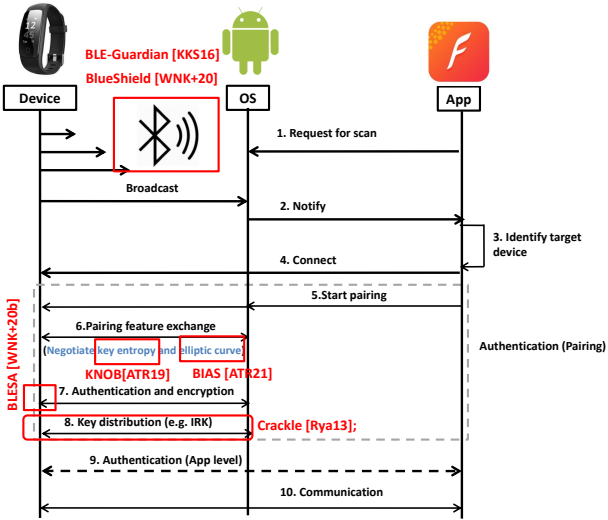



## BLESA [WNK+20b]

**BIESA: Spoofing Attacks against Reconnections in Bluetooth Low Energy**. In WOOT 2020.

- Fake BLE device attacks against mobiles.
- Android and iOS have fixed it

# State-of-the-Art

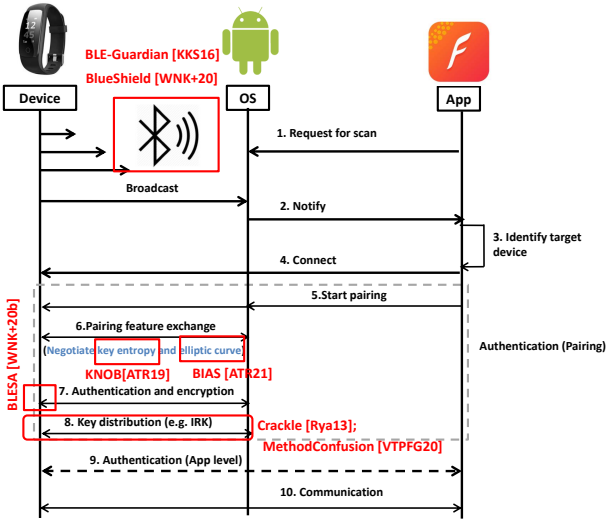



## Crackle [Rya13]

**With Low Energy Comes Low Security**. In WOOT 2013.

- Brute force attacks against long term keys.
- Bluetooth after 4.1 is no longer vulnerable

# State-of-the-Art

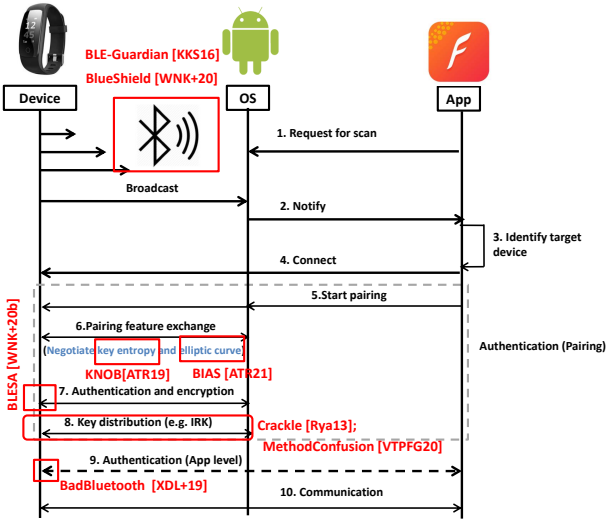



## Bluetooth Method Confusion [VTPFG21]

**Method Confusion Attack on Bluetooth Pairing**. In Oakland 2021

- Man in the middle attack (similar to the active attacks against DH)
- Attackers manipulates the pairing methods and target the ECDH key exchange process.

# State-of-the-Art

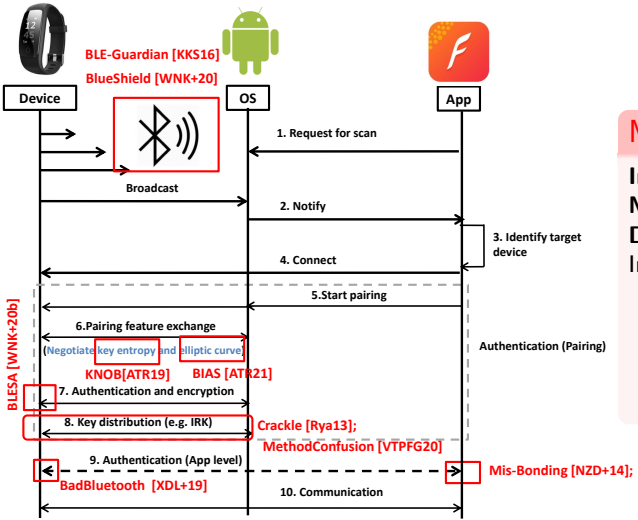



## BadBluetooth [XDL+19]

**Breaking Android Security Mechanisms via Malicious Bluetooth Peripherals**. In NDSS 2019.

- Fake devices manipulate BLE communication due to the lack of app-level authentication.
- Defense is up to the apps

# State-of-the-Art

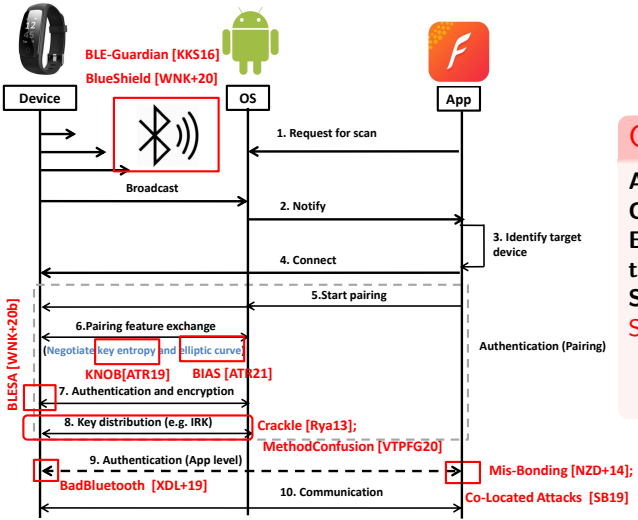



## Mis-Bonding [NZD+14]

**Inside Job: Understanding and Mitigating the Threat of External Device Mis-Bonding on Android.**. In NDSS 2014.

- Malicious apps manipulate BLE communication due to lack of app-level authentication.
- Defense is up to the devices

# State-of-the-Art

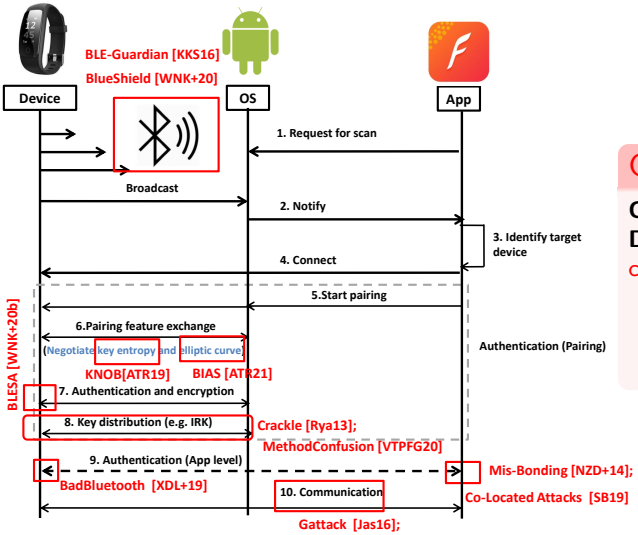



## Co-Located Attacks [SB19]

**A Study of the Feasibility of Co-located App Attacks against BLE and a Large-Scale Analysis of the Current Application-Layer Security Landscape**. In USENIX Security 2019.

- Large-scale analysis of mis-bonding issues.

# State-of-the-Art

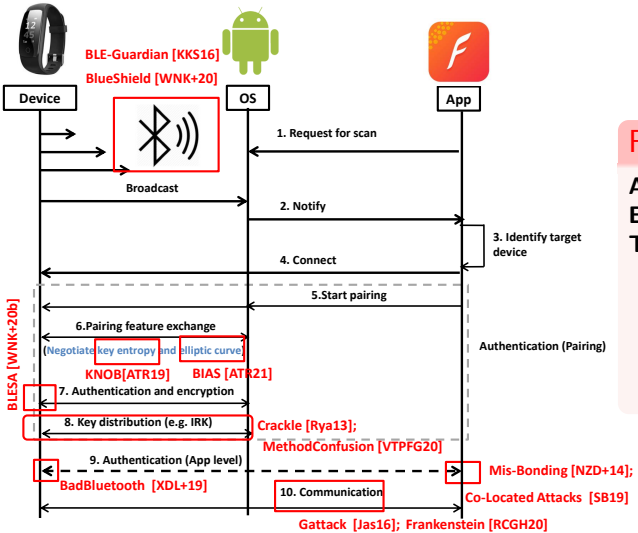



## Gattacking [Jas16]

**Gattacking Bluetooth Smart Devices**. In Black hat USA conference 2016.

- Poorly designed communication protocols are subject to various attacks (e.g., replay attacks).

# State-of-the-Art

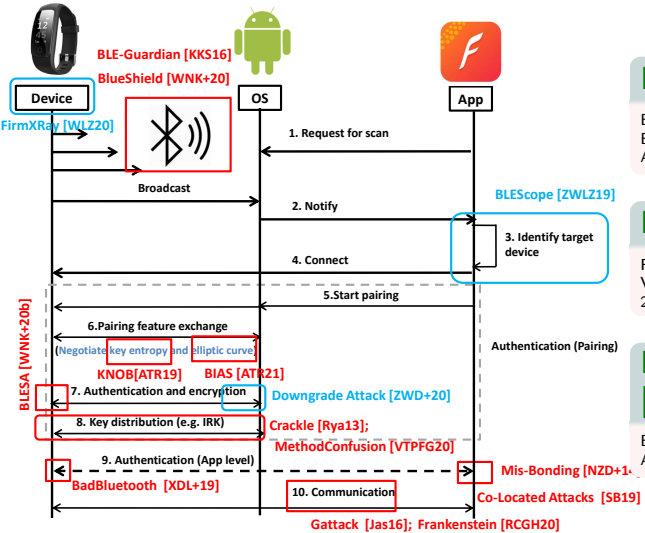



## Frankenstein [RCGH20]

**Advanced Wireless Fuzzing to Exploit New Bluetooth Escalation Targets**. In USENIX Security 2020.

- BLE Fuzzing tool injects HCI traffic or Bluetooth frames into Bluetooth communication in order to uncover Remote Code Execution bugs.

# Our Contributions




## BLEScope [ZWLZ19]

BLEScope: Automatic Fingerprinting of Vulnerable BLE IoT Devices with Static UUIDs from Mobile Apps. In CCS 2019.

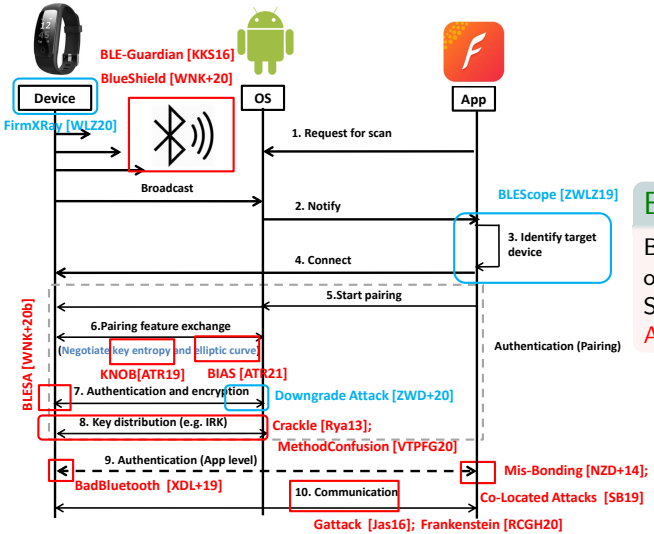
## FirmXRay [WLZ20]

FirmXRay: Detecting Bluetooth Link Layer Vulnerabilities From Bare-Metal Firmware. In CCS 2020.

## Downgrade Attacks [ZWD+20]

Breaking Secure Pairing of BLE Using Downgrade Attacks. In USENIX Security 2020.

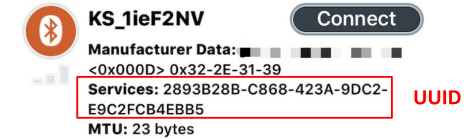
# Our BLEScope [ZWLZ19] Work




## BLEScope [ZWLZ19]

BLEScope: Automatic Fingerprinting of Vulnerable BLE IoT Devices with Static UUIDs from Mobile Apps. In ACM CCS 2019.

# The Key Finding in BLEScope [ZWLZ19]



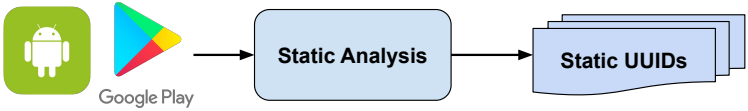

```
public class TemperatureService {
    public static final UUID EVENT_CHAR_UUID;
    public static final UUID PAIR_STATUS_CHAR_UUID;
    public static final UUID REQUEST_CHAR_UUID;
    public static final UUID RESPONSE_CHAR_UUID;
    public static final ParcelUuid SERVICE_PARCEL_UUID;
    public static final UUID SERVICE_UUID;

    static {
        TemperatureService.SERVICE_UUID = UUID.fromString("2893B28B-C868-423A-9DC2-E9C2FCB4EBB5");
        TemperatureService.SERVICE_PARCEL_UUID = new ParcelUuid(TemperatureService.SERVICE_UUID);
        TemperatureService.REQUEST_CHAR_UUID = UUID.fromString("28930000-C868-423A-9DC2-E9C2FCB4EBB5");
        TemperatureService.RESPONSE_CHAR_UUID = UUID.fromString("28930001-C868-423A-9DC2-E9C2FCB4EBB5");
        TemperatureService.EVENT_CHAR_UUID = UUID.fromString("28930002-C868-423A-9DC2-E9C2FCB4EBB5");
        TemperatureService.PAIR_STATUS_CHAR_UUID = UUID.fromString("28930003-C868-423A-9DC2-E9C2FCB4EBB5");
    }
```

UUID

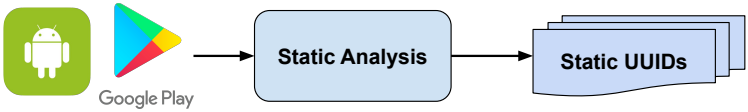
## Key Observation

1. UUIDs are broadcasted by BLE IoT devices to nearby phones.
2. UUIDs are static.
3. Mobile apps contain UUIDs.
4. Mobile apps identify target BLE IoT devices based on their broadcasted UUIDs.

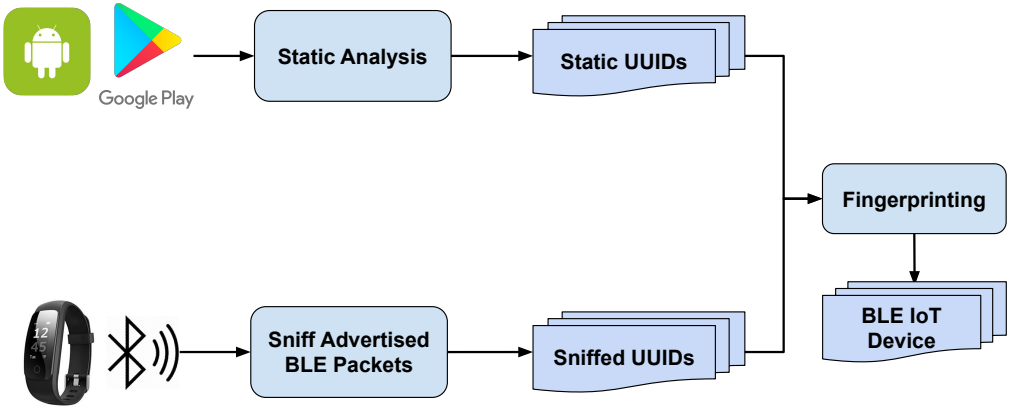
# Attack: How to Fingerprint a BLE IoT Device with Static UUIDs



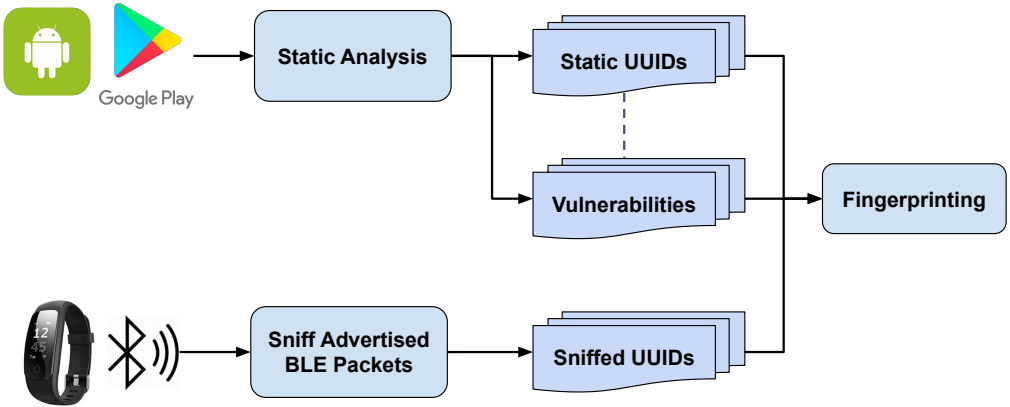
# Attack: How to Fingerprint a BLE IoT Device with Static UUIDs



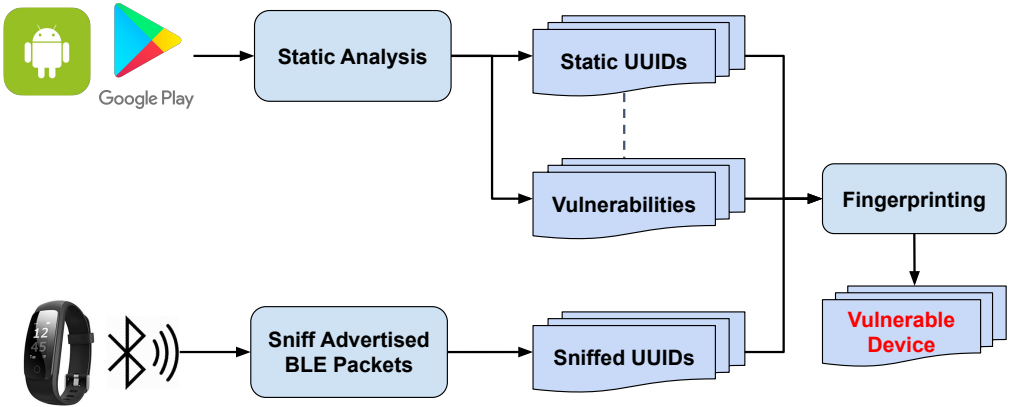

# Attack: How to Fingerprint a BLE IoT Device with Static UUIDs



# Attack: How to Fingerprint a BLE IoT Device with Static UUIDs



# Attack: How to Fingerprint a BLE IoT Device with Static UUIDs



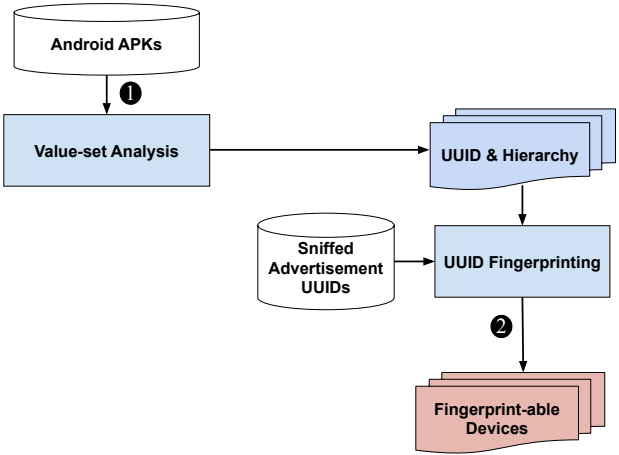
# Introducing BLEScope

"**Automatic Fingerprinting of Vulnerable BLE IoT Devices with Static UUIDs from Mobile Apps**". Chaoshun Zuo, Haohuang Wen, Zhiqiang Lin, and Yinqian Zhang. In *Proceedings of the 26th ACM Conference on Computer and Communications Security (CCS'19)*, London, UK. November 2019

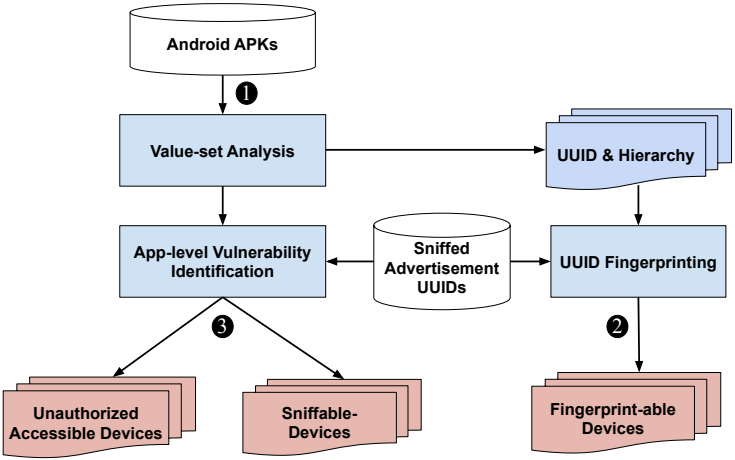
1. **Novel Discovery**. We are the *first* to discover BLE IoT devices can be fingerprinted with static UUIDs.
2. **Effective Techniques**. We have implemented an automatic tool BLEScope to harvest UUIDs and detect vulnerabilities from mobile apps.
3. **Evaluation**. We have tested our tool with $18,166$ BLE mobile apps from Google Play store, and found $168,093$ UUIDs and $1,757$ vulnerable BLE IoT apps.
4. **Countermeasures.** We present channel-level protection, app-level protection, and protocol-level protection (with dynamic UUID generation).

# Overview of BLEScope

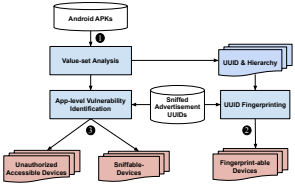
# Overview of BLEScope



# Overview of BLEScope



# Overview of BLEScope




## Challenges

1. How to extract UUIDs from mobile apps
2. How to reconstruct UUID hierarchy
3. How to identify flawed vulnerable authentication apps

## Solutions: Using **Automated Program Analysis**

1. Resolving UUIDs using context and **value-set analysis**
2. Reconstructing UUID hierarchy with **control dependence**
3. Identifying flawed authentication with **data dependence**

# Results from Google Play Store

## IoT Mobile App Collection

1. We downloaded 2 million mobile apps from Google Play as of April 2019.
2. We identified BLE IoT apps by searching for after-connection BLE APIs.
3. 18,166 BLE IoT apps are found for our analysis

# Results from Google Play Store

## IoT Mobile App Collection

1. We downloaded 2 million mobile apps from Google Play as of April 2019.
2. We identified BLE IoT apps by searching for after-connection BLE APIs.
3. 18,166 BLE IoT apps are found for our analysis

| Item | Value | % |
|---|---|---|
| # Apps Support BLE | 18,166 | 100.0 |
| # "Just Works" Pairing | 11,141 | 61.3 |
| # Vulnerable Apps | 1,757 | 15.8 |
| # Absent Cryptographic Usage | 1,510 | 13.6 |
| # Flawed Authentication | 1,434 | 12.9 |

Table: Insecure app identification result.

# Results from Google Play Store

## IoT Mobile App Collection

1. We downloaded 2 million mobile apps from Google Play as of April 2019.
2. We identified BLE IoT apps by searching for after-connection BLE APIs.
3. 18,166 BLE IoT apps are found for our analysis

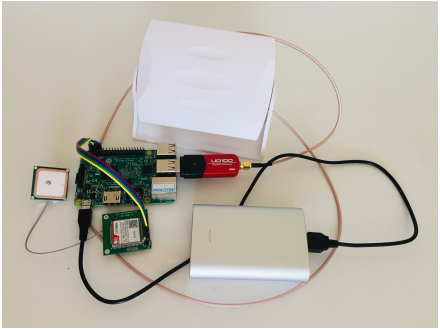
| Item | Value | % |
|---|---|---|
| # Apps Support BLE | 18,166 | 100.0 |
| # "Just Works" Pairing | 11,141 | 61.3 |
| # Vulnerable Apps | 1,757 | 15.8 |
| # Absent Cryptographic Usage | 1,510 | 13.6 |
| # Flawed Authentication | 1,434 | 12.9 |

Table: Insecure app identification result.

| Category | # App | "Just Works" | Absent Crypto | Flawed Auth. |
|---|---|---|---|---|
| Health & Fitness | 3,849 | 2,639 | 221 | 207 |
| Tools | 2,833 | 1,895 | 385 | 362 |
| Lifestyle | 2,173 | 1,081 | 147 | 141 |
| Business | 1,660 | 972 | 90 | 85 |
| Travel & Local | 967 | 582 | 90 | 87 |

Table: Top 5 category of the IoT apps.

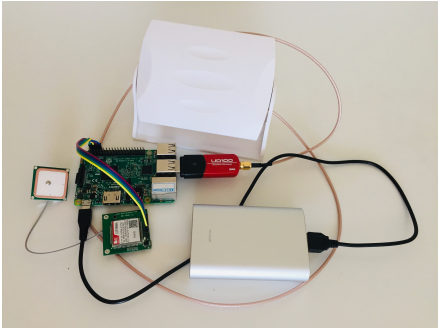
# Results from Our Field Test



## BLE Sniffer

- Raspberry-Pi
- Parani-UD100 (Bluetooth adapter)
- Antenna RP-SMA-R/A (**1km** range)
- SIM7000A GPS module (GPS sensor)

# Results from Our Field Test



# Results from Our Field Test

| Item | Value | % |
|---|---|---|
| # Unique BLE Device | 30,862 | |
| # Unique BLE Device w. UUID | 5,822 | 18.9 |
| # Fingerprintable | 5,509 | 94.6 |
| # Vulnerable | 431 | 7.4 |
| # Sniffable | 369 | 6.7 |
| # Unauthorized Accessible | 342 | 6.2 |

Table: Experimental result of our field test.

# Results from Our Field Test

| Company Name | # Devices |
|---|---|
| Google | 2,436 |
| Tile, Inc. | 441 |
| - | 243 |
| - | 208 |
| Logitech International SA | 131 |
| Nest Labs Inc. | 114 |
| Google | 92 |
| Hewlett-Packard Company | 74 |
| - | 46 |
| - | 44 |
| - | 44 |

Table: Top 10 devices in the field test.

# Results from Our Field Test

| Company Name | # Devices |
|---|---|
| Google | 2,436 |
| Tile, Inc. | 441 |
| - | 243 |
| - | 208 |
| Logitech International SA | 131 |
| Nest Labs Inc. | 114 |
| Google | 92 |
| Hewlett-Packard Company | 74 |
| - | 46 |
| - | 44 |
| - | 44 |

Table: Top 10 devices in the field test.

# Results from Our Field Test

| Device Description | # Device |
|---|---|
| Digital Thermometer | 7 |
| Car Dongle | 6 |
| Key Finder A | 6 |
| Smart Lamp | 5 |
| Key Finder B | 5 |
| Smart Toy A | 4 |
| Smart VFD | 4 |
| Air Condition Sensor | 4 |
| Smart Toy B | 4 |
| Accessibility Device | 4 |

Table: Top 10 **vulnerable** devices.

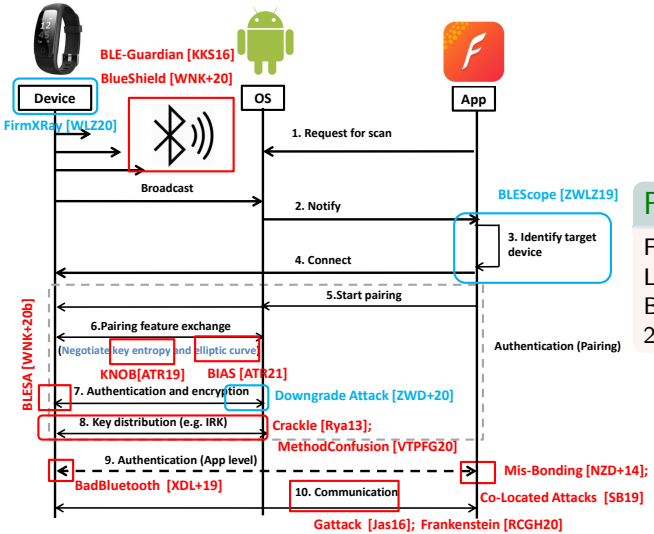
# Results from Our Field Test

| Device Description | # Device |
|---|---|
| Digital Thermometer | 7 |
| Car Dongle | 6 |
| Key Finder A | 6 |
| Smart Lamp | 5 |
| Key Finder B | 5 |
| Smart Toy A | 4 |
| Smart VFD | 4 |
| Air Condition Sensor | 4 |
| Smart Toy B | 4 |
| Accessibility Device | 4 |

Table: Top 10 **vulnerable** devices.

# Results from Our Field Test

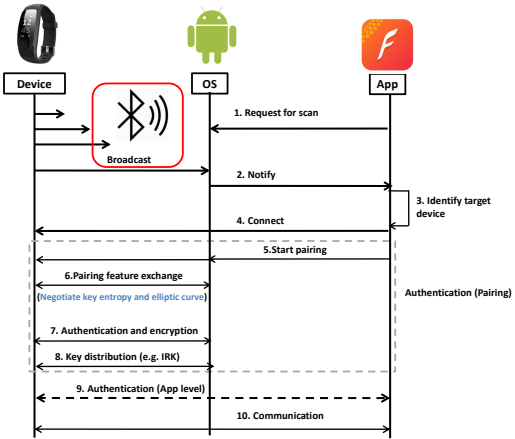
# Our FirmXRay [WLZ20] Work




## FirmXRay [WLZ20]

FirmXRay: Detecting Bluetooth Link Layer Vulnerabilities From Bare-Metal Firmware. In ACM CCS 2020.

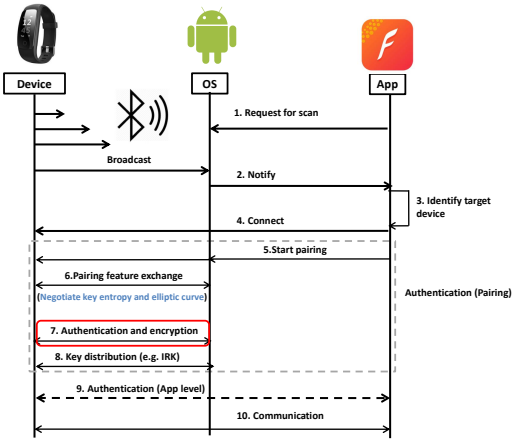
# BLE Link Layer Vulnerabilities




## Vulnerabilities

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

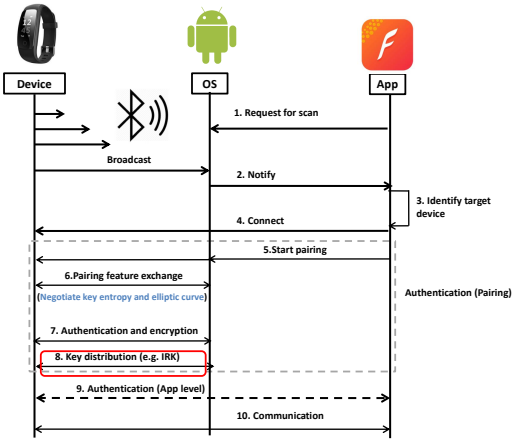
# BLE Link Layer Vulnerabilities




## Vulnerabilities

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

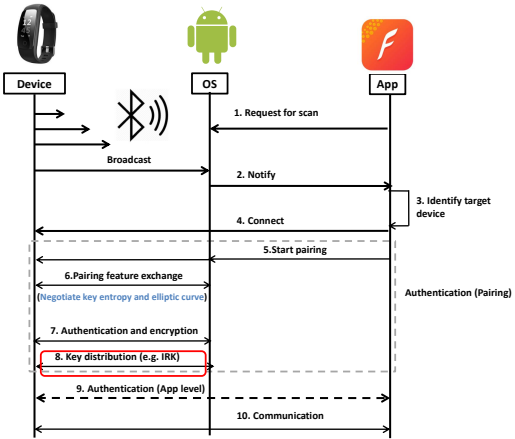
# BLE Link Layer Vulnerabilities




## 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



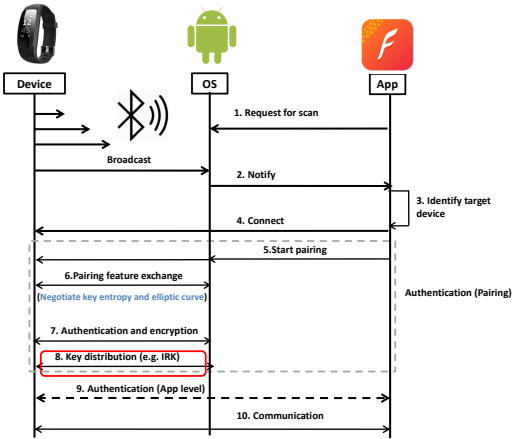

## Vulnerabilities

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

**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
```

**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 |
|---|---|---|---|

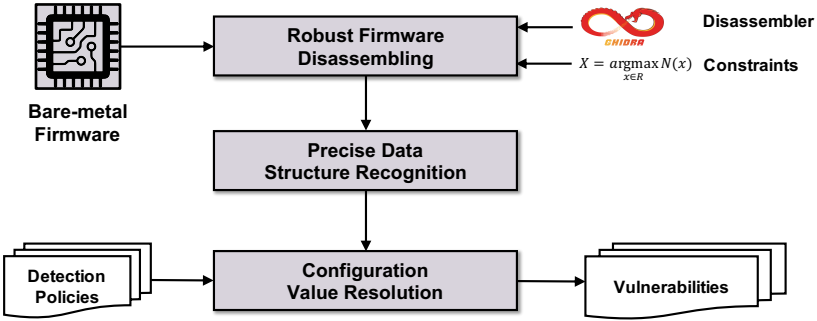
```
           // BOND  = 1, MITM  = 0
           // IO    = 3, OOB   = 0
20003269   uint8 min_key_size
20003270   uint8 max_key_size
20003271   ble_gap_sec_kdist_t  kdist_own
20003275   ble_gap_sec_kdist_t  kdist_peer
```

Value computation

**Register Values**

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

# FirmXRay Overview



# Robust Firmware Disassembling

**Correct Base**
**0x1B000**

```
20452   ldr     r0, pc+0x72
20454   blx     r0=>0x22A90
...
204c4   0x22A90
...
        Function Foo()
22a90   push    {r3, r4, r5, lr}
```

**(1) Absolute Function Pointer**

```
1fe52   ldr     r0, pc+0x146
1fe54   ldmia   r0, {r4, r5, r6}
...
1ff98   0x23058
...
23058   "KinsaHealth"
```

**(2) Absolute String Pointer**

# Robust Firmware Disassembling

**Incorrect 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"
```

**Correct Base**
**0x1B000**

```
20452   ldr     r0, pc+0x72
20454   blx     r0=>0x22A90
...
204c4   0x22A90
...
        Function Foo()
22a90   push    {r3, r4, r5, lr}
```

**(1) Absolute Function Pointer**

```
1fe52   ldr     r0, pc+0x146
1fe54   ldmia   r0, {r4, r5, r6}
...
1ff98   0x23058
...
23058   "KinsaHealth"
```

**(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"
```

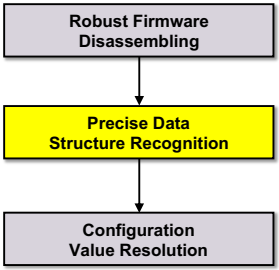
**Absolute Pointers:** 0x22A90, 0x23058

**Gadgets:** 0x07A90, 0x08058

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

N(0x1B000) = 2

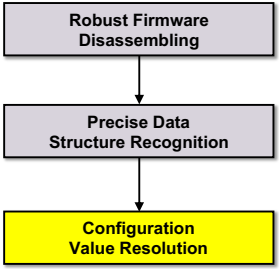
# Precise Data Structure Recognition




**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

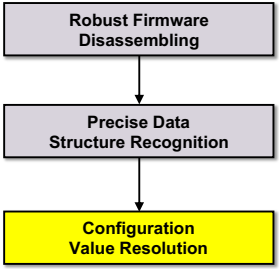



**Read Only Memory**

```
1  243a8   mov   r2, #0x0
2  243aa   orr   r2, #0x1
3  243ac   and   r2, #0xe1
4  243ae   add   r2, #0xc
5  243b0   and   r2, #0xdf
6  243b2   ldr   r1, [0x260c8]
7  243b4   str   r2, [r1,#0x0]
...
8  25f44   ldr   r2, [0x260c8]
9  25f46   mov   r1, #0x0
10 25f48   svc   0x7f
// SD_BLE_GAP_SEC_PARAMS_REPLY
...
11 260c8   0x20003268
           // ble_gap_sec_parms_t*
```

**Program Path**

# Configuration Value Resolution



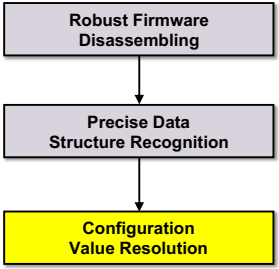

**Read Only Memory**

```
1  243a8   mov   r2, #0x0
2  243aa   orr   r2, #0x1
3  243ac   and   r2, #0xe1
4  243ae   add   r2, #0xc
5  243b0   and   r2, #0xdf
6  243b2   ldr   r1, [0x260c8]
7  243b4   str   r2, [r1,#0x0]
...
8  25f44   ldr   r2, [0x260c8]
9  25f46   mov   r1, #0x0
10 25f48   svc   0x7f
// SD_BLE_GAP_SEC_PARAMS_REPLY
...
11 260c8   0x20003268
           // ble_gap_sec_parms_t*
```

**Program Path**

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

# Configuration Value Resolution



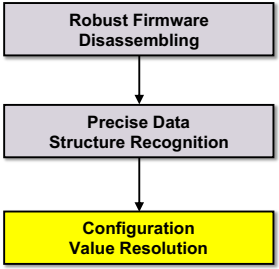

**Read Only Memory**

```
1  243a8   mov   r2, #0x0
2  243aa   orr   r2, #0x1
3  243ac   and   r2, #0xe1
4  243ae   add   r2, #0xc
5  243b0   and   r2, #0xdf
6  243b2   ldr   r1, [0x260c8]
7  243b4   str   r2, [r1,#0x0]
...
8  25f44   ldr   r2, [0x260c8]
9  25f46   mov   r1, #0x0
10 25f48   svc   0x7f
// SD_BLE_GAP_SEC_PARAMS_REPLY
...
11 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**

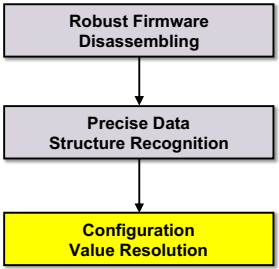
```
243a8   mov   r2, #0x0
243aa   orr   r2, #0x1
243ac   and   r2, #0xe1
243ae   add   r2, #0xc
243b0   and   r2, #0xdf
243b2   ldr   r1, [0x260c8]
243b4   str   r2, [r1,#0x0]
...
25f44   ldr   r2, [0x260c8]
25f46   mov   r1, #0x0
25f48   svc   0x7f
// SD_BLE_GAP_SEC_PARAMS_REPLY
...
260c8   0x20003268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

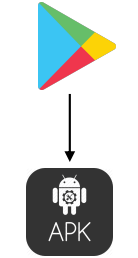
# Configuration Value Resolution




| 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 |

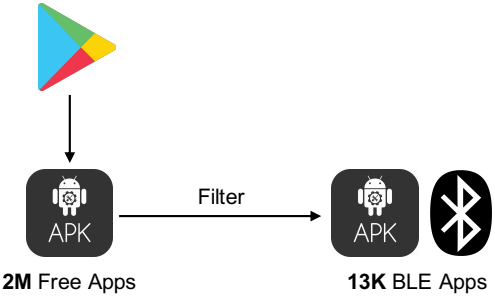
# Firmware Collection

# Firmware Collection

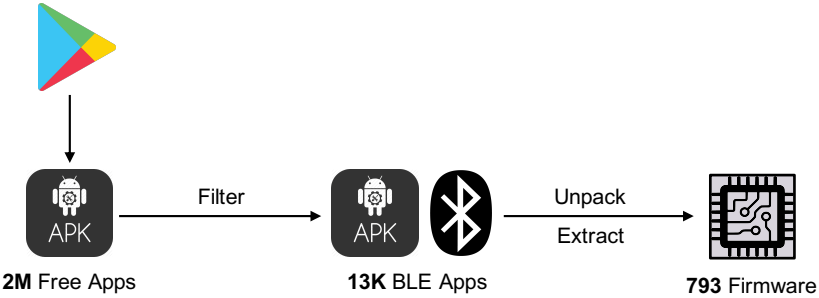



**2M** Free Apps

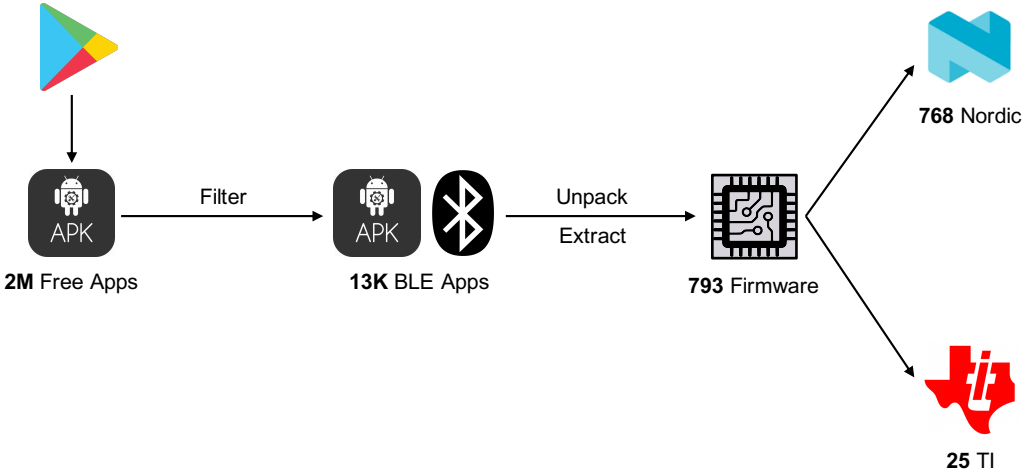
# Firmware Collection



# 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

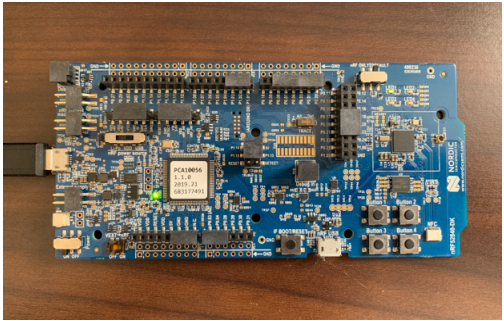
## Passive MITM Vulnerability Identification

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 | ✓ | | ✓ |

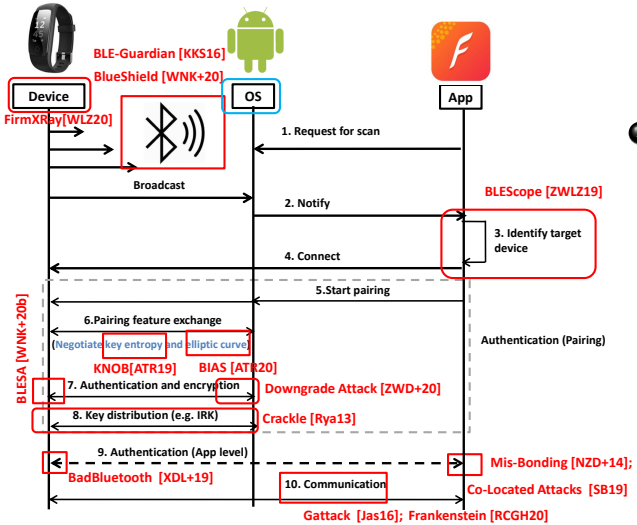
A2: Unauthorized Control

# 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 | ✓ | | ✓ |

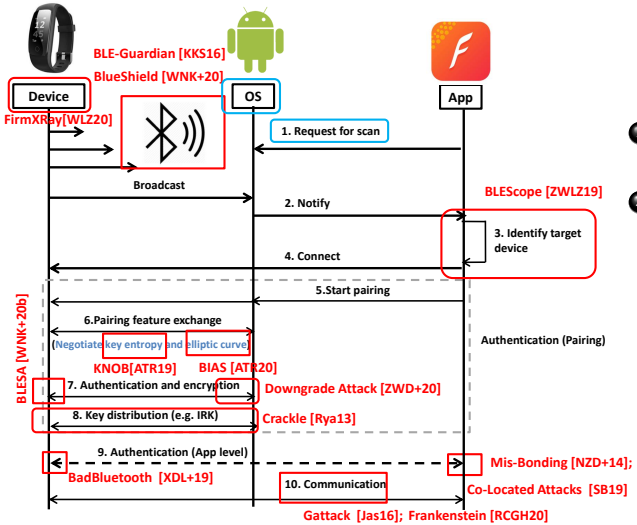
A3: Sensitive Data Eavesdropping

# Near Term



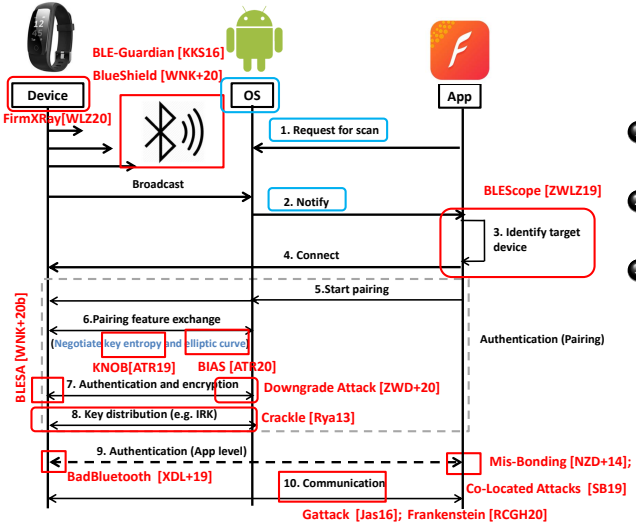

❶ **OS Defense:** OS-level defense to patch multiple security issues.

# Near Term



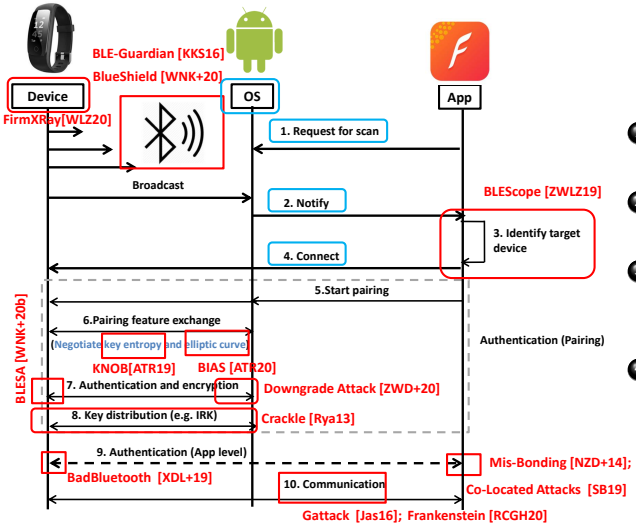

1. **OS Defense:** OS-level defense to patch multiple security issues.
2. **Scanning Defense:** Defending against malicious scanning.

# Near Term




1. **OS Defense:** OS-level defense to patch multiple security issues.
2. **Scanning Defense:** Defending against malicious scanning.
3. **Notification Fingerprinting:** Exploring notification fingerprinting against BLE devices.

# Near Term




1. **OS Defense:** OS-level defense to patch multiple security issues.
2. **Scanning Defense:** Defending against malicious scanning.
3. **Notification Fingerprinting:** Exploring notification fingerprinting against BLE devices.
4. **Connection Security:** Exploring a defense for jamming attacks.

# Other Directions

1. **Other New Security Features.** New security features (e.g., Cross-Transport Key Derivation) are keeping introducing, bringing new security attack surfaces.
2. **Privacy-preserving Protocols**. BLE Privacy-preserving protocols such as identity resolution protocol may be vulnerable, and further understanding is needed.

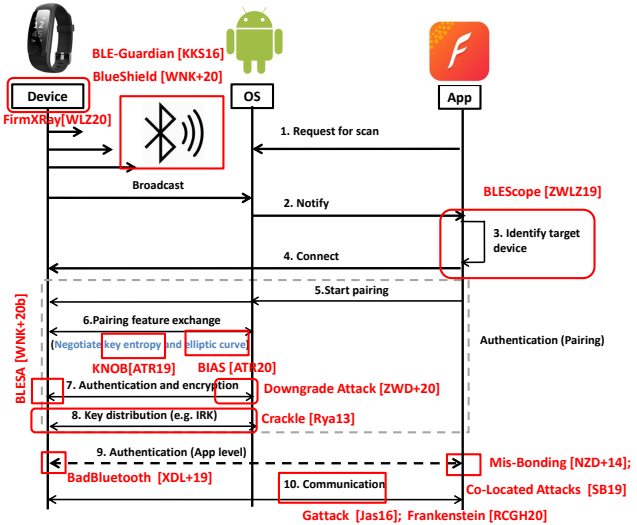
# Other Directions

1. **Other New Security Features.** New security features (e.g., Cross-Transport Key Derivation) are keeping introducing, bringing new security attack surfaces.
2. **Privacy-preserving Protocols**. BLE Privacy-preserving protocols such as identity resolution protocol may be vulnerable, and further understanding is needed.

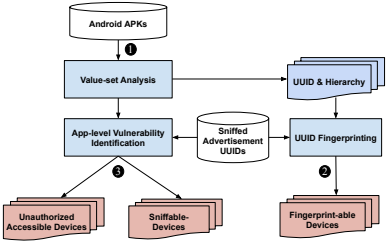
## Recent Papers of Bluetooth Research with COVID-19

1. Qingchuan Zhao, Haohuang Wen, Zhiqiang Lin, Dong Xuan, and Ness Shroff. **On the Accuracy of Measured Distances of Bluetooth-based Contact Tracing** (short paper). In SECURECOMM'20, October 2020.
2. Haohuang Wen, Qingchuan Zhao, Zhiqiang Lin, Dong Xuan, and Ness Shroff. **A Study of the Privacy of COVID-19 Contact Tracing Apps**. In SECURECOMM'20, October 2020.

# The Landscape of Bluetooth Security and Privacy



# BLEScope [CCS 2019]



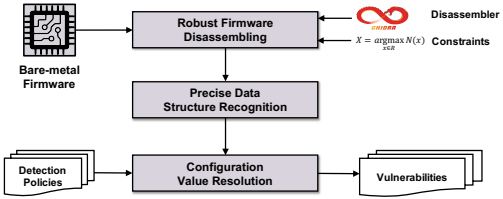

## BLEScope

- Automatic UUID extraction and hierarchy reconstruction from mobile apps
- Identify app-level vulnerabilities by directly analyzing mobile apps

## App Analysis and Field Test Result

- We analyzed 18,166 apps and discovered 168,093 UUIDs and 1,757 vulnerable apps
- 5,822 BLE devices were discovered in the field test, and 94.6% can be fingerprinted

# FirmXRay [CCS 2020]



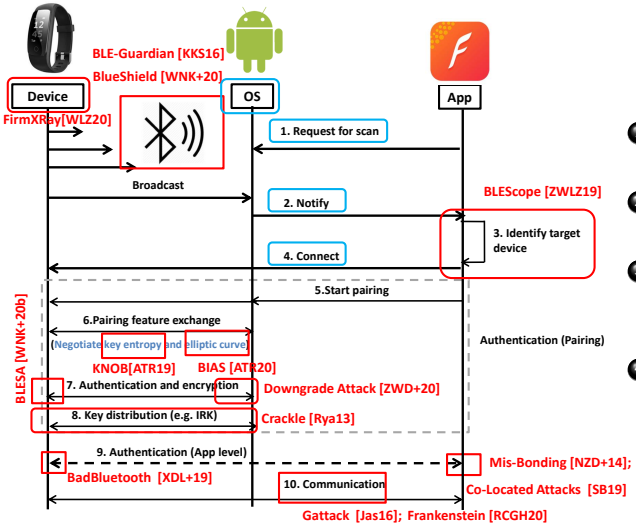


## BLEScope

- 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.

# Future Directions




1. **OS Defense:** OS-level defense to may patch multiple security issues.
2. **Scanning Defense:** Defending against malicious scanning.
3. **Notification Fingerprinting:** Exploring notification fingerprinting against BLE devices.
4. **Connection Security:** Exploring a defense for jamming attacks.

# Thank You

## Automatically Uncovering Vulnerabilities in Bluetooth Applications from Binary Code

Zhiqiang Lin
zlin@cse.ohio-state.edu

Joint work w/ Haohuang Wen, Yinqian Zhang, and Chaoshun Zuo

12/04/2020

# References I


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*, 28th {USENIX} Security Symposium ({USENIX} Security 19), 2019, pp. 1047–1061.

Daniele Antonioli, Nils Ole Tippenhauer, and Kasper Rasmussen, *Bias: Bluetooth impersonation attacks*, Proceedings of the IEEE Symposium on Security and Privacy (S&P), 2020.

*Bluetooth specification version 4.2*, https://www.bluetooth.org/DocMan/handlers/DownloadDoc.ashx?doc_id=286439, 2014.

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.

Sławomir Jasek, *Gattacking bluetooth smart devices*, Black hat USA conference, 2016.

Fawaz Kassem, Kyu-Han Kim, and Kang G Shin, *Protecting privacy of {BLE} device users*, 25th {USENIX} Security Symposium ({USENIX} Security 16), 2016, pp. 1205–1221.

Muhammad Naveed, Xiao-yong Zhou, Soteris Demetriou, XiaoFeng Wang, and Carl A Gunter, *Inside job: Understanding and mitigating the threat of external device mis-binding on android.*, NDSS, 2014.

Jan Ruge, Jiska Classen, Francesco Gringoli, and Matthias Hollick, *Frankenstein: Advanced wireless fuzzing to exploit new bluetooth escalation targets*, arXiv preprint arXiv:2006.09809 (2020).

Mike Ryan, *Bluetooth: With low energy comes low security*, 7th {USENIX} Workshop on Offensive Technologies ({WOOT} 13), 2013.

Pallavi Sivakumaran and Jorge Blasco, *A study of the feasibility of co-located app attacks against {BLE} and a large-scale analysis of the current application-layer security landscape*, 28th {USENIX} Security Symposium ({USENIX} Security 19), 2019, pp. 1–18.

# References II


Bluetooth SIG, *Market update 2020*, [EB/OL], 2020, https://www.bluetooth.com/wp-content/uploads/2020/03/2020_Market_Update-EN.pdf Accessed July, 2020.

Maximilian Von Tschirschnitz, Ludwig Peuckert, Fabian Franzen, and Jens Grossklags, *Method confusion attack on bluetooth pairing*, Proceedings of the IEEE Symposium on Security and Privacy (S&P) (2021).

Haohuang Wen, Zhiqiang Lin, and Yinqian Zhang, *Firmxray: Detecting bluetooth link layer vulnerabilities from bare-metal firmware*, Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, 2020, pp. 167–180.

Jianliang Wu, Yuhong Nan, Vireshwar Kumar, Mathias Payer, and Dongyan Xu, *Blueshield: Detecting spoofing attacks in bluetooth low energy networks*.

Jianliang Wu, Yuhong Nan, Vireshwar Kumar, Dave Jing Tian, Antonio Bianchi, Mathias Payer, and Dongyan Xu, *Blesa: Spoofing attacks against reconnections in bluetooth low energy*.

Fenghao Xu, Wenrui Diao, Zhou Li, Jiongyi Chen, and Kehuan Zhang, *Badbluetooth: Breaking android security mechanisms via malicious bluetooth peripherals.*, NDSS, 2019.

Yue Zhang, Jian Weng, Rajib Dey, Yier Jin, Zhiqiang Lin, and Xinwen Fu, *Breaking secure pairing of bluetooth low energy using downgrade attacks*, 28th {USENIX} Security Symposium ({USENIX} Security 20), 2020.

Chaoshun Zuo, Haohuang Wen, Zhiqiang Lin, and Yinqian Zhang, *Automatic fingerprinting of vulnerable ble iot devices with static uuids from mobile apps*, Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, 2019, pp. 1469–1483.