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
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.

Bluetooth Prototype

1994
History of Bluetooth

- **1994**: Introduction of Bluetooth Prototype
- **1998**: Formation of Bluetooth SIG
History of Bluetooth

- Bluetooth Prototype
  - 1994
- Bluetooth SIG
  - 1994
- Bluetooth 1.0+1.0b
  - 1998
  - Many bugs
  - Bluetooth Basic Rate (~700KB/s)
- 1999

Timeline:

- 1994: Bluetooth Prototype
- 1998: Bluetooth 1.0+1.0b
- 1999: Bluetooth 1.0+1.0b

References:
[ZWLZ19]
[WLZ20]
History of Bluetooth

- 1994: Bluetooth Prototype
- 1998: Bluetooth 1.0+1.0b
- 1999: Bluetooth 1.1
- 2001: (~700KB/s)

- Bluetooth SIG
- (~700KB/s)

- Fixed security issues.
- First marketable product version.
History of Bluetooth

- **1994**: Bluetooth Prototype
- **1998**: Bluetooth 1.0+1.0b (~700KB/s)
- **1999**: Bluetooth 1.1 (~700KB/s)
- **2001**: Bluetooth 2.0/2.1 +EDR (~2.1MB/s)
- **2004**: Bluetooth SIG
- **2009**: Bluetooth 2.0/2.1 +EDR

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

- **1994**: Bluetooth Prototype
- **1998**: Bluetooth SIG
- **1999**: Bluetooth 1.0+1.0b
- **2001**: Bluetooth 1.1 (~700KB/s)
- **2004**: Bluetooth 2.0/2.1 +EDR (~2.1MB/s)
- **2009**: Introduced High-speed channel (~24MB/s)
- **2009**: Bluetooth 3.0+HS (~24MB/s)
History of Bluetooth

1994: Bluetooth Prototype
1998: Bluetooth SIG
1999: Bluetooth 1.0+
2001: Bluetooth 1.1 (~700KB/s)
2004: Bluetooth 3.0+HS (~24MB/s)
2009: Bluetooth 2.0/2.1 +EDR (~2.1MB/s)
2010: Bluetooth 3.0+HS (~24MB/s)

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

- **1994**: Bluetooth Prototype
- **1998**: Bluetooth SIG
- **1999**: Bluetooth 1.1
- **2001**: Bluetooth 1.0+1.0b
- **2004**: Bluetooth 2.0/2.1 +EDR
- **2009**: Bluetooth 3.0+HS
- **2010**: Bluetooth 4.0
- **2013**: Bluetooth 4.1

- **(~700KB/s)**
- **(~2.1MB/s)**
- **(~700KB/s)**
- **(~24MB/s)**
- **(~24MB/s)**

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

- **1994**: Bluetooth Prototype
  - (~700KB/s)

- **1998**: Bluetooth SIG
  - (~700KB/s)

- **1999**: Bluetooth 1.0+1.0b
  - (~700KB/s)

- **2001**: Bluetooth 1.1
  - (~2.1MB/s)

- **2004**: Bluetooth 2.0/2.1 +EDR
  - (~24MB/s)

- **2009**: Bluetooth 3.0+HS
  - (~24MB/s)

- **2010**: Bluetooth 4.0
  - Low energy (LE) protocol for IoT;
    - 128-bit encryption/LE Privacy and Whitelisting

- **2013**: Bluetooth 4.1
  - (~24MB/s)

- **2014**: Bluetooth 4.2
History of Bluetooth

<table>
<thead>
<tr>
<th>Year</th>
<th>Bluetooth Version</th>
<th>Data Transfer Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1994</td>
<td>Bluetooth Prototype</td>
<td>(~700KB/s)</td>
</tr>
<tr>
<td>1998</td>
<td>Bluetooth SIG</td>
<td></td>
</tr>
<tr>
<td>1999</td>
<td>Bluetooth 1.0</td>
<td>(~700KB/s)</td>
</tr>
<tr>
<td>2001</td>
<td>Bluetooth 1.0+1.0b</td>
<td>(~2.1MB/s)</td>
</tr>
<tr>
<td>2004</td>
<td>Bluetooth 1.1</td>
<td>(~24MB/s)</td>
</tr>
<tr>
<td>2009</td>
<td>Bluetooth 2.0/2.1 +EDR</td>
<td>(~24MB/s)</td>
</tr>
<tr>
<td>2009</td>
<td>Bluetooth 3.0+HS</td>
<td>(~24MB/s)</td>
</tr>
<tr>
<td>2010</td>
<td>Bluetooth 4.0</td>
<td>(~24MB/s)</td>
</tr>
<tr>
<td>2013</td>
<td>Bluetooth 4.1</td>
<td>(~50MB/s)</td>
</tr>
<tr>
<td>2014</td>
<td>Bluetooth 4.2</td>
<td></td>
</tr>
<tr>
<td>2016</td>
<td>Bluetooth 5.0</td>
<td></td>
</tr>
</tbody>
</table>

Significant increase in range and data transfer rate
History of Bluetooth

- **1994**: Bluetooth Prototype
- **1998**: Bluetooth SIG
- **1999**: Bluetooth 1.0+1.0b
- **2001**: Bluetooth 1.1 (~700KB/s)
- **2004**: Bluetooth 1.1 (~700KB/s)
- **2009**: Bluetooth 2.0/2.1 +EDR (~2.1MB/s)
- **2010**: Bluetooth 3.0+HS (~24MB/s)
- **2013**: Bluetooth 4.0 (~24MB/s)
- **2014**: Bluetooth 4.1 (~50MB/s)
- **2016**: Bluetooth 4.2
- **2019**: Bluetooth 5.0 (~50MB/s)
- **2020**: Bluetooth 5.1/5.2 (~50MB/s)
History of Bluetooth

- **1994**: Bluetooth Prototype
- **1998**: Bluetooth SIG
- **1999**: Bluetooth 1.0+1.0b
- **2001**: Bluetooth 1.1
- **2004**: Bluetooth 2.0/2.1 +EDR
- **2009**: Bluetooth 3.0+HS
- **2010**: Bluetooth 4.0
- **2013**: Bluetooth 4.1
- **2014**: Bluetooth 4.2
- **2016**: Bluetooth 5.0
- **2019**: Bluetooth 5.1/5.2

- **1998**: (~700KB/s)
- **2001**: (~2.1MB/s)
- **2019**: (~50MB/s)

- **1998**: (~700KB/s)
- **2009**: (~24MB/s)
- **2014**: (~24MB/s)
- **2019**: (~50MB/s)
Total Annual Bluetooth Device Shipments [SIG20]
Total Annual Bluetooth Device Shipments [SIG20]
Total Annual Bluetooth Device Shipments [SIG20]
## Total Annual Bluetooth Device Shipments [SIG20]

<table>
<thead>
<tr>
<th>Category</th>
<th>Shipments 2020</th>
<th>Shipments 2025</th>
</tr>
</thead>
<tbody>
<tr>
<td>Annual Bluetooth Audio Streaming Device Shipments</td>
<td>2.11 billion</td>
<td>3.57 billion</td>
</tr>
<tr>
<td>Annual Bluetooth Phone, Tablet &amp; PC Shipments</td>
<td>1.91 billion</td>
<td>3.67 billion</td>
</tr>
<tr>
<td>Annual Bluetooth Entertainments Shipments</td>
<td>1.54 billion</td>
<td>2.93 billion</td>
</tr>
</tbody>
</table>

### Graphs

- **Annual Bluetooth Audio Streaming Device Shipments**
- **Annual Bluetooth Phone, Tablet & PC Shipments**
- **Annual Bluetooth Entertainments Shipments**
Total Annual Bluetooth Device Shipments [SIG20]
Bluetooth IoT Devices and Companion Apps

BLE IoT Devices

Companion Mobile Apps
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

1. Listen to port 443
The General Workflow of Device Communication in TCP/IP Setting

1. Listen to port 443

2. <Request, 192.168.1.1, port 443>
The General Workflow of Device Communication in TCP/IP Setting

1. Listen to port 443
2. <Request, 192.168.1.1, port 443>
3. Connect
The General Workflow of Device Communication in TCP/IP Setting

1. Listen to port 443
2. <Request, 192.168.1.1, port 443>
3. Connect
4. Authentication (Transport Layer Security / Secure Sockets Layer)
The General Workflow of Device Communication in TCP/IP Setting

1. Listen to port 443
2. <Request, 192.168.1.1, port 443>
3. Connect
4. Authentication (Transport Layer Security / Secure Sockets Layer)
5. Authentication (Application level)
The General Workflow of Device Communication in TCP/IP Setting

1. Listen to port 443
2. <Request, 192.168.1.1, port 443>
3. Connect
4. Authentication (Transport Layer Security / Secure Sockets Layer)
5. Authentication (Application level)
6. Communication
The General Workflow of BLE IoT Devices and Companion Apps
The General Workflow of BLE IoT Devices and Companion Apps

1. Request for scan

Broadcast
The General Workflow of BLE IoT Devices and Companion Apps

1. Request for scan
2. Notify

Device ➔ OS ➔ App

Broadcasting
The General Workflow of BLE IoT Devices and Companion Apps

1. Request for scan
2. Notify
3. Identify target device
The General Workflow of BLE IoT Devices and Companion Apps

1. Request for scan
2. Notify
3. Identify target device
4. Connect

Device - OS - App
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

1. Request for scan
2. Notify
3. Identify target device
4. Connect
5. Start pairing
6. Pairing feature exchange (Negotiate key entropy and elliptic curve)
The General Workflow of BLE IoT Devices and Companion Apps

1. Request for scan
2. Notify
3. Identify target device
4. Connect
5. Start pairing
6. Pairing feature exchange (Negotiate key entropy and elliptic curve)
7. Authentication and encryption

Device

OS

App

Broadcast
The General Workflow of BLE IoT Devices and Companion Apps

1. Request for scan
2. Notify
3. Identify target device
4. Connect
5. Start pairing
6. Pairing feature exchange (Negotiate key entropy and elliptic curve)
7. Authentication and encryption
8. Key distribution (e.g. IRK)
The General Workflow of BLE IoT Devices and Companion Apps

1. Request for scan
2. Notify
3. Identify target device
4. Connect
5. Start pairing
6. Pairing feature exchange
   (Negotiate key entropy and elliptic curve)
7. Authentication and encryption
8. Key distribution (e.g. IRK)
9. Authentication (App level)
The General Workflow of BLE IoT Devices and Companion Apps

1. Request for scan
2. Notify
3. Identify target device
4. Connect
5. Start pairing
6. Pairing feature exchange (Negotiate key entropy and elliptic curve)
7. Authentication and encryption
8. Key distribution (e.g. IRK)
9. Authentication (App level)
10. Communication
The General Workflow of BLE IoT Devices and Companion Apps

1. Request for scan
2. Notify
3. Identify target device
4. Connect
5. Start pairing
6. Pairing feature exchange (Negotiate key entropy and elliptic curve)
7. Authentication and encryption
8. Key distribution (e.g. IRK)
9. Authentication (App level)
10. Communication
State-of-the-Art

- BLE-Guardian [KKS16]


- Defending against sensitive information leakage during broadcasting
State-of-the-Art

BlueShield [WNK+20a]


- Detecting spoofing BLE devices during broadcasting.
State-of-the-Art

1. Request for scan
2. Notify
3. Identify target device
4. Connect
5. Start pairing
6. Pairing feature exchange
   (Negotiate key entropy and elliptic curve)
7. Authentication and encryption
8. Key distribution (e.g., IRK)
9. Authentication (App level)
10. Communication

KNOB [ATR19]


- 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]


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

BLESA [WNK+20b]
- Fake BLE device attacks against mobiles.
- Android and iOS have fixed it
State-of-the-Art


- 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.
### BadBluetooth [XDL+19]


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


- 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]


- Large-scale analysis of mis-bonding issues.
State-of-the-Art

Gattacking [Jas16]


▶ Poorly designed communication protocols are subject to various attacks (e.g., replay attacks).
State-of-the-Art

Frankenstein [RCGH20]


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

**Device OS App**
1. Request for scan
2. Notify
3. Identify target device
4. Connect
5. Start pairing
6. Pairing feature exchange
7. Authentication and encryption
8. Key distribution (e.g. IRK)
9. Authentication (App level)
10. Communication

**Authentication (Pairing)**
- Downgrade [ZWD+20]

**BLEScope [ZWLZ19]**

**FirmXRay [WLZ20]**
FirmXRay: Detecting Bluetooth Link Layer Vulnerabilities From Bare-Metal Firmware. In CCS 2020.

**Downgrade Attacks [ZWD+20]**

**References**
- BLEScope [ZWLZ19]
- FirmXRay [WLZ20]
- Future Work
- Takeaway
- References
Our **BLEScope [ZWLZ19]** Work

The Key Finding in **BLEScope** [ZWLZ19]

**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

1. Static Analysis
2. Sniff Advertised BLE Packets
3. Sniffed UUIDs
4. Static UUIDs
5. Fingerprinting
6. BLE IoT Device

---

**Introduction**

**Background**

**BLEScope [ZWLZ19]**

**FirmXRay [WLZ20]**

**Future Work**

**Takeaway**

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

Static Analysis

Vulnerabilities

Static UUIDs

Fingerprinting

Sniff Advertised BLE Packets

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

1. Sniff Advertised BLE Packets
2. Sniffed UUIDs
3. Static UUIDs
4. Vulnerabilities
5. Fingerprinting
6. Vulnerable Device
Introducing BLEScope


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

Android APKs

1

Value-set Analysis

UUID & Hierarchy
Overview of BLEScope

1. Android APKs → Value-set Analysis

2. Value-set Analysis → UUID & Hierarchy
   → Sniffed Advertisement UUIDs → UUID Fingerprinting
   → Fingerprint-able Devices
Overview of BLEScope

1. Android APKs
2. Value-set Analysis
3. App-level Vulnerability Identification

- Sniffed Advertisement UUIDs
- UUID & Hierarchy
- Sniffable Devices
- Fingerprint-able Devices
- Unauthorized Accessible Devices
- UUID Fingerprinting

References
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

<table>
<thead>
<tr>
<th>Item</th>
<th>Value</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td># Apps Support BLE</td>
<td>18,166</td>
<td>100.0</td>
</tr>
<tr>
<td># &quot;Just Works&quot; Pairing</td>
<td>11,141</td>
<td>61.3</td>
</tr>
<tr>
<td># Vulnerable Apps</td>
<td>1,757</td>
<td>15.8</td>
</tr>
<tr>
<td># Absent Cryptographic Usage</td>
<td>1,510</td>
<td>13.6</td>
</tr>
<tr>
<td># Flawed Authentication</td>
<td>1,434</td>
<td>12.9</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Item</th>
<th>Value</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td># Apps Support BLE</td>
<td>18,166</td>
<td>100.0</td>
</tr>
<tr>
<td># &quot;Just Works&quot; Pairing</td>
<td>11,141</td>
<td>61.3</td>
</tr>
<tr>
<td># Vulnerable Apps</td>
<td>1,757</td>
<td>15.8</td>
</tr>
<tr>
<td># Absent Cryptographic Usage</td>
<td>1,510</td>
<td>13.6</td>
</tr>
<tr>
<td># Flawed Authentication</td>
<td>1,434</td>
<td>12.9</td>
</tr>
</tbody>
</table>

Table: Insecure app identification result.

<table>
<thead>
<tr>
<th>Category</th>
<th># App</th>
<th>&quot;Just Works&quot;</th>
<th>Absent Crypto</th>
<th>Flawed Auth.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Health &amp; Fitness</td>
<td>3,849</td>
<td>2,639</td>
<td>221</td>
<td>207</td>
</tr>
<tr>
<td>Tools</td>
<td>2,833</td>
<td>1,895</td>
<td>385</td>
<td>362</td>
</tr>
<tr>
<td>Lifestyle</td>
<td>2,173</td>
<td>1,081</td>
<td>147</td>
<td>141</td>
</tr>
<tr>
<td>Business</td>
<td>1,660</td>
<td>972</td>
<td>90</td>
<td>85</td>
</tr>
<tr>
<td>Travel &amp; Local</td>
<td>967</td>
<td>582</td>
<td>90</td>
<td>87</td>
</tr>
</tbody>
</table>

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

Table: Experimental result of our field test.

<table>
<thead>
<tr>
<th>Item</th>
<th>Value</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td># Unique BLE Device</td>
<td>30,862</td>
<td></td>
</tr>
<tr>
<td># Unique BLE Device w. UUID</td>
<td>5,822</td>
<td>18.9</td>
</tr>
<tr>
<td># Fingerprintable</td>
<td>5,509</td>
<td>94.6</td>
</tr>
<tr>
<td># Vulnerable</td>
<td>431</td>
<td>7.4</td>
</tr>
<tr>
<td># Sniffable</td>
<td>369</td>
<td>6.7</td>
</tr>
<tr>
<td># Unauthorized Accessible</td>
<td>342</td>
<td>6.2</td>
</tr>
</tbody>
</table>
Results from Our Field Test

<table>
<thead>
<tr>
<th>Company Name</th>
<th># Devices</th>
</tr>
</thead>
<tbody>
<tr>
<td>Google</td>
<td>2,436</td>
</tr>
<tr>
<td>Tile, Inc.</td>
<td>441</td>
</tr>
<tr>
<td>Nest Labs Inc.</td>
<td>114</td>
</tr>
<tr>
<td>Logitech International SA</td>
<td>131</td>
</tr>
<tr>
<td>Hewlett-Packard Company</td>
<td>74</td>
</tr>
<tr>
<td>Google</td>
<td>92</td>
</tr>
<tr>
<td>Hewlett-Packard Company</td>
<td>46</td>
</tr>
<tr>
<td>Nest Labs Inc.</td>
<td>44</td>
</tr>
</tbody>
</table>

Table: Top 10 devices in the field test.
Results from Our Field Test

<table>
<thead>
<tr>
<th>Company Name</th>
<th># Devices</th>
</tr>
</thead>
<tbody>
<tr>
<td>Google</td>
<td>2,436</td>
</tr>
<tr>
<td>Tile, Inc.</td>
<td>441</td>
</tr>
<tr>
<td>-</td>
<td>243</td>
</tr>
<tr>
<td>-</td>
<td>208</td>
</tr>
<tr>
<td>Logitech International SA</td>
<td>131</td>
</tr>
<tr>
<td>Nest Labs Inc.</td>
<td>114</td>
</tr>
<tr>
<td>Google</td>
<td>92</td>
</tr>
<tr>
<td>Hewlett-Packard Company</td>
<td>74</td>
</tr>
<tr>
<td>-</td>
<td>46</td>
</tr>
<tr>
<td>-</td>
<td>44</td>
</tr>
<tr>
<td>-</td>
<td>44</td>
</tr>
</tbody>
</table>

Table: Top 10 devices in the field test.
Results from Our Field Test

<table>
<thead>
<tr>
<th>Device Description</th>
<th># Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digital Thermometer</td>
<td>7</td>
</tr>
<tr>
<td>Car Dongle</td>
<td>6</td>
</tr>
<tr>
<td>Key Finder A</td>
<td>6</td>
</tr>
<tr>
<td>Smart Lamp</td>
<td>5</td>
</tr>
<tr>
<td>Key Finder B</td>
<td>5</td>
</tr>
<tr>
<td>Smart Toy A</td>
<td>4</td>
</tr>
<tr>
<td>Smart VFD</td>
<td>4</td>
</tr>
<tr>
<td>Air Condition Sensor</td>
<td>4</td>
</tr>
<tr>
<td>Smart Toy B</td>
<td>4</td>
</tr>
<tr>
<td>Accessibility Device</td>
<td>4</td>
</tr>
</tbody>
</table>

Table: Top 10 **vulnerable** devices.
Results from Our Field Test

<table>
<thead>
<tr>
<th>Device Description</th>
<th># Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digital Thermometer</td>
<td>7</td>
</tr>
<tr>
<td><strong>Car Dongle</strong></td>
<td>6</td>
</tr>
<tr>
<td>Key Finder A</td>
<td>6</td>
</tr>
<tr>
<td>Smart Lamp</td>
<td>5</td>
</tr>
<tr>
<td>Key Finder B</td>
<td>5</td>
</tr>
<tr>
<td>Smart Toy A</td>
<td>4</td>
</tr>
<tr>
<td>Smart VFD</td>
<td>4</td>
</tr>
<tr>
<td>Air Condition Sensor</td>
<td>4</td>
</tr>
<tr>
<td>Smart Toy B</td>
<td>4</td>
</tr>
<tr>
<td>Accessibility Device</td>
<td>4</td>
</tr>
</tbody>
</table>

Table: Top 10 **vulnerable** devices.
Results from Our Field Test
Our FirmXRay [WLZ20] Work

BLE Link Layer Vulnerabilities

### Vulnerabilities

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

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

```
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*
```

Register Values

- r1 = 0x0
- r2 = 0x0
An Example of a Just Works Pairing Vulnerability

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] // ble_gap_sec_parms_t*
7 243b4   str    r2, [r1, #0x0]
... 25f44   ldr    r2, [0x260c8]
8 25f46   mov    r1, #0x0
9 25f48   svc    0x7f // SD_BLE_GAP_SEC_PARAMS_REPLY
... 11 260c8  0x20003268  // ble_gap_sec_parms_t*
```

Register Values

- r1 = 0x0
- r2 = 0xD
An Example of a Just Works Pairing Vulnerability

Read Only Memory

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Register</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>243a8</td>
<td>mov r2, #0x0</td>
<td>r2</td>
<td>0x0</td>
</tr>
<tr>
<td>243aa</td>
<td>orr r2, #0x1</td>
<td>r2</td>
<td>0x1</td>
</tr>
<tr>
<td>243ac</td>
<td>and r2, #0xe1</td>
<td>r2</td>
<td>0xe1</td>
</tr>
<tr>
<td>243ae</td>
<td>add r2, #0xc</td>
<td>r2</td>
<td>0xc</td>
</tr>
<tr>
<td>243b0</td>
<td>and r2, #0xdf</td>
<td>r2</td>
<td>0xdf</td>
</tr>
<tr>
<td>243b2</td>
<td>ldr r1, [0x260c8]</td>
<td>r1</td>
<td>0x260c8</td>
</tr>
<tr>
<td>243b4</td>
<td>str r2, [r1, #0x0]</td>
<td>r2</td>
<td>0x0</td>
</tr>
</tbody>
</table>

Random Access Memory

```c
struct ble_gap_sec_params_t {
    uint8 pairing_feature;  // 20003268
    uint8 min_key_size;     // 20003269
    uint8 max_key_size;     // 20003270
    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

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]

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

// ble_gap_sec_params_t*
20003268

// ble_gap_sec_params_t*
20003268
An Example of a Just Works Pairing Vulnerability

**Read Only Memory**

```asm
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*
```

**Random Access Memory**

```c
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

```
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]
...```

Random Access Memory

```
Struct ble_gap_sec_params_t

20003268  uint8  pairing_feature = 0xD

<table>
<thead>
<tr>
<th>BOND</th>
<th>MITM</th>
<th>IO</th>
<th>OOB</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>
// 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

// SD_BLE_GAP_SEC_PARAMS_REPLY
...
An Example of a Just Works Pairing Vulnerability

Correct Firmware Disassembling

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]
...```

Random Access Memory

```
Struct ble_gap_sec_params_t

20003268  uint8 pairing_feature = 0xD

<table>
<thead>
<tr>
<th>BOND</th>
<th>MITM</th>
<th>IO</th>
<th>OOB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
|      |      |    |     | // 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

Correct Firmware Disassembling
An Example of a Just Works Pairing Vulnerability

Correct Firmware Disassembling

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_secParms_t*

Recognize data structures

Random Access Memory

Struct ble_gap_sec_params_t

20003268  uint8 pairing_feature = 0xD

<table>
<thead>
<tr>
<th>BOND</th>
<th>MITM</th>
<th>IO</th>
<th>OOB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
// 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

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>243a8 mov r2, #0x0</td>
</tr>
<tr>
<td>2</td>
<td>243aa orr r2, #0x1</td>
</tr>
<tr>
<td>3</td>
<td>243ac and r2, #0xe1</td>
</tr>
<tr>
<td>4</td>
<td>243ae add r2, #0xc</td>
</tr>
<tr>
<td>5</td>
<td>243b0 and r2, #0xdf</td>
</tr>
<tr>
<td>6</td>
<td>243b2 ldr r1, [0x260c8]</td>
</tr>
<tr>
<td>7</td>
<td>243b4 str r2, [r1, #0x0]</td>
</tr>
</tbody>
</table>

... Values

r1 = 0x0
r2 = 0x20003268

Correct Firmware Disassembling

Recognize data structures

Random Access Memory

Struct ble_gap_sec_params_t

\[
\begin{array}{|c|c|c|c|}
\hline
\text{BOND} & \text{MITM} & \text{IO} & \text{OOB} \\
\hline
\text{pairing_feature} & 0xD & & 0x0 \\
\hline
\end{array}
\]

// BOND = 1, MITM = 0
// IO   = 3, OOB  = 0

uint8 min_key_size
uint8 max_key_size
ble_gap_sec_kdist_t kdist_own
ble_gap_sec_kdist_t kdist_peer

Value computation

Register Values

r1 = 0x0
r2 = 0x20003268

// SD_BLE_GAP_SEC_PARAMS_REPLY

...
FirmXRay Overview

- Robust Firmware Disassembling
- Precise Data Structure Recognition
- Configuration Value Resolution
- Detection Policies
- Vulnerabilities

\[ X = \text{argmax}_{x \in \mathbb{R}} N(x) \]
Robust Firmware Disassembling

(1) Absolute Function Pointer

(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}

05454  ldr  r0, pc+0x72
05454  blx  r0=>0x22A90

... 054c4  0x22A90

Function Foo()

07a90  push  {r3, r4, r5, lr}

Correct Base 0x1B000

04e52  ldr  r0, [pc+0x146]
04e54  ldmia  r0=>0x23058, {r4, r5, r6}

... 04f98  0x23058

"KinsaHealth"

04e54  ldmia  r0=>0x23058, {r4, r5, r6}

... 08058  "KinsaHealth"

(1) Absolute Function Pointer

(2) Absolute String Pointer

(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

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}

Absolute Pointers: 0x22A90, 0x23058

Gadgets: 0x07A90, 0x08058

N(0x1B000) = 2
Robust Firmware Disassembling

```
0x05452  ldr  r0, pc+0x72
0x05454  blx  r0=>0x22A90
...
0x054c4  0x22A90
...
Function Foo()
0x07a90  push  {r3, r4, r5, lr}
```

Absolute Pointers: 0x22A90, 0x23058

Gadgets: 0x07A90, 0x08058

```
0x04e52  ldr  r0, pc+0x146
0x04e54  ldmia r0=>0x23058, {r4, r5, r6}
...
0x04f98  0x23058

0x08058  "KinsaHealth"
```

Base 0x0

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

```plaintext
Absolute Pointers: 0x22A90, 0x23058

Gadgets: 0x07A90, 0x08058

N(0x1B000) = 2
```

0x1B000 = 0x22A90 - 0x07A90 = 0x23058 - 0x08058
Precise Data Structure Recognition

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(r0, r1, r2)
... 
11 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]
...```

Program Path

```
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_params_t*
```
Configuration Value Resolution

Robust Firmware
Disassembling

Precise Data
Structure Recognition

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

Robust Firmware Disassembling

Precise Data Structure Recognition

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

Robust Firmware Disassembling

Precise Data Structure Recognition

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
...```

Program Path

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

r2 = 0x20003268```

r2 = 0x20003268
Configuration Value Resolution

Robust Firmware Disassembling

Precise Data Structure Recognition

Configuration Value Resolution

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

2M Free Apps
Firmware Collection

2M Free Apps → Filter → 13K BLE Apps
Firmware Collection

2M Free Apps  
13K BLE Apps  
793 Firmware
Firmware Collection

2M Free Apps

Filter

13K BLE Apps

Unpack

Extract

793 Firmware

768 Nordic

25 TI
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

<table>
<thead>
<tr>
<th>Category</th>
<th># Firmware</th>
<th># Device</th>
<th>Avg. Size (KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Nordic-based Firmware</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wearable</td>
<td>204</td>
<td>138</td>
<td>98.2</td>
</tr>
<tr>
<td>Others</td>
<td>76</td>
<td>22</td>
<td>223.5</td>
</tr>
<tr>
<td>Sensor</td>
<td>67</td>
<td>51</td>
<td>80.9</td>
</tr>
<tr>
<td>Tag (Tracker)</td>
<td>58</td>
<td>41</td>
<td>84.2</td>
</tr>
<tr>
<td>Robot</td>
<td>41</td>
<td>21</td>
<td>117.7</td>
</tr>
<tr>
<td>Medical Devices</td>
<td>41</td>
<td>21</td>
<td>138.6</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>793</td>
<td>538</td>
<td>102.7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Category</th>
<th># Firmware</th>
<th># Device</th>
<th>Avg. Size (KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TI-based Firmware</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sensor</td>
<td>19</td>
<td>19</td>
<td>132.9</td>
</tr>
<tr>
<td>Smart Lock</td>
<td>2</td>
<td>2</td>
<td>46.3</td>
</tr>
<tr>
<td>Smart Toy</td>
<td>2</td>
<td>2</td>
<td>47.8</td>
</tr>
<tr>
<td>Medical Devices</td>
<td>1</td>
<td>1</td>
<td>70.2</td>
</tr>
<tr>
<td>Others</td>
<td>1</td>
<td>1</td>
<td>76.7</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>793</td>
<td>538</td>
<td>102.7</td>
</tr>
</tbody>
</table>

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

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

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.
Identity Tracking Vulnerability Identification

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

<table>
<thead>
<tr>
<th>Firmware Name</th>
<th>Mobile App</th>
<th>Category</th>
<th># Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>cogobeacon</td>
<td>com.aegismobility.guardian</td>
<td>Car Accessory</td>
<td>4</td>
</tr>
<tr>
<td>sd_bl</td>
<td>fr.solem.solemwf</td>
<td>Agricultural Equip.</td>
<td>2</td>
</tr>
<tr>
<td>LRFL_nRF52</td>
<td>fr.solem.solemwf</td>
<td>Agricultural Equip.</td>
<td>2</td>
</tr>
<tr>
<td>orb</td>
<td>one.shade.app</td>
<td>Smart Light</td>
<td>1</td>
</tr>
<tr>
<td>sd_bl</td>
<td>com.rainbird</td>
<td>Agricultural Equip.</td>
<td>1</td>
</tr>
</tbody>
</table>

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.

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

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.

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

Table: Firmware that enforce LESC pairing.
Attack Case Studies

nRF52840 DK

Vulnerable BLE Devices
## Attack Case Studies

<table>
<thead>
<tr>
<th>Device Name</th>
<th>Category</th>
<th>Attacks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nuband Activ+</td>
<td>Wearable</td>
<td>✓</td>
</tr>
<tr>
<td>Kinsa Smart</td>
<td>Thermometer</td>
<td></td>
</tr>
<tr>
<td>Chipolo ONE</td>
<td>Tag</td>
<td>✓</td>
</tr>
<tr>
<td>SwitchBot Button Pusher</td>
<td>Smart Home</td>
<td></td>
</tr>
<tr>
<td>XOSS Cycling Computer</td>
<td>Sensor</td>
<td>✓</td>
</tr>
</tbody>
</table>

**A1: User Tracking**

![Image of devices](image-url)
## Attack Case Studies

<table>
<thead>
<tr>
<th>Device Name</th>
<th>Category</th>
<th>Attacks</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>A1</td>
</tr>
<tr>
<td>Nuband Activ+</td>
<td>Wearable</td>
<td>✓</td>
</tr>
<tr>
<td>Kinsa Smart</td>
<td>Thermometer</td>
<td></td>
</tr>
<tr>
<td>Chipolo ONE</td>
<td>Tag</td>
<td>✓</td>
</tr>
<tr>
<td><strong>SwitchBot Button Pusher</strong></td>
<td><strong>Smart Home</strong></td>
<td>✓</td>
</tr>
<tr>
<td>XOSS Cycling Computer</td>
<td>Sensor</td>
<td>✓</td>
</tr>
</tbody>
</table>

**A2: Unauthorized Control**
## Attack Case Studies

<table>
<thead>
<tr>
<th>Device Name</th>
<th>Category</th>
<th>Attacks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nuband Activ+</td>
<td>Wearable</td>
<td>✓</td>
</tr>
<tr>
<td>Kinsa Smart</td>
<td>Thermometer</td>
<td>✓</td>
</tr>
<tr>
<td>Chipolo ONE</td>
<td>Tag</td>
<td>✓</td>
</tr>
<tr>
<td>SwitchBot Button Pusher</td>
<td>Smart Home</td>
<td>✓</td>
</tr>
<tr>
<td>XOSS Cycling Computer</td>
<td>Sensor</td>
<td>✓</td>
</tr>
</tbody>
</table>

A3: Sensitive Data Eavesdropping
Near Term

**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

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

#### Scanning Defense: Defending against malicious scanning.

#### Notification Fingerprinting: Exploring notification fingerprinting against BLE devices.

**1. Request for scan**

**2. Notify**

**3. Identify target device**

**4. Connect**

**5. Start pairing**

**6. Pairing feature exchange**

- Negotiate key entropy and elliptic curve

**7. Authentication and encryption**

**8. Key distribution (e.g., IRK)**

**9. Authentication (App level)**

**10. Communication**

---

**References**

- BLEScope [ZWLZ19]
- FirmXRay [WLZ20]
- BLESA [WNK+20b]
- BlueShield [WNK+20]
- BLE-Guardian [KKS16]
- Downgrade Attack [ZWD+20]
- KNOB [ATR19]
- Crackle [Rya13]
- BIAS [ATR20]
- Mis-Bonding [NJD+14]
- BadBluetooth [XDL+19]
- Gattack [Jas16]
- Co-located Attacks [SB19]
- Frankenstein [RCGH20]
- Gattack [Jas16]
- Frankenstein [RCGH20]
Near Term

1. OS Defense: OS-level defense to patch multiple security issues.
### 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


The Landscape of Bluetooth Security and Privacy

Device

FirmXRay [WLZ20]

BLEGuardian [KKS16]

BlueShield [WNK+20]

OS

App

1. Request for scan

2. Notify

3. Identify target device

4. Connect

5. Start pairing

6. Pairing feature exchange

7. Authentication and encryption

8. Key distribution (e.g. IRK)

9. Authentication (App level)

10. Communication

Broadcast

BLEScope [ZWLZ19]

Mis-Bonding [NZD+14]; Co-Located Attacks [SB19]

Crackle [Rya13]; KNOB [ATR19]; BIAS [ATR20]; Gattack [Jas16]; Frankenstein [RCGH20]

Authentication (Pairing)

Downgrade Attack [ZWD+20]
**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
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 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


Bluetooth SIG, Market update 2020, [EB/OL], 2020, 


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.

