Bluetooth:

With Low Energy Comes Low Security

Mike Ryan
iSEC Partners

USENIX Security / WOOT
Aug 13, 2013
Outline

→ What is Bluetooth Low Energy?
→ Protocol overview
→ Sniffing Techniques
→ [In]security
→ Injection
Outline

→ What is Bluetooth Low Energy?
→ Protocol overview
→ Sniffing Techniques
→ [In]security
→ Injection
What is Bluetooth Low Energy?
What is Bluetooth Low-Energy Smart?

- New modulation and link layer for low-power devices
- vs classic Bluetooth
  - Incompatible with classic Bluetooth devices
  - PHY and link layer almost completely different
  - High-level protocols reused (L2CAP, ATT)
- Introduced in Bluetooth 4.0 (2010)
- AKA BTLE
Where is BTLE?

- High end smartphones
- Sports / fitness devices
- Door locks
- Upcoming medical devices
By The Numbers

- 186% YoY Growth for H1 2013\(^1\)
- “over 7 million Bluetooth Smart ICs were estimated to have shipped for use in sports and fitness devices in the first half of 2013 alone”
- “Analysts Forecast Bluetooth Smart to Lead Market Share in Wireless Medical and Fitness Devices”\(^2\)

\(^1\)http://www.bluetooth.com/Pages/Press-Releases-Detail.aspx?ItemID=170
\(^2\)http://www.bluetooth.com/Pages/Press-Releases-Detail.aspx?ItemID=165
Outline

→ What is Bluetooth Low Energy?
→ Protocol overview
→ Sniffing Techniques
→ [In]security
→ Injection
## Protocol Stack

<table>
<thead>
<tr>
<th>Layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>GATT</td>
</tr>
<tr>
<td>ATT</td>
</tr>
<tr>
<td>L2CAP</td>
</tr>
<tr>
<td>Link Layer</td>
</tr>
<tr>
<td>PHY</td>
</tr>
</tbody>
</table>
PHY Layer

- GFSK, +/- 250 kHz, 1 Mbit/sec
- 40 channels in 2.4 GHz
- Hopping
## Physical Channels

- **Advertising:** 3 channels
- **Data:** 37 channels

<table>
<thead>
<tr>
<th>RF Channel</th>
<th>RF Center Frequency</th>
<th>Channel Type</th>
<th>Data Channel Index</th>
<th>Advertising Channel Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2402 MHz</td>
<td>Advertising</td>
<td></td>
<td>37</td>
</tr>
<tr>
<td>1</td>
<td>2404 MHz</td>
<td>Data</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2406 MHz</td>
<td>Data</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>Data channels</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>2424 MHz</td>
<td>Data</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>2426 MHz</td>
<td>Advertising</td>
<td></td>
<td>38</td>
</tr>
<tr>
<td>13</td>
<td>2428 MHz</td>
<td>Data</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>2430 MHz</td>
<td>Data</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>Data channels</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>2478 MHz</td>
<td>Data</td>
<td>36</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>2480 MHz</td>
<td>Advertising</td>
<td></td>
<td>39</td>
</tr>
</tbody>
</table>
Hopping

- Hop along 37 data channels
- One data packet per channel
- Next channel $\equiv$ channel + hop increment (mod 37)
- Time between hops: hop interval

3 → 10 → 17 → 24 → 31 → 1 → 8 → 15 → ...

hop increment = 7
Link Layer

- PDU min of 2 bytes due to 2 byte header
- LLID: Control vs Data
- Length

Figure 2.1: Link Layer packet format
L2CAP and Beyond

→ Use existing decoders for this
→ Not a Hard Problem™
Recap

- GATT
- ATT
- L2CAP
- Link Layer
- PHY
Outline

→ What is Bluetooth Low Energy?
→ Protocol overview
→ Sniffing Techniques
→ [In]security
→ Injection
sniffing Bluetooth is hard
sniffing
Bluetooth LE is slightly less hard
How do we sniff it?

Start at the bottom and work our way up:

- Ubertooth
- Link Layer
- L2CAP
- ATT
- GATT
- PC
Ubertooth Block Diagram

CC2591 RF Amp → RF
CC2400 Radio

PHY layer
RF ↔ Bits

Link layer
Bits ↔ Packets

LPC175x ARM MCU

Packets → USB
Capturing: PHY Layer

→ Configure CC2400
  → Set modulation parameters to match Bluetooth Smart
  → Tune to proper channel
→ Follow connections according to hop pattern
  → Hop increment and hop interval, sniffed from connect packet or recovered in promiscuous mode
→ Hand off bits to ARM MCU
What Info Do We Need?

<table>
<thead>
<tr>
<th>Preamble</th>
<th>Access Address</th>
<th>PDU</th>
<th>CRC</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1 octet)</td>
<td>(4 octets)</td>
<td>(2 to 39 octets)</td>
<td>(3 octets)</td>
</tr>
</tbody>
</table>

> Access Address
- Advertising: Fixed **0x8E89BED6**
- Connection: Varies

> Channel number
- Hop interval
- Hop increment

> Nice to have: CRCInit

Easy mode: Connect packet!
Link Layer

<table>
<thead>
<tr>
<th>LSB</th>
<th>MSB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preamble (1 octet)</td>
<td></td>
</tr>
<tr>
<td>Access Address (4 octets)</td>
<td></td>
</tr>
<tr>
<td>PDU (2 to 39 octets)</td>
<td>CRC (3 octets)</td>
</tr>
</tbody>
</table>

Figure 2.1: Link Layer packet format

What we know: Access Address
What we have: Sea of bits
What we want: Start of PDU

CC2400 does this for free

10001110 11110101010101
100111000000100011001
11100100110100011101
PHY Layer.. Link Layer..

We converted RF to packets

Now what?
Capturing Packets... To PCAP!

→ ubertooth-btle speaks packets
→ libpcap → dump raw packet data
→ PPI header (similar airodump-ng and kismet)

→ We have a DLT for Bluetooth Smart
  → Unique identifier for the protocol
  → Public release of Wireshark plugin Coming Soon™
### Wireshark Awesomeness

#### Bluetooth Attribute Protocol
- **Opcode:** Read By Type Request (0x08)
- **Starting Handle:** 0x0000
- **Ending Handle:** 0xffff
- **UUID:** Device Name (0x2a00)
- **CRC:** 0x11fa7f

#### Bluetooth Low Energy
- **PPI version:** 0, 19 bytes
- **DLT:** 147, Payload: btle (Bluetooth Low Energy)
- **Access Address:** 0x50e55292
- **Data FDU Header:** 0x0000
- **Bluetooth L2CAP Protocol**
  - **Bluetooth Attribute Protocol**
    - **Opcode:** Read By Type Response (0x09)
    - **Length:** 19
    - **Attribute Data, Handle:** 0x0003
      - **Value:** 544920424c452053656e736f7220546168656e74
        - **CRC:** 0x6781c4

#### Wireshark Packet Analysis
- **Frame 520:** 39 bytes on wire (312 bits), 39 bytes captured (312 bits)
- **PPI version:** 0, 19 bytes
- **DLT:** 147, Payload: btle (Bluetooth Low Energy)
- **Bluetooth Attribute Protocol**
  - **Opcode:** Read By Type Request (0x08)
  - **Starting Handle:** 0x0000
  - **Ending Handle:** 0xffff
  - **UUID:** Device Name (0x2a00)
  - **CRC:** 0x11fa7f
Promiscuous Mode

→ Techniques for recovering
  → Access Address
  → CRCInit
  → Hop Interval
  → Hop Increment
Recovering Access Address

→ Sit on data channel waiting for empty data packets
→ Collect candidate AA's and pick one when it's been observed enough

Not depicted: whitening!
Recovering CRCInit

→ Filter packets by Access Address
→ Plug CRC into LFSR and run it backward

Figure 3.2: The LFSR circuit generating the CRC

See also “Bluesniff: Eve meets Alice and Bluetooth”, USENIX WOOT ’07
Recovering Hop Interval

→ Observation: 37 is prime
→ Sit on data channel and wait for two consecutive packets

\[ \frac{\Delta t}{37} = \text{hop interval} \]
Recovering Hop Increment

- Start on data channel 0, jump to data channel 1 when a packet arrives
- We know hop interval, so we can calculate how many channels were hopped between 0 and 1
Recovering Hop Increment (math)

0 + hopIncrement \times channelsHopped \equiv 1 \pmod{37}

hopIncrement \equiv channelsHopped^{-1} \pmod{37}

channelsHopped^{-1} \equiv channelsHopped^{37-2} \pmod{37}

We use a LUT to convert that to hop increment
Sniffing Summary

- Connection following
- Promiscuous: Recovering the four values
  - Access address
  - CRCInit
  - Hop interval
  - Hop Increment
Outline

- What is Bluetooth Low Energy?
- Protocol overview
- Sniffing Techniques
- [In]security
- Injection
Encryption

- Provided by link layer
- Encrypts and MACs PDU
- AES-CCM

\[\text{Figure 2.1: Link Layer packet format}\]
Custom Key Exchange Protocol

→ Three stage process

→ 3 pairing methods
  → Just Works™
  → 6-digit PIN
  → OOB

→ “None of the pairing methods provide protection against a passive eavesdropper” - Bluetooth Core Spec
Cracking the TK

\[
\text{confirm} = \text{AES(TK, AES(TK, rand XOR p1) XOR p2)}
\]

GREEN = we have it
RED = we want it

TK: integer between 0 and 999,999
Just Works™: always 0!
Cracking the TK – With crackle

Total time to crack: < 1 second
And That's It

→ TK → STK
→ STK → LTK
→ LTK → Session keys

KEY EXCHANGE = BROKEN
100% PASSIVE
LTK Reuse

¬ Good for security: pair in a faraday cage
¬ Counter-mitigation: Active attack to force re-pairing
Decrypting

→ Assumption: Attacker has LTK – reused!

→ Procedure
  → Attacker passively capturing packets
  → Connection established
  → Session information captured
Decrypting – With \textit{crackle}

\begin{itemize}
  \item Yes, \textit{crackle} does that too!
  \item \textit{crackle} will decrypt
    \begin{itemize}
      \item a PCAP file with a pairing setup
      \item a PCAP file with an encrypted session, given an LTK
    \end{itemize}
\end{itemize}
Am I Affected?

→ Probably

→ Exception: Some vendors implement their own security on top of GATT
  
  → Did they talk to a cryptographer?
Security Recap

→ Key exchange broken

→ LTK reuse means all communication is effectively compromised

→ 99% passive

→ Worst case scenario: one active attack with off-the-shelf hardware
Outline

→ What is Bluetooth Low Energy?
→ Protocol overview
→ Sniffing Techniques
→ [In]security
→ Injection
Injection

- Pretty much the same as receiving, opposite direction
- Follow the spec!
  - Link layer header
  - Payload data
- Hand that off to Ubertooth
  - Calculate CRC
  - Whiten
- Devil is in the CC2400 details
Demo
Capabilities

→ Ubertooth
  → Passively intercept Bluetooth Smart
  → Promiscuous mode
  → Injection
→ Wireshark plugins
→ crackle
  → Crack TK's sniffed with Ubertooth
  → Decrypt PCAP files with LTK
Software

→ Ubertooth and libbbtbb
  → http://ubertooth.sourceforge.net/

→ crackle
  → http://lacklustre.net/projects/crackle/

→ nano-ecc (8-bit ECDH and ECDSA)
  → https://github.com/iSECPartners/nano-ecc
Thanks

Mike Ossmann
Dominic Spill

Mike Kershaw (dragorn)
#ubertooth on freenode
bluez
Bluetooth SIG
USENIX
iSEC Partners
Thank You

Mike Ryan
iSEC Partners
@mpeg4codec
mikeryan@isecpartners.com
http://lacklustre.net/
Apocrypha (extra)

```
0x1FFEFFFFFC00: remove channels 12, 27-36
```
Encryption Mitigation (extra)

- Every session uses a different session key
- Every session uses several nonces

  **Solution**: jam the connection to restart a session

- LTK exchanged once, used many times

  **Solution**: inject LTK_REJECT_IND message
L2CAP (extra)
ATT/GATT (extra)

- Services: groups of characteristics
- Characteristics
  - Operations
- Everything identified by UUID
  - 128 bit
  - Sometimes shortened to 16 bits
Example GATT Service: Heart Rate (extra)

- Service: \(0x180D\)
- Characteristic 1: \(0x2A37\) – Heart Rate
  - Can't read or write
  - Notify: subscribe to updates
- Characteristic 2: \(0x2A38\) – Sensor Location
  - Readable: 8 bit int, standardized list
- Other characteristics: \(0x2803, 0x2902, \ldots\)