

Bluetooth Smart:

The Good, The Bad, The Ugly... and The Fix

Mike Ryan
iSEC Partners

Black Hat USA
Aug 01, 2013

Why Bluetooth Smart?

→ Because it's appearing EVERYWHERE

Why Bluetooth Smart? (2)

- 186% YoY Growth for H1 2013¹
- “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>

The Good

Bluetooth Smart

What is Bluetooth 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 the same (L2CAP, ATT)
- Introduced in Bluetooth 4.0 (2010)
- AKA Bluetooth Low Energy / BTLE

Protocol Stack



PHY Layer

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

Hopping

- Hop along 37 data channels
- One data packet per channel
- Next channel = (channel + hop increment) mod 37

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

hop increment = 7

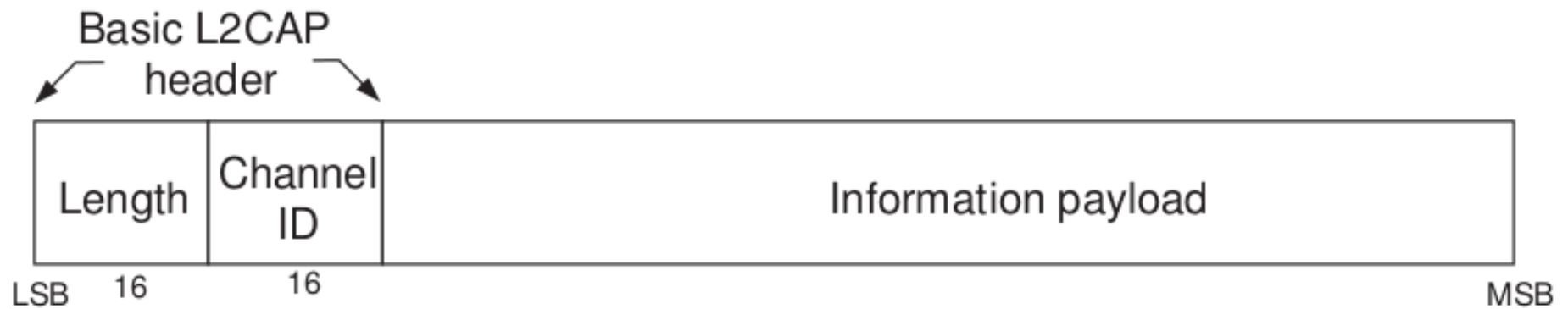
Link Layer



Figure 2.1: Link Layer packet format

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

L2CAP: A Few Bytes Octets of Bloat



ATT/GATT

- Services: groups of characteristics
- Characteristics
 - Operations
- Everything identified by UUID
 - 128 bit
 - Sometimes shortened to 16 bits

Example GATT Service: Heart Rate

- 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, ...**

Recap



sniffing
Bluetooth
is
hard

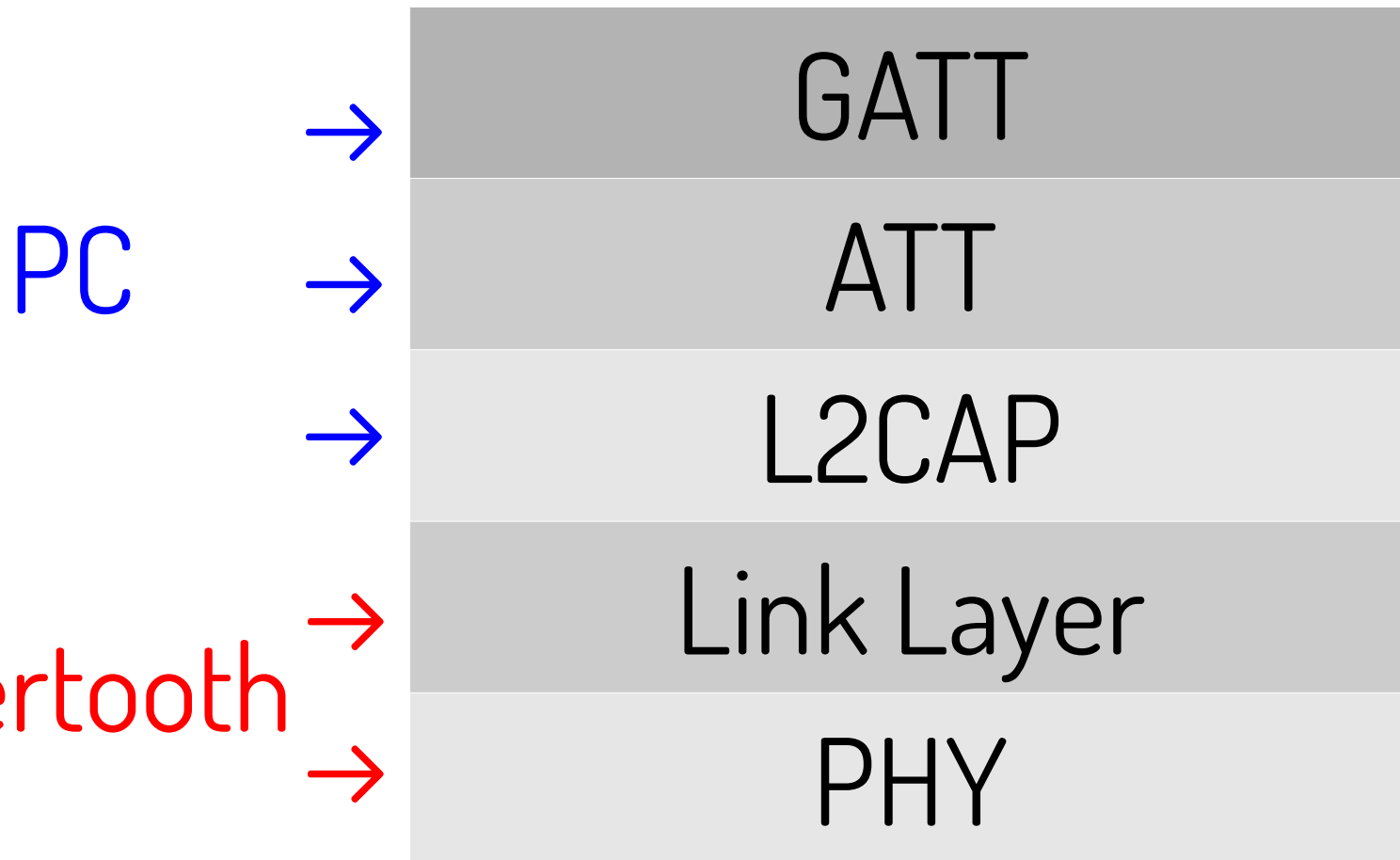
sniffing

Bluetooth LE

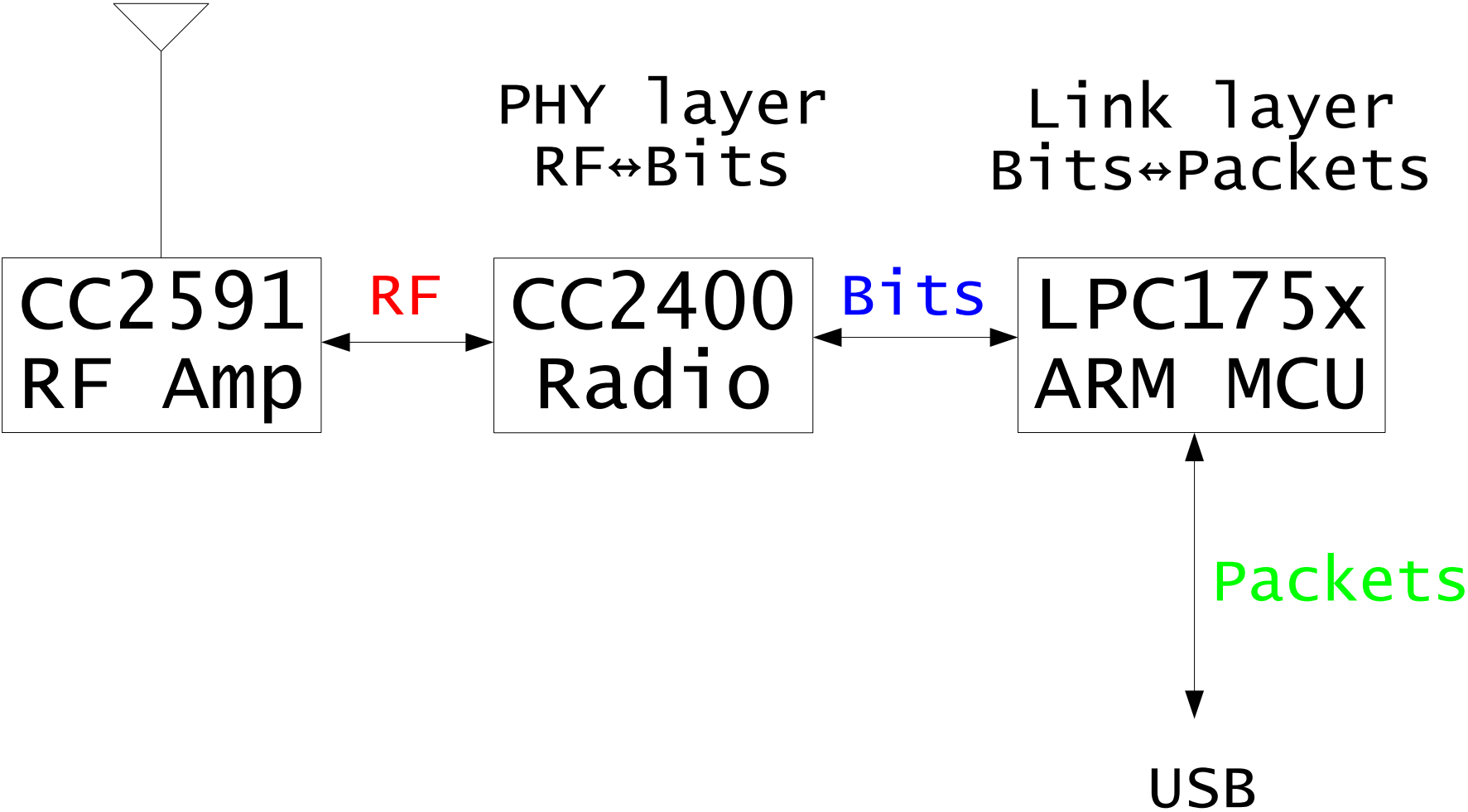
is slightly
less hard

How do we sniff it?

Start at the bottom and work our way up:



Ubertooth Block Diagram



Capturing Packets

- 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

Link Layer



Figure 2.1: Link Layer packet format

What we have: Sea of bits

What we want: Start of PDU

What we know: AA

```
10001110111101010101
10011100000100011001
11100100110100011101
```

CC2400 does this
FO FREE

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

The image displays two side-by-side screenshots of the Wireshark network protocol analyzer interface, showing packet captures for Bluetooth Low Energy (BLE) traffic. Both screenshots are filtered by the expression 'btatt'.

Left Screenshot (Packet 520):

- Packet List:** Shows packet 520 at time 44.565491, length 39 bytes, protocol ATT, with info 'Read By Type Request, Device Name'.
- Packet Details:**
 - 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 Low Energy
 - Access Address: 0x50655292
 - Data PDU Header: 0x0b02
 - Bluetooth L2CAP Protocol
 - Bluetooth Attribute Protocol
 - Opcode: Read By Type Request (0x08)
 - Starting Handle: 0x0001
 - Ending Handle: 0xffff
 - UUID: Device Name (0x2a00)
 - CRC: 0x11fa7f
- Packet Bytes:** Shows hex and ASCII data for the packet, with the payload starting at offset 0010: 7c 20 20 92 52 65 50 02 0b 07 00 04 00 08 01 00.
- Packet Info:** UUID (btatt.uuid16), 2 bytes

Right Screenshot (Packet 523):

- Packet List:** Shows packet 523 at time 44.634088, length 53 bytes, protocol ATT, with info 'Read By Type Response, Attribute Value'.
- Packet Details:**
 - Frame 523: 53 bytes on wire (424 bits), 53 bytes captured (424 bits)
 - PPI version 0, 19 bytes
 - DLT: 147, Payload: btle (Bluetooth Low Energy)
 - Bluetooth Low Energy
 - Access Address: 0x50655292
 - Data PDU Header: 0x190a
 - Bluetooth L2CAP Protocol
 - Bluetooth Attribute Protocol
 - Opcode: Read By Type Response (0x09)
 - Length: 19
 - Attribute Data, Handle: 0x0003
 - Handle: 0x0003
 - Value: 544920424c4520536556e736f7220546167
 - CRC: 0x6781c4
- Packet Bytes:** Shows hex and ASCII data for the packet, with the payload starting at offset 0010: d2 2a 20 92 52 65 50 0a 19 15 00 04 00 09 13 03.
- Packet Info:** Value (btatt.value), 17 bytes

Encryption

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

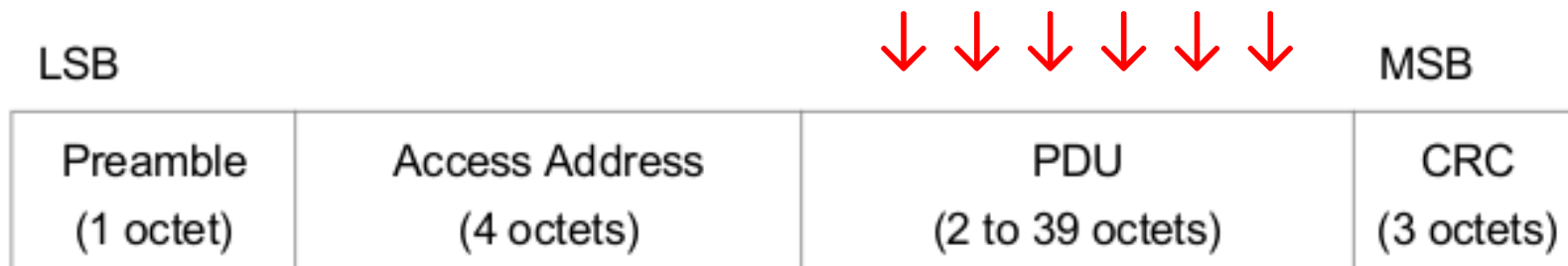


Figure 2.1: Link Layer packet format

The Bad

Key Exchange

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

confirm
=
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

The Ugly

LTK Reuse

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 *crackle*

- Yes, crackle does that too!
- crackle will decrypt
 - a PCAP file with a pairing setup
 - a PCAP file with an encrypted session, given an LTK

The Ugly: 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

The Fix

Secure Simple Pairing

My Qualifications

- Infosec Researcher
- Infosec Consultant
- Occasional programmer
- Husband
- Able to grill a mean steak

Shameless Plug:
iSEC Partners

NOT LISTED: Cryptographer

Why Secure Simple Pairing?

- Eavesdropping protection: ECDH
- In production since 2007, only one weakness
- Downside: ECDH is expensive
 - secp192r1: ~5 seconds on 8-bit CPU
 - No open source implementation (until now)

The Five Phases of SSP

1. Public key exchange
2. Authentication Stage 1
3. Authentication Stage 2
4. Link Key Calculation
5. LMP Authentication and Encryption

SSP in Bluetooth Smart

1. Public key exchange
2. Authentication Stage 1: Numeric comparison only
3. Authentication Stage 2
4. Link Key Calculation
5. ~~LMP Authentication and Encryption~~

Backward Compatibility

- OOB not broken
- Use calculated link key as 128-bit OOB data
- Most chips have support

Demo

→ D

→ e

• m

- o

Am I Affected?

- Probably
- Exception: Some vendors implement their own security on top of GATT
 - Did they talk to a cryptographer?

Summary

- The Good: Bluetooth Smart
- The Bad: Key Exchange
- The Ugly: LTK Reuse
- The Fix: SSP

Capabilities

- Ubertooth
 - Passively intercept Bluetooth Smart
 - Promiscuous mode and injection (not discussed)
- Wireshark plugins
- crackle
 - Crack TK's sniffed with Ubertooth
 - Decrypt PCAP files with LTK
- nano-ecc: 8-bit ECDH implementation

Software

- Ubertooth and libbtbb
 - <http://ubertooth.sourceforge.net/>
- nano-ecc (8-bit ECDH and ECDSA)
 - <https://github.com/iSECPartners/nano-ecc>
- crackle
 - <http://lacklustre.net/projects/crackle/>

Thanks

Mike Ossmann
Dominic Spill

Mike Kershaw (dragorn)
#ubertooth on freenode

bluez

Bluetooth SIG

Black Hat

iSEC Partners

Thank You

Mike Ryan

iSEC Partners

@mpeg4codec

mikeryan@isecpartners.com

<http://lacklustre.net/>

Feedback

Please scan badge
when leaving

Thanks again!