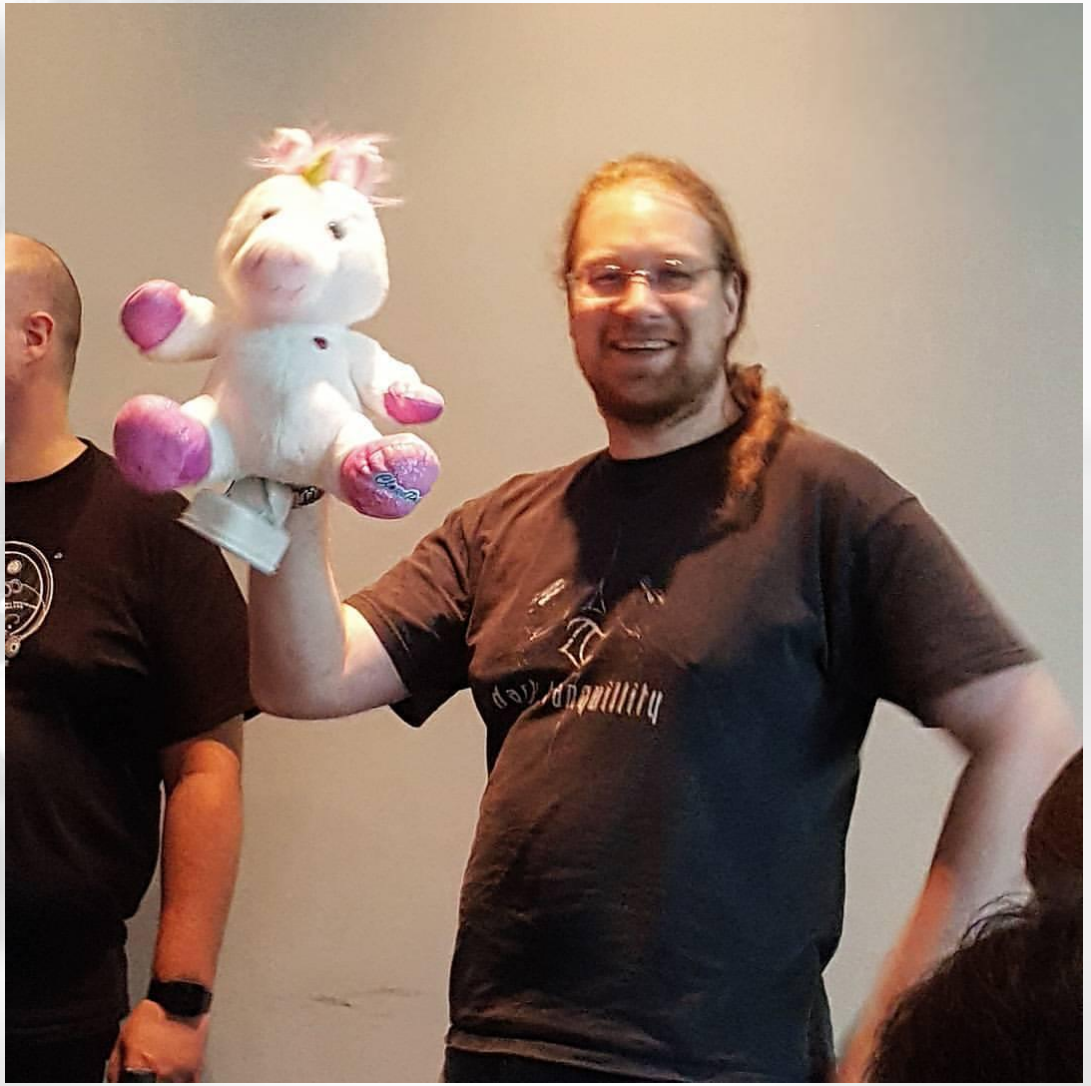
From Wearables to Insertables: An Intro to BLE Hacking

info@pentestpartners.com
+44 (0)20 3095 0500
@PenTestPartners
PenTestPartnersLLP

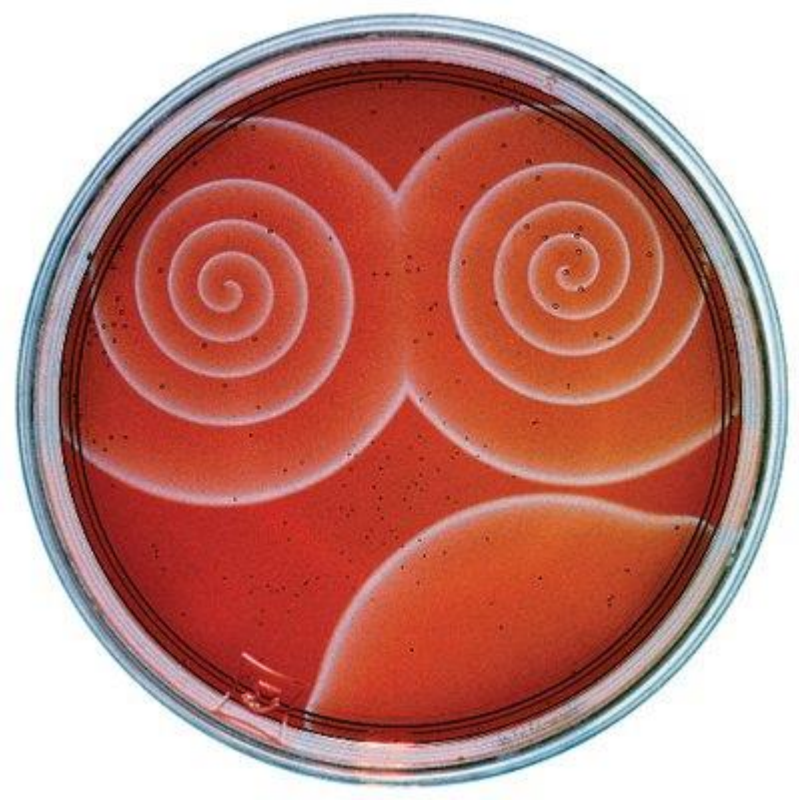
Who are we?

David Lodge, @tautologyo



Who are we?

Mark Carney, @LargeCardinal



Tools

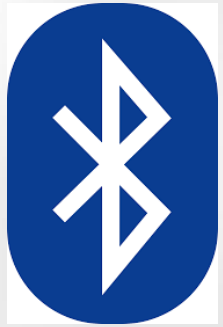
- Get these downloaded

- Bluez stack
  - hcitool
  - gatttool

- bluepy
- Bleah
- bleno/noble
- btleproxy
- gattacker
- HCI Snoop
- nRF Connect / BLE Scanner

Why Should I care?

- Lots of basic RF protocols, in no particular order:
  - IEEE 802.15.4 (Zigbee, Xbee)
  - IEEE 1902.1 (RuBee)
  - ANT/ANT+
  - Bluetooth/BLE
  - Z-Wave
  - RFID
  - ISA100.11a
  - ULE (DECT)



That's too much to talk about

- BLE, aka:
  - Bluetooth 4.0 (to 4.2)
  - BT Smart
  - Bluetooth Low Energy
- As opposed to Bluetooth Classic
  - BR/EDR (Basic Rate/Enhanced Data Rate)
- Defined in a 2772 page specification

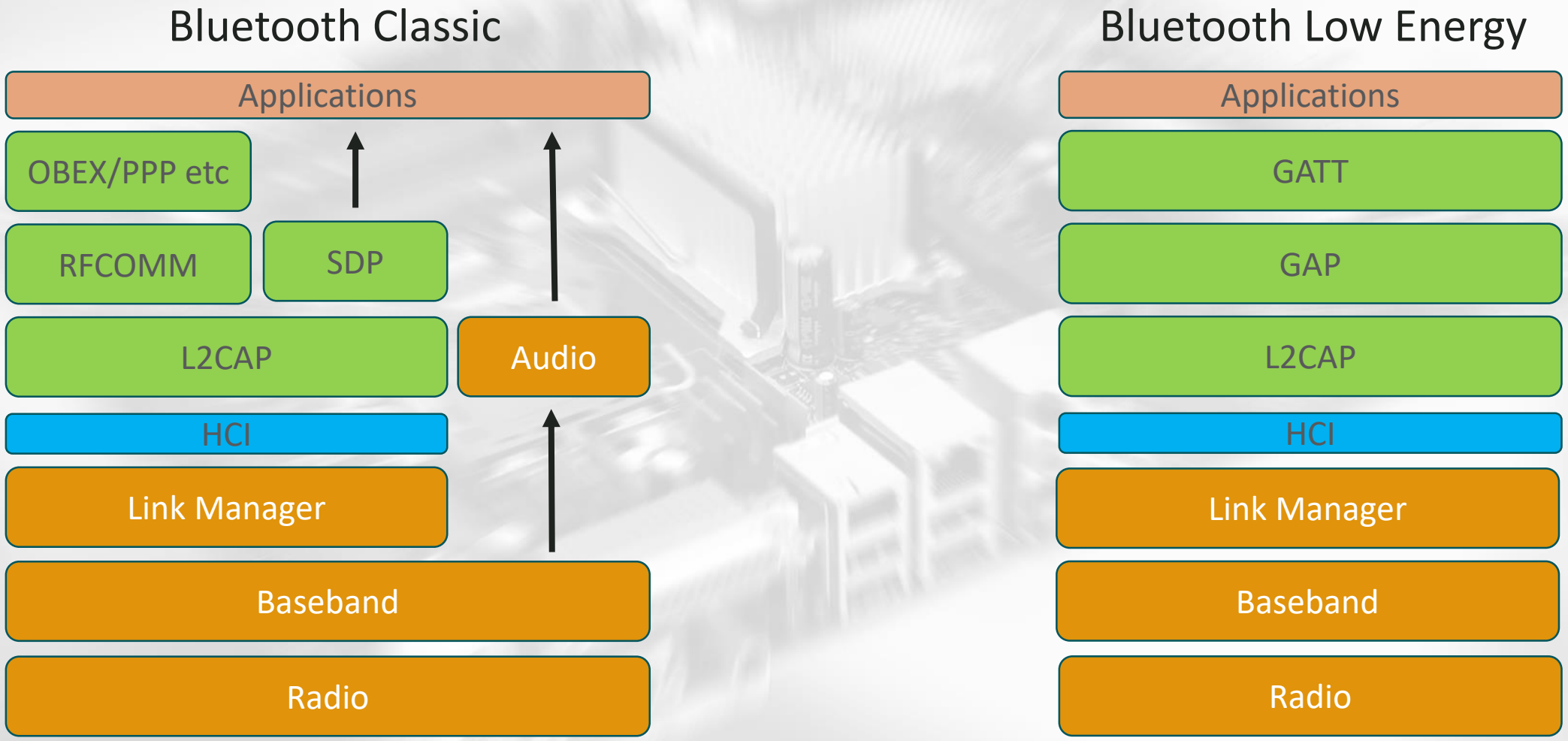
What is it?

- A simplification of the Bluetooth protocol stack
- Designed for low power (duh) applications
- Simplified RF
- Simplified interface
- Simplified security

Basic data types

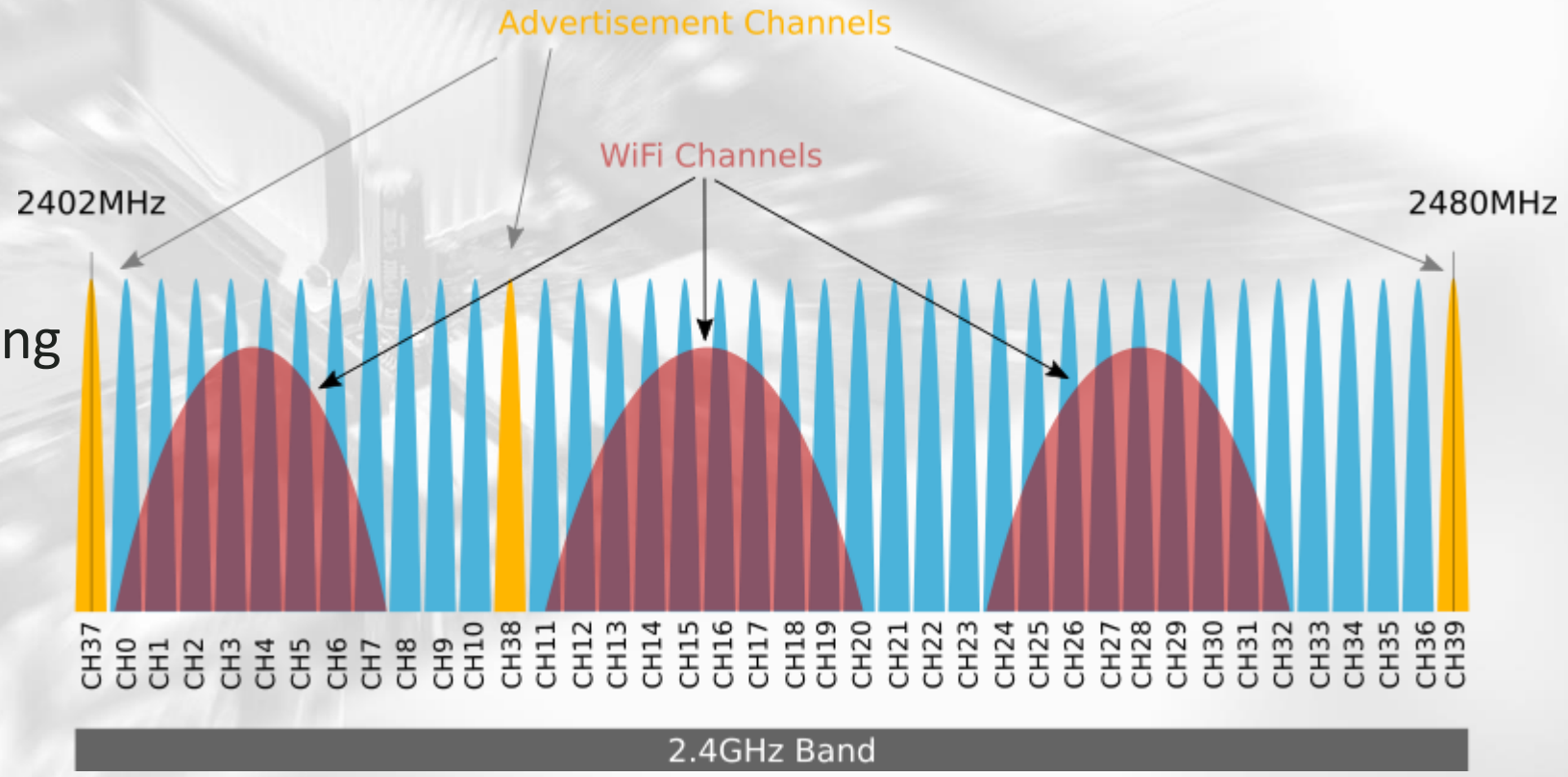
- Octet instead of byte
- **Everything** is little endian
- UUIDs can have variable lengths:
  - 128-bit: `dcc828ef-5799-4072-b6c3-ad7abf5b72eb`
  - 32-bit: `xxxxxxxx-0000-1000-8000-00805F9B34FB`
  - 16-bit: `0000xxxx-0000-1000-8000-00805F9B34FB`

What is it?


Bluetooth Classic
Applications
OBEX/PPP etc
RFCOMM
SDP
L2CAP
Audio
HCI
Link Manager
Baseband
Radio
Bluetooth Low Energy
Applications
GATT
GAP
L2CAP
HCI
Link Manager
Baseband
Radio

The RF Stack

- Uses 40 channels (at 2.4 GHz)
- Three are for advertising
- 37 for data
- Simplified frequency hopping



Addressing

- All devices have a BD_ADDR, an EUI-48
  - df:d7:fa:b6:f4:50

- This can be public (i.e. a real address):

company_assigned			company_id		
LSB					MSB

- Or random:

Public random address		
Random part of address	1	1

Addressing

- Or, just to complicate, non-resolvable private:

Non-resolvable private address		
Random part of address	0	0

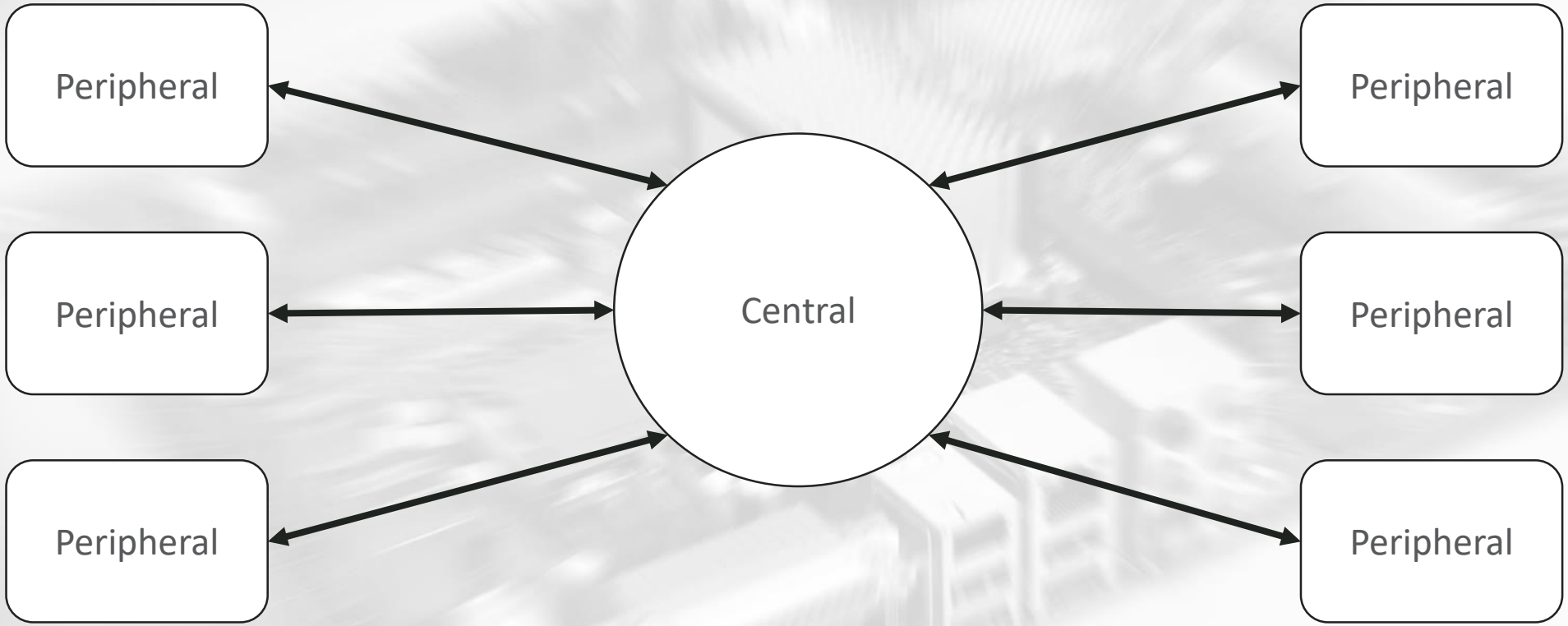
- Or resolvable private:

Resolvable private address			
hash	prand	1	0

GAP

- Generic Access Profile
- Handles connection and advertising
- Basically from nowt to talking over GATT

Device types



Packet Format

- Below format used for all packets on the Link Layer
  - GAP
  - Advertising

LSB			MSB
Preamble (1 octet)	Access Address (4 octets)	PDU (2 to 257 octets)	CRC (3 octets)

- Preamble defines type
- Address depends on type:
  - Data is random
  - Advertising is specific

Advertising

- Highlights what devices are available
- Uses three channels:

RF Channel	Centre Frequency (MHz)	BLE Channel
0	2402	37
12	2426	38
39	2480	39

Advertising

- Preamble set to 10101010b
- Access Address set to 10001110100010011011111011010110b

LSB			MSB
Preamble 0xAA	Access Address 0x8E89BED6	PDU (2 to 257 octets)	CRC (3 octets)

- PDU

LSB						MSB
Header						Payload
PDU type (4 bits)	RFU (2 bits)	TXAdd (1 bit)	RXAdd (1 bit)	Length (6 bits)	RFU (2 bits)	(Length octets)

Advertising

- PDU Types:
  - ADV_ID – connectable undirected advertising
  - ADV_DIRECT_IND – connectable directed advertising
  - ADV_NONCONN_IND – non-connectable undirected advertising
  - ADV_SCAN_IND – scannable undirected advertising

  - SCAN_REQ – scanning request
  - SCAN_RSP – scanning response

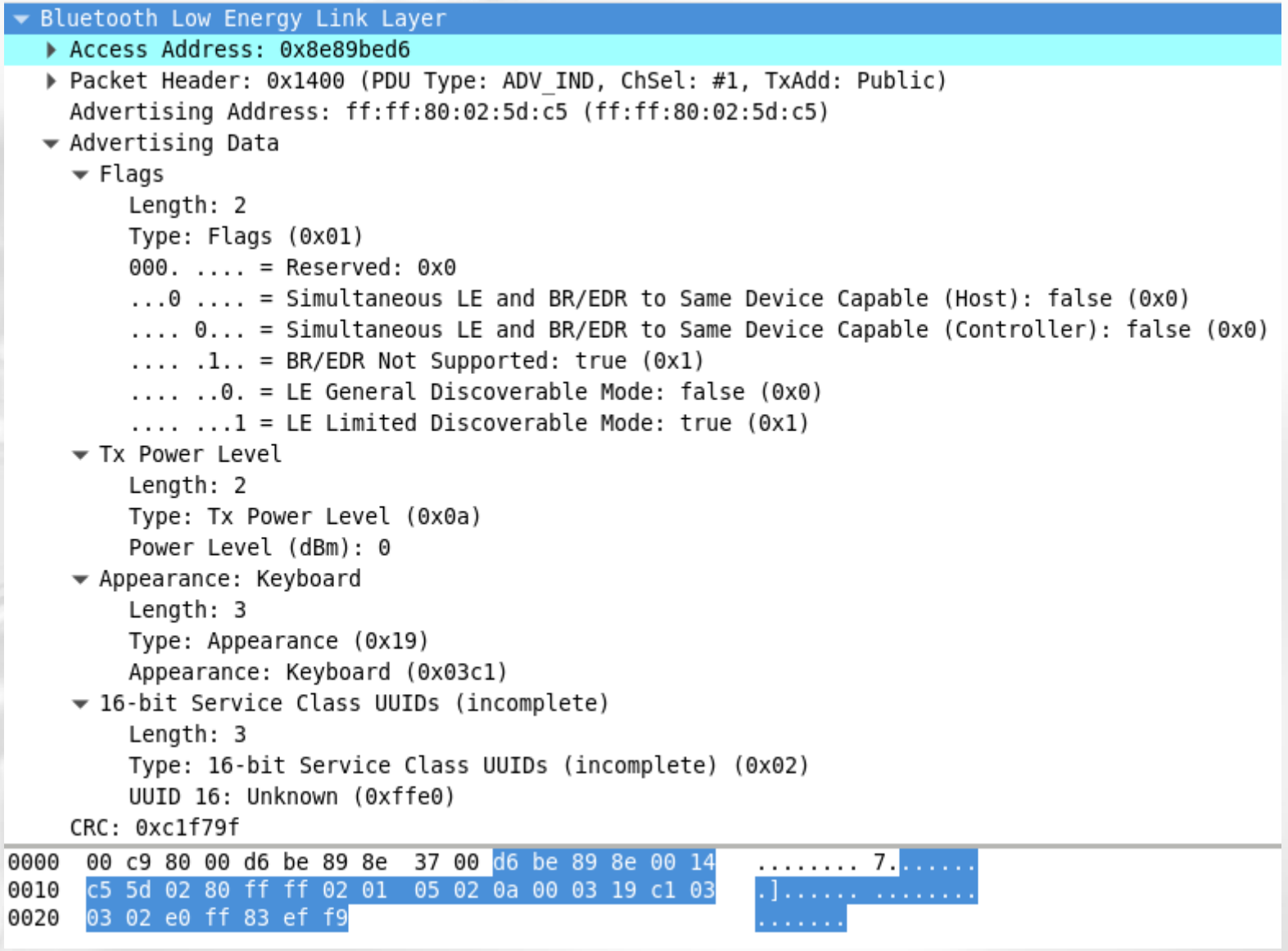
  - CONNECT_REQ – connection request

Advertising

- AD data is sent in in ADV_*_IND and SCAN_RSP packets
- Defined outside the main specification
  - In Core Specification Supplement (CSS) – this defines data types
- Includes stuff like:
  - Service UUID
  - Local Name
  - URI

Advertising

- Let's see this in Wireshark
- Using Ubertooth-One to sniff



Advertising - Practical

- Using Bluez tools – hcitool
  - hcitool lescan --passive
  - hcitool lescan
  - hcitool leinfo <address>

```
[dave@mictlan ~]$ sudo hcitool lescan
LE Scan ...
FF:FF:80:02:5D:C5 (unknown)
FF:FF:80:02:5D:C5 iTAG
FF:FF:80:02:5D:C5 (unknown)
32:FD:7D:04:4C:71 (unknown)
```

```
[dave@mictlan ~]$ sudo hcitool leinfo FF:FF:80:02:5D:C5
Requesting information ...
        Handle: 69 (0x0045)
        LMP Version: 4.0 (0x6) LMP Subversion: 0x4103
        Manufacturer: Telink Semiconductor Co. Ltd (529)
        Features: 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00
```

- Using bleah
  - bleah
  - bleah <address>

```
ff:ff:80:02:5d:c5 (-74 dBm)
Vendor                  | ?
Allows Connections      | ✓
Flags                   | LE Limited Discoverable, BR/EDR
Tx Power                | u'00'
Complete Local Name     | iTAG
Incomplete 16b Services | u'e0ff'
Appearance              | u'c103'
```

Connections

- First – forget everything about Bluetooth
- I'm serious

- Pairing in BLE is something different
- We don't pair, we connect
- We must connect before we can do anything

- See CONNECT_REQ above

- BT 4.0 restricts number of simultaneous connections to 1

GATT and ATT

- Generic Attribute Profile and Attribute Protocol
- How we read and send data to a peripheral
- GATT is build on ATT
- GATT adds structure that can be understood by a Central Device
- Most of the time you'll be using GATT

ATT

- ATT is a big table of attributes
- Each attribute has properties:

- Type – defined by 32-bit or 128-bit UUID
- Handle – 16 bit uint, 0x0000 is reserved
- Value – An array of unformatted octets
- Permissions – divided into:
  - Access (read, write, read/write)
  - Encryption (encryption required/not required)
  - Authentication (authentication required/not required)
  - Authorisation (authorisation required/not required)

Handle	Value
1	00 18
2	12
3	69 54 41 47 20 20 20 20 20 20 20 20 20 20 20 20
4	02
5	00 00
6	0f 18
7	12
8	63
9	02 18
10	1c
11	00
12	e0 ff
13	12
14	00

- Attributes that are write only are known as “Control-Point Attributes)

ATT - Practical

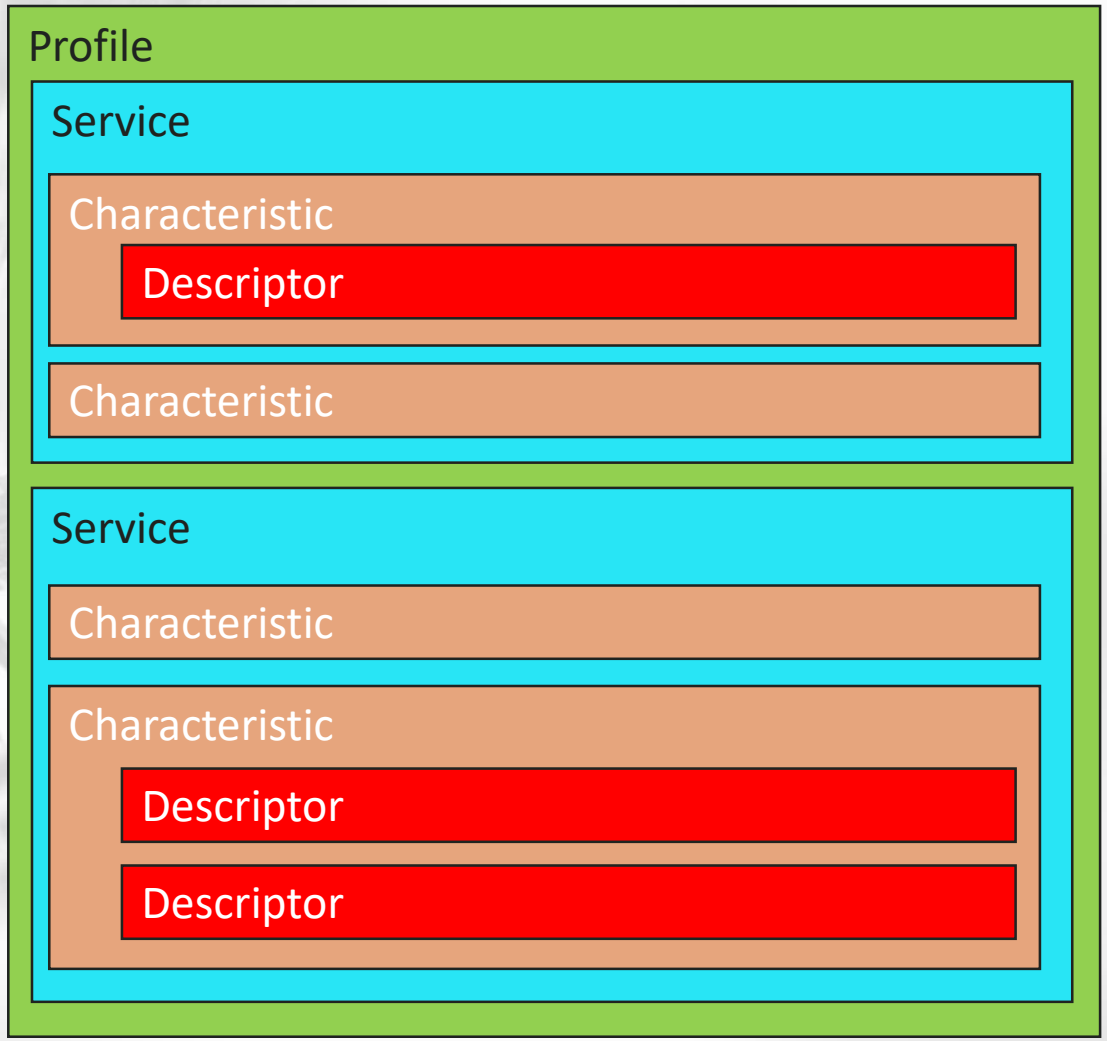
- Read an attribute
  - gatttool -b FF:FF:80:02:5D:C5 --char-read --handle 1

- Questions
  - What is the content of handle 10? 0x1c
  - How many handles can you read? 14
  - Can you write to any handles (--char-write) Handle 12 only

GATT

- GATT builds on ATT
- Adds structure for data types

- Hierarchy
  - Profile
  - Service
  - Characteristics
  - Descriptors



GATT - Profile

- Contains one or more services
- Fulfils a use case
- That’s pretty much it

GATT - Service

- Collection of data to fulfil a function
- Two types:
  - Primary – the primary function of the device
  - Secondary – auxiliary functions, referenced by the primary services
- Referenced service is a pointer to another service
  - Unlimited nesting
- No limitations on number of services
- Each service has multiple characteristics

GATT - Characteristic

- Maps to an attribute in ATT
- Can contain data or be written to
- Properties:
  - UUID – a unique UUID to reference the characteristic
  - Handle – the ATT handle
  - Properties – the permissions on the characteristic – are not quite the same as the ATT permissions
  - Security options – whether we need to pair/bond to read the characteristic
  - Data – the data the characteristic holds

GATT – Contains structure

Handle	Value	Service	Characteristic	Type	Meaning
1	00 18	Generic Access		Primary service 16 bit UUID	00001800-0000-1000-8000-00805f9b34fb
2	02 03 00 00 2a		Devicename	Characteristic descriptor	Permissions: read; Handle: 3; UUID: 2a00
3	4d 61 73 74 65 72 20 4c 6f 63 6b			Characteristic value	“Master Lock“
4	02 05 00 01 2a		Appearance	Characteristic descriptor	Permissions: read; Handle: 5; UUID: 2a01
5	00 00			Characteristic value	00 00
6	02 07 00 04 2a		Peripheral Preferred Connection Parameters	Characteristic descriptor	Permissions: read; Handle: 7; UUID: 2a04
7	10 00 20 00 00 00 1e 00			Characteristic value	connInterval_min: 20; connInterval_max: 40; Slave Latency: 0; connTimeout: 300
8	0a 18	Device Information		Primary service 16 bit UUID	0000180a-0000-1000-8000-00805f9b34fb
9	02 0a 00 29 2a		Manufacturer Name	Characteristic descriptor	Permissions: read; Handle: 10; UUID: 2a29
10	4d 61 73 74 65 72 20 4c 6f 63 6b			Characteristic value	“Master Lock“
11	fb 6d b3 e6 37 44 6f 84 e4 11 5b 5d 02 00 e0 94	Lock		Primary service 128 bit UUID	94e00002-5d5b-11e4-846f-4437e6b36dfb
12	1e 0d 00 fb 6d b3 e6 37 44 6f 84 e4 11 5b 5d 02 00 e0 94		Lock	Characteristic descriptor	Permissions: notify, write, write without response, read; Handle: 13; UUID: 94e00002-5d5b-11e4-846f-4437e6b36dfb
13	00			Characteristic value	No value
14	00 00			Client Characteristic Configuration Descriptor	00 00

GATT - Characteristics

- Can be referenced by a UUID or handle

```
[dave@mictlan ~]$ sudo gatttool -b FF:FF:80:02:5B:41 --char-read --handle 0x0003
Characteristic value/descriptor: 69 54 41 47 20 20 20 20 20 20 20 20 20 20 20 20
[dave@mictlan ~]$ sudo gatttool -b FF:FF:80:02:5B:41 --char-read --uuid 00002a00-0000-1000-8000-00805f9b34fb
handle: 0x0003   value: 69 54 41 47 20 20 20 20 20 20 20 20 20 20 20 20
```

- Properties
  - Broadcast
  - Read
  - Write without response
  - Write
  - Notify
  - Indicate
  - Authenticated signed writes
  - Extended properties

GATT - Characteristics

- A characteristic can be static data or programmatic
  - What is the value of handle 8?
  - `gatttool -b ff:ff:80:02:5b:41 --char-read --handle 8`
  - What do you think this is used for?

- More than just read:
  - What happens if we write a value to handle 0xb?
  - `gatttool -b ff:ff:80:02:5b:41 --char-write-req --handle 0xb --value 0x1`

GATT - Characteristics

- Notifications and indications
  - Sends a callback to the central device if a characteristic changes
- Try this python script:
  - https://github.com/pentestpartners/snippets/blob/master/ble-itag.py
- Try running it and pressing the button.
- The magic is this bit:
  - `svc=p.getServiceByUUID(UUID("0000ffe0-0000-1000-8000-00805f9b34fb"))`
  - `ch=svc.getCharacteristics( UUID("0000ffe1-0000-1000-8000-00805f9b34fb"))[0]`
  - `noch=ch.getHandle()+1`
  - `p.writeCharacteristic(noch,b'\x01\x00')`

GATT - Descriptors

- We've already seen some of their uses
  - Describing characteristics
  - As hooks for notifications
- Can also be comments
  - Attribute containing a clear text string (UTF-8), after a characteristic's value

GATT – Tool usage

- Gatttool – we've seen some examples, also
  - --primary – list primary services
  - --characteristics – list characteristics
  - for i in `seq 1 20`;do gatttool -b $device --read-char --handle $i;done
- Bleah – a lot easier
  - bleah --b $device
  - bleah --b $device -e

GATT - Security

- 99% of devices do not use security

- A characteristic can be marked as needing security with security options.
- LE Security Mode 1 is encryption:
  1. None
  2. Unauthenticated pairing with encryption (Just works)
  3. Authenticated pairing with encryption (OOB, passkey)
  4. LE Secure connections pairing with encryption (4.2 only)

- LE Security Mode 2 is data signing:
  1. Unauthenticated pairing with data signing
  2. Authenticated pairing with data signing

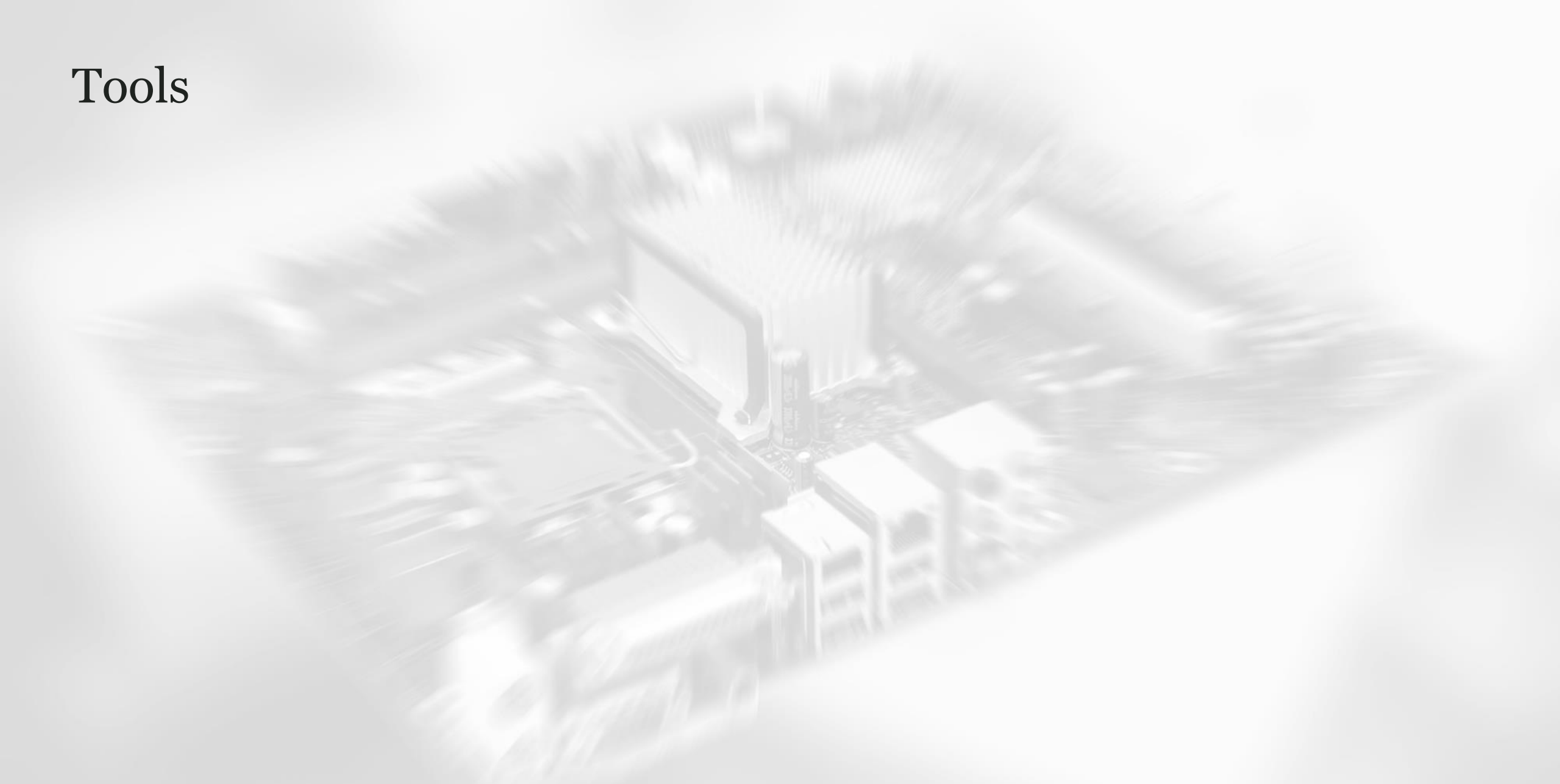
GATT - Security

- Pairing is not the same as BR/EDR
- Connection is connecting at mode 1, level 1
  - 4.0 and 4.1 may restrict simultaneous connections to a peripheral
- Pairing is using an encryption key and not storing it
- Bonding is storing an encryption key permanently

GATT - Security

- Phases of encryption:
- Talk over ATT/GATT, unencrypted
- Generate an STK and exchange it
  - 4.2 will generate an LTF at this point
- Generate an LTK and other keys
  - CSRK for encryption
  - IRK for private MAC address

Tools



Tools – hcitool

- hcitool – talks to HCI layer
- lescan
  - Scans for broadcast advertising beacons
  - Useful for identifying devices

```
[dave@mictlan ~]$ sudo hcitool lescan
LE Scan ...
FF:FF:10:15:2E:88 (unknown)
FF:FF:10:15:2E:88 iTAG
D0:87:45:FD:40:59 Tile
D0:87:45:FD:40:59 (unknown)
FF:FF:80:02:5B:41 (unknown)
FF:FF:80:02:5B:41 iTAG
```

- leinfo
  - Gives some basic information

```
[dave@mictlan ~]$ sudo hcitool leinfo FF:FF:10:15:2E:88
Requesting information ...
        Handle: 69 (0x0045)
        LMP Version: 4.0 (0x6) LMP Subversion: 0x4103
        Manufacturer: Telink Semiconductor Co. Ltd (529)
        Features: 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00
```

Tools – gatttool

- gatttool – talks to GATT/ATT
  - CLI mode
  - Interactive mode

- -b specifies address
- Need to manually specify random (-t)

- Enumerate stuff:
  - --primary – services
  - --characteristics – list characteristics

```
[dave@mictlan ~]$ gatttool -b FF:FF:10:15:2d:18 --primary
attr handle = 0x0001, end grp handle = 0x0005 uuid: 00001800-0000-1000-8000-00805f9b34fb
attr handle = 0x0006, end grp handle = 0x0008 uuid: 0000180f-0000-1000-8000-00805f9b34fb
attr handle = 0x0009, end grp handle = 0x000b uuid: 00001802-0000-1000-8000-00805f9b34fb
attr handle = 0x000c, end grp handle = 0x000e uuid: 0000ffe0-0000-1000-8000-00805f9b34fb
```

Tools – gatttool

- Read/write characteristics:
  - char-read
  - char-write - Write
  - char-write-req – Write without response
- Read indicate/notify
- Gets flaky beyond simple read/write

Tools – bleah

The new gatttool

Written by evilsocket (bettercap)

Looks prettier

- bleah
- bleah –b $device –e

Tools – bleah

```
@ Scanning for 5s [-128 dBm of sensitivity] ...

┌ ff:ff:10:15:2d:04 (-51 dBm) ──────────────────────────┐
│ Vendor                    │ ?                         │
│ Allows Connections        │ ✓                         │
│ Flags                     │ LE Limited Discoverable, BR/EDR │
│ Tx Power                  │ u'00'                     │
│ Complete Local Name       │ iTAG                      │
│ Incomplete 16b Services   │ u'e0ff'                   │
│ Appearance                │ u'c103'                   │
└───────────────────────────┴───────────────────────────┘

@ Connecting to ff:ff:10:15:2d:04 ... connected.
@ Enumerating all the things ....

┌──────────────┬──────────────────────────────────────────────────────────────┬──────────────────────────────┬──────────────────┐
│ Handles      │ Service > Characteristics                                    │ Properties                   │ Data             │
├──────────────┼──────────────────────────────────────────────────────────────┼──────────────────────────────┼──────────────────┤
│ 0001 -> 0005 │ Generic Access ( 00001800-0000-1000-8000-00805f9b34fb )      │                              │                  │
│ 0003         │   Device Name ( 00002a00-0000-1000-8000-00805f9b34fb )       │ NOTIFY READ                  │ u'iTAG           '│
│ 0005         │   Appearance ( 00002a01-0000-1000-8000-00805f9b34fb )        │ READ                         │ Unknown          │
│              │                                                              │                              │                  │
│ 0006 -> 0008 │ Battery Service ( 0000180f-0000-1000-8000-00805f9b34fb )     │                              │                  │
│ 0008         │   Battery Level ( 00002a19-0000-1000-8000-00805f9b34fb )     │ NOTIFY READ                  │ u'c'             │
│              │                                                              │                              │                  │
│ 0009 -> 000b │ Immediate Alert ( 00001802-0000-1000-8000-00805f9b34fb )     │                              │                  │
│ 000b         │   Alert Level ( 00002a06-0000-1000-8000-00805f9b34fb )       │ NOTIFY WRITE NO RESPONSE WRITE │                │
│              │                                                              │                              │                  │
│ 000c -> 000e │ ffe0 ( 0000ffe0-0000-1000-8000-00805f9b34fb )                │                              │                  │
│ 000e         │   ffe1 ( 0000ffe1-0000-1000-8000-00805f9b34fb )              │ NOTIFY READ                  │ '\x01'           │
└──────────────┴──────────────────────────────────────────────────────────────┴──────────────────────────────┴──────────────────┘
```

Tools – bluepy

Python library

Not going to teach

Best for programmatically doing stuff

Example template:
https://github.com/pentestpartners/snippets/blob/master/ble-itag.py

Hardware

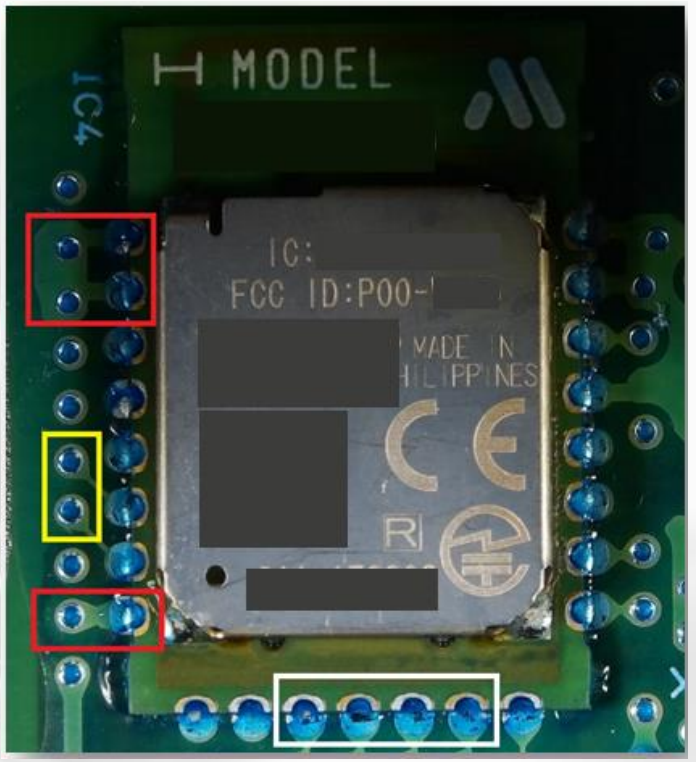
- Two main techniques:
  - Use of BLE module
  - Use of BLE MCU, often controls the whole board
- Often based on age and complexity of device
- Modules are often easier

Hardware - Modules

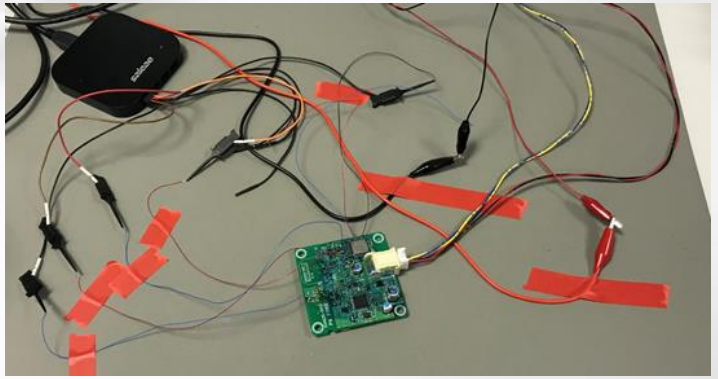
- Easy to just add in
- Uses SPI or UART (or both) to communicate with main board
- Raspberry Pi does BLE this way
- Easiest to intercept:
  - Find datasheet
  - Solder wires to pads or nearby vias/resistors/capacitors
  - Use logic analyser
  - Profit

Hardware - Modules

- Had to censor this, as customer device
  - Yellow – UART
  - Red – Ground
  - White – SPI

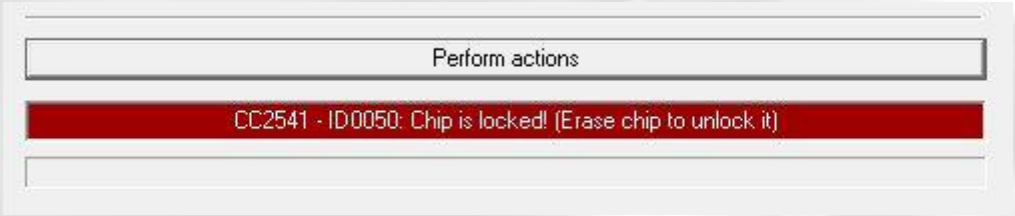



```
OK
BTRECV:11,7BC292E010F72EA324E52EE8E83D2C745E
BTRECV:11,80213BBBEC5FF61C62467DE05C30936D9B
OK
BTRECV:11,44C6342F5CBABE21A72AABB7FE55877D53
BTDONE:0,43528E9B6F65
BTRECV:11,440D4ED9C6B7AE140660BA39317B6AA007
OK
BTDONE:0,43528E9B6F65
BTRECV:11,445F71BB865BBC38032927E5C79CFF8B51
OK
BTDONE:0,43528E9B6F65
BTRECV:11,4401FA4AD24D61CD6AD73DE7161882C3F4
OK
BTDONE:0,43528E9B6F65
BTRECV:11,44E2CE7CDFDDD664A89A0B7CD25F9A3CE8
BTRECV:11,442771483910BA8EE51E44794508C99B5A
```



Hardware - MCU

- Depends a lot on MCU
- We're looking for stuff like on the right
- MCU dependant, in this case CC2540:
  - DC – Debug Clock
  - DD – Debug Data
  - RESET – RESET pin
  - GND – Ground
  - VCC – Vcc

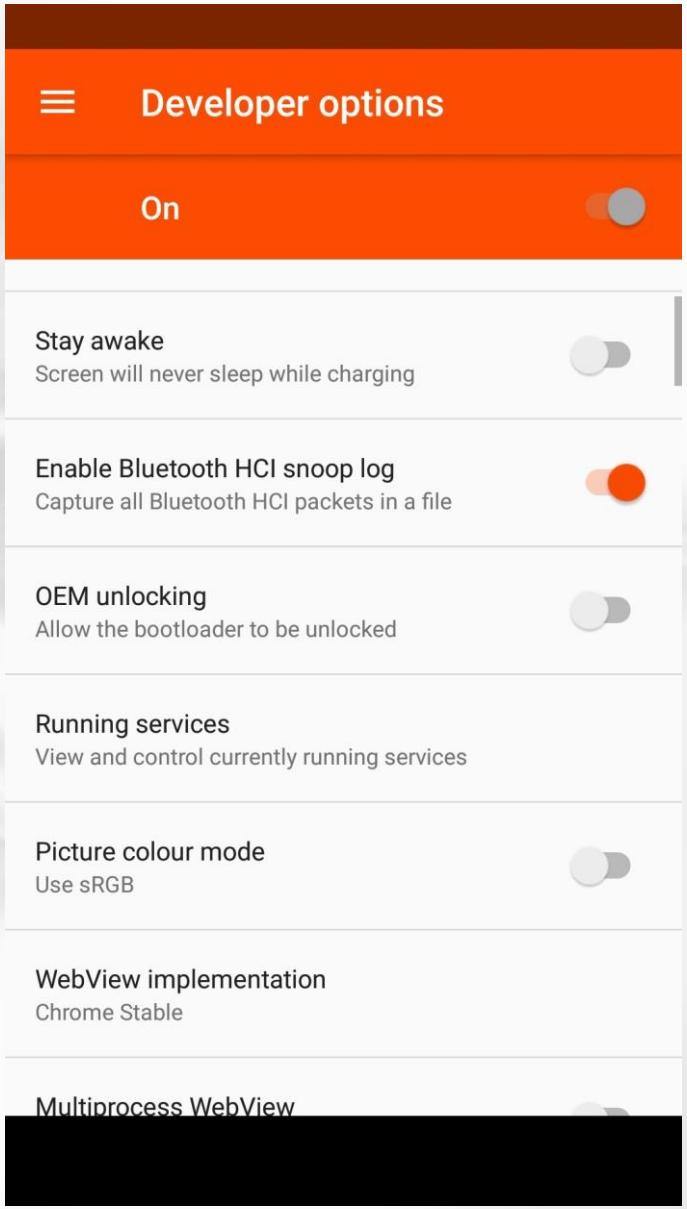


Android as a BLE Hacking tool

- Android offers many options for developers to work with BLE
- There are a few ways you can analyse BLE traffic on Android:
  - HCI Snoop
  - Apps
    - nRF Connect
    - BLE Scanner

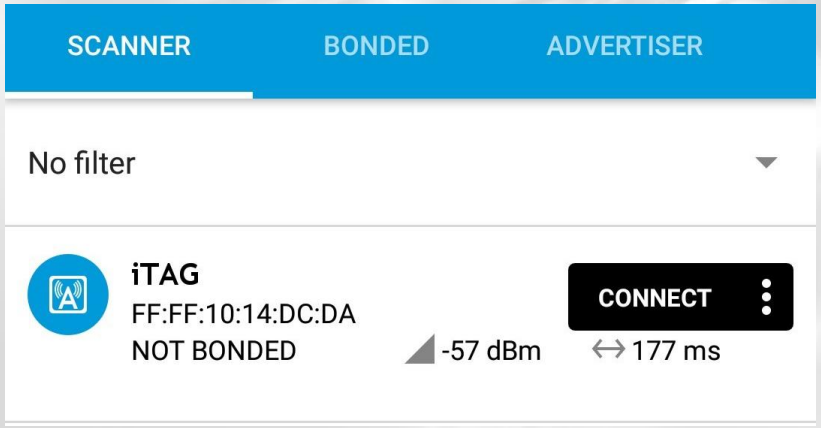
Android – HCI Snoop

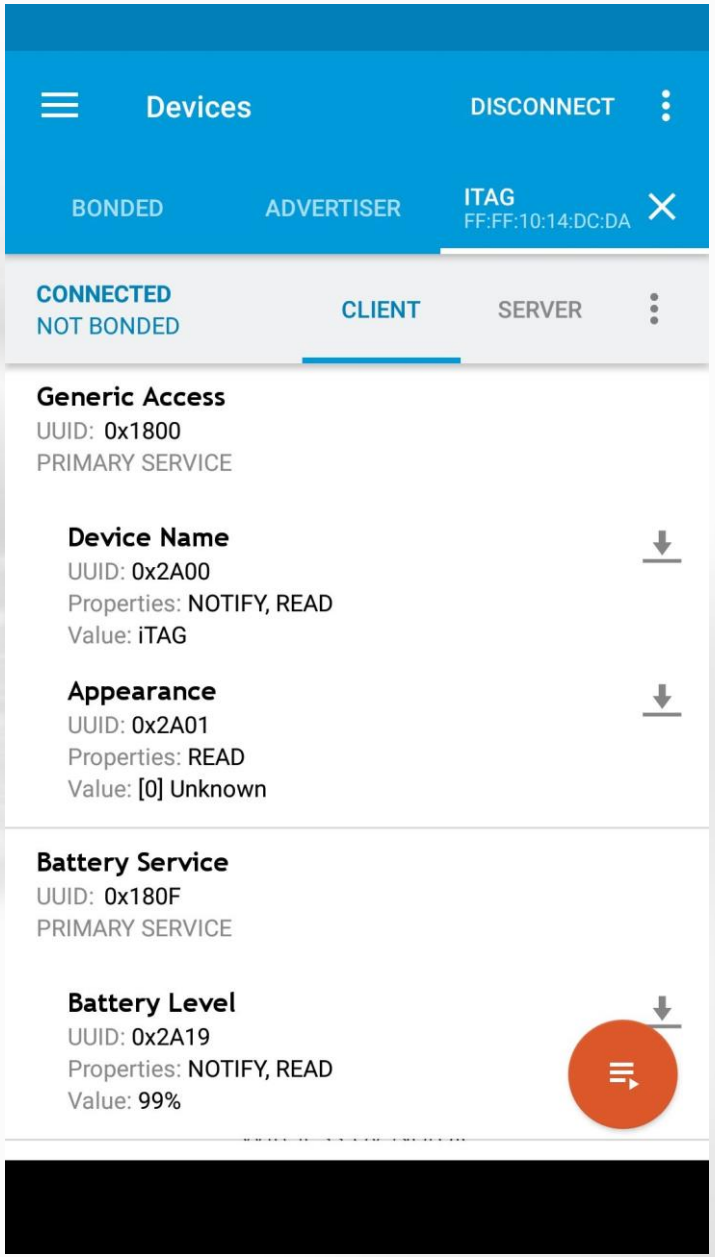
- Enable in Developer Settings
  - Settings -> Developer Options -> Enable Bluetoothe HCI Snoop Log
  - Does NOT require rooting!
- Creates a log file hci_snoop_XXXX.log
  - Usually in /data/sdcard/
- This can be opened and parsed in wireshark!
- Very useful for recording all traffic between an app and a BLE enabled device
- Gives a good starting point for IoT device investigations/research/testing



Android – nRF Connect

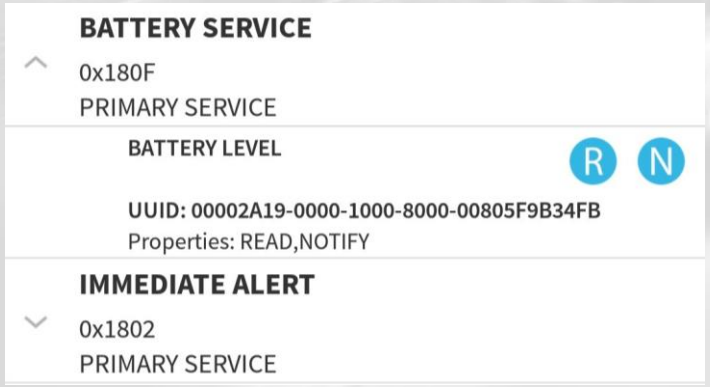
- Enable in Developed by nRF for use in developing BLE devices (e.g. with nRF51822)
- Gives an easy-to-use interface for detecting and interacting with BLE devices
- Shows characteristics and services
- Allows the user to send strings/bytes to a characteristic if WRITE is enabled
- Handles bonding/pairing seamlessly

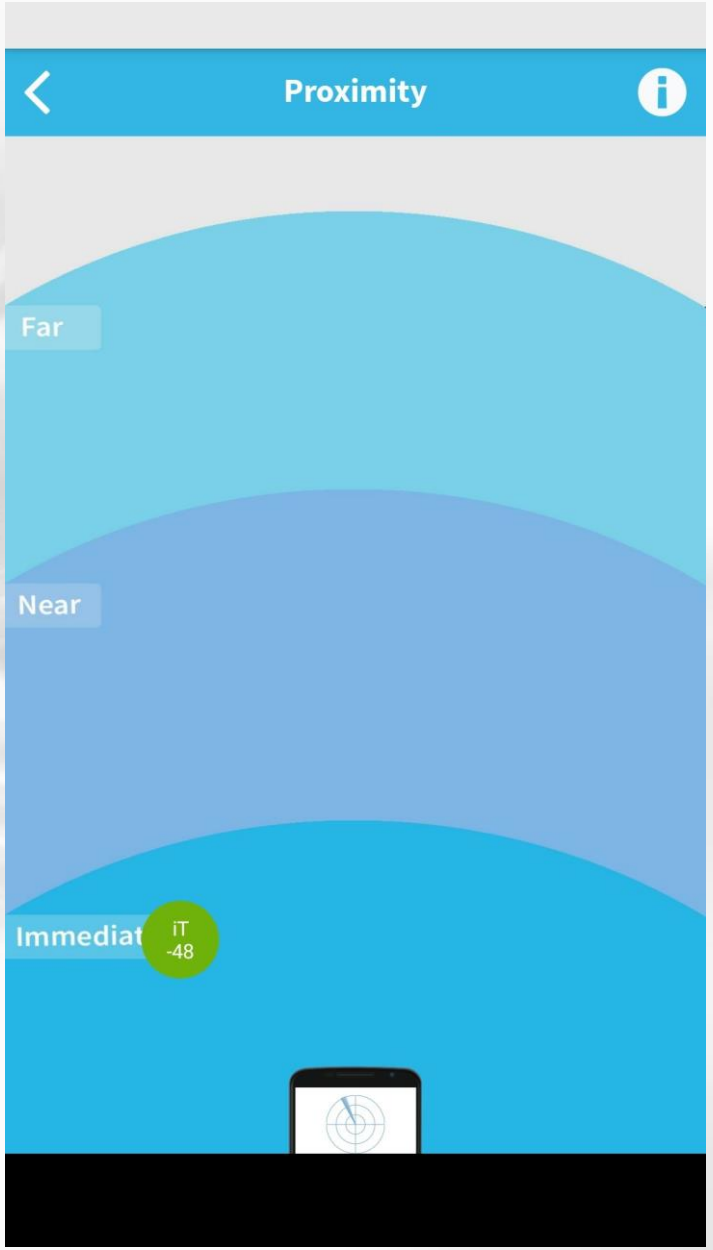




Android – BLE Scanner

- A free android app for use in BLE testing
- Has some nice features like other apps
- Live map based off RSSI values for BLE device
- Easy to use interface
- Automatically gets characteristic data from devices
- Easy to read/write and get notifications with a smooth interface





Capture the Flag

- I have set up a couple of Pis with a BLE CTF
- Six flags to find
- All hidden around BLE stuff
- Any encoding is trivial (e.g. ASCII to Hex)
- Prize for the person with the most flags/fastest person
  - Prize arrived at my house yesterday
  - I'm not at my house
  - F**king postal service

BONUS!! The Annoyatron 2000!!

- Remember those little ‘annoyatron’ PCB projects?
- We have updated it for the 21st century and BLE age!
- Searches for iTAG devices
- If it finds some, it chooses one at random and sets the ‘Notify’ bit to 0x02 (high alert)
- It then waits for a random amount of time
- See the gitlab snippet!

https://gitlab.com/snippets/1720857

@tautologyo
@LargeCardinal

info@pentestpartners.com
+44 (0)20 3095 0500
@PenTestPartners
PenTestPartnersLLP