OPEN SOURCE SUMMIT

China 2019

OPEN SOURCE SUMMIT
China 2019

Unfit Story of Fitness Trackers : Hacking the BLE devices

Yogesh Ojha

- Cyber Security Analyst @ TCS, India
- IOT & Mobile Application Security

Expectations

You can expect to learn about:

- Basic Understanding of Bluetooth
- Bluetooth Classic vs Bluetooth Low Energy
- BLE Stack
- Capturing BLE Packets/BLE MiTM
- Reverse Engineering the Mobile Applications of Fitness trackers
- Uploading the firmware over the air

Bluetooth Story...

Bluetooth is a short-range wireless communication protocol and allows devices such as smartphones, headsets, to transfer data and/or voice wirelessly.

Developed in 1994 as a replacement for cables.

Uses 2.4GHz frequency and creates 10 meters radius called piconet!

Bluetooth Low Energy (4.0)

Bluetooth low energy aka Bluetooth Smart

- Designed to be power efficient
- Low cost and easy to implement
- Used in sensors, lightbulbs, medical devices, wearables and many other “smart” products.

Bluetooth
SMART

Bluetooth classic vs BLE

Bluetooth Classic

- Great for products that requires continuous streaming of data
- High power consumption
- Faster data rate
- High application throughput
- Best Suited for:
  - Headsets, Speakers
  - Bluetooth Hotspot etc

Bluetooth Low Energy

- Great for products that do not require continuous streaming of data.
- Ultra low power consumption
- Slower Data rate
- Low application throughput
- Best Suited for:
  - Home Automation
  - Fitness trackers etc

It is designed to operate in sleep mode and waken up only when connection is initiated. Like maybe your light is on or off or a quick command to turn on or off the light.

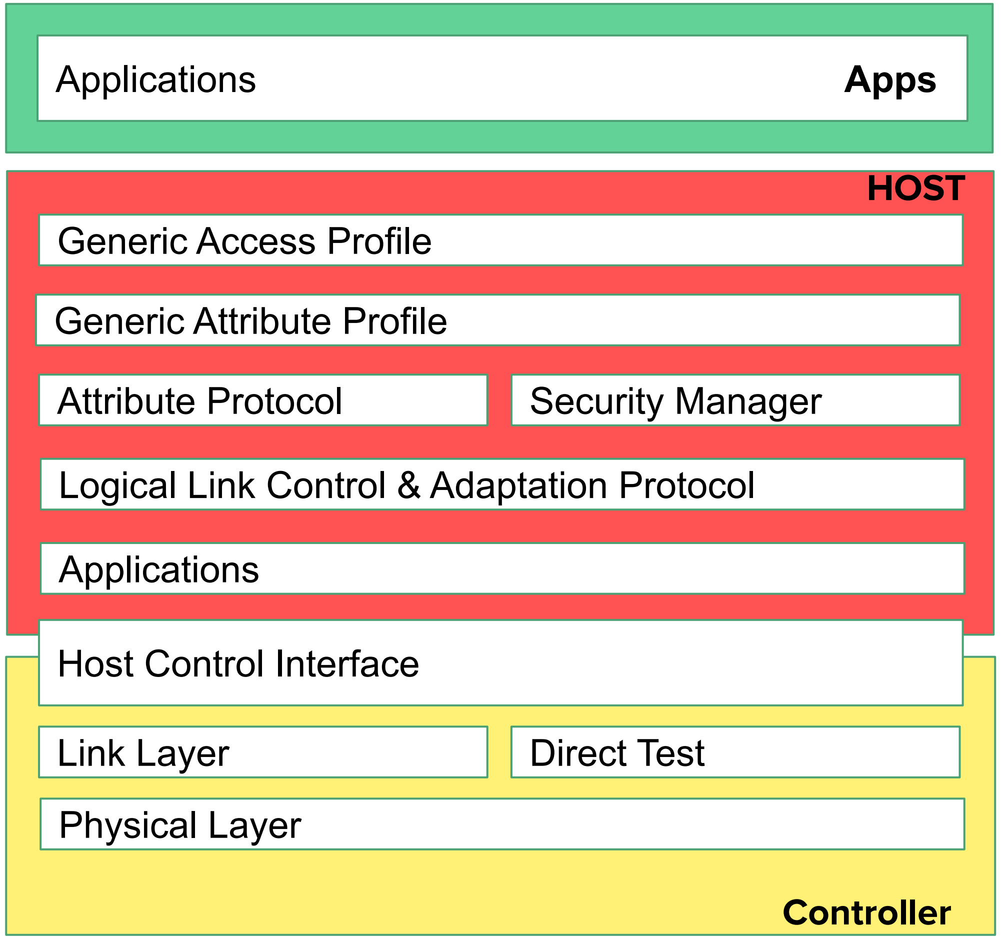
Bluetooth Low Energy

(4.0)

Fitness Tracker - BLE Applications

BLE Stack

- Generic Attribute Profile (GATT)
- Generic Access Profile(GAP)



Generic Attribute Profile (GATT)

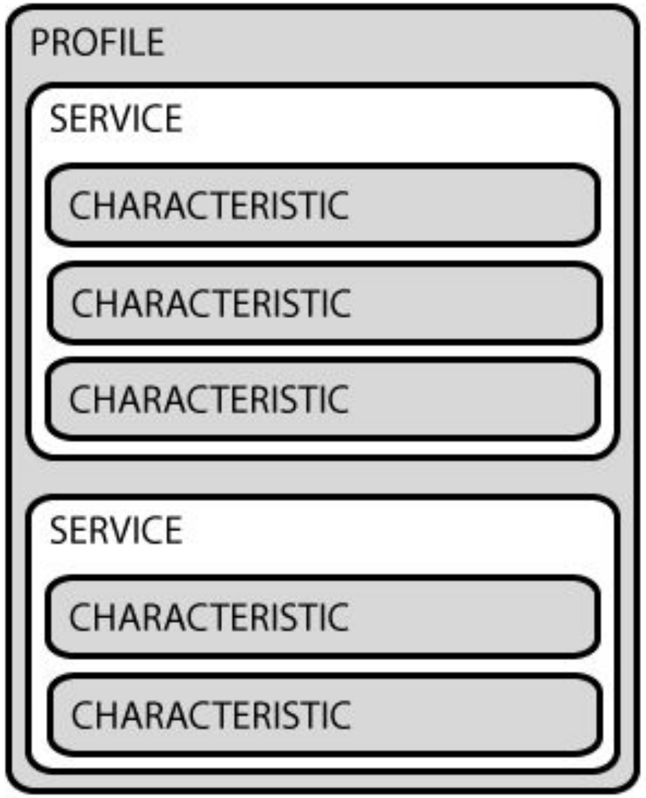
GATT defines the way that these BLE devices communicate with each (client & server) other using something called **Services** and **Characteristics**.

Here Connections are Exclusive! Means your BLE peripheral can only be connected to one central device at a time! It will stop advertising itself and other devices will no longer be able to see it or connect to it until the existing connection is broken.

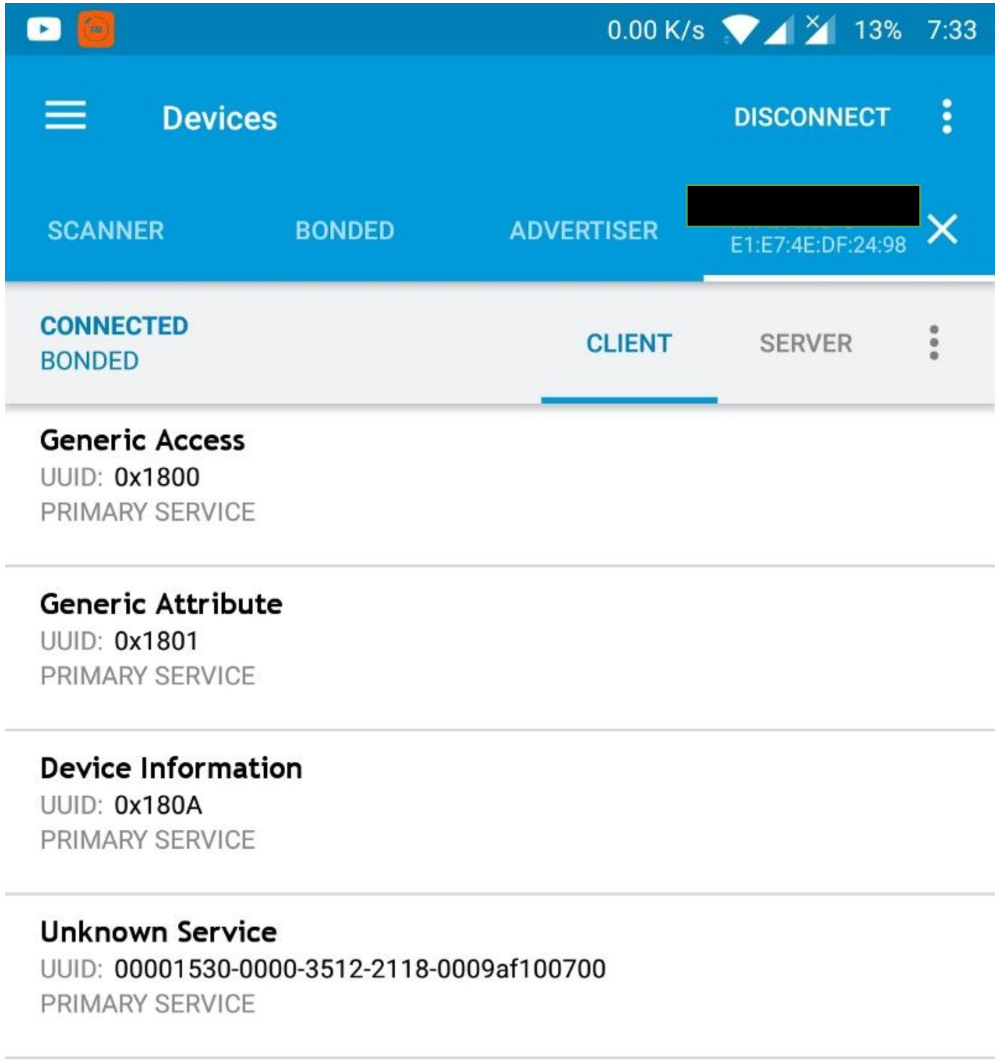
Services & Characteristics

Services: Set of provided features and associated behaviors to interact with the peripheral. Each service contains a collection of characteristics.

Characteristics: Characteristics are defined attribute types that contain a single logical value.



Services & Characteristics



Generic Access
UUID: 0x1800
PRIMARY SERVICE

Generic Attribute
UUID: 0x1801
PRIMARY SERVICE

Device Information
UUID: 0x180A
PRIMARY SERVICE

Unknown Service
UUID: 00001530-0000-3512-2118-0009af100700
PRIMARY SERVICE

Basic Process

0. Select the target
    a. Install Bluez stack, hcitool & gattool
1. Enumerate the **services** and **characteristics**
    a. Do the scan using hcitool
    b. Connect using gatttool
    c. List all the services and characteristics
2. Reverse Engineer the mobile application (if any)
    a. For reverse engineering android application use apktool.
3. Finally do some cool stuff!

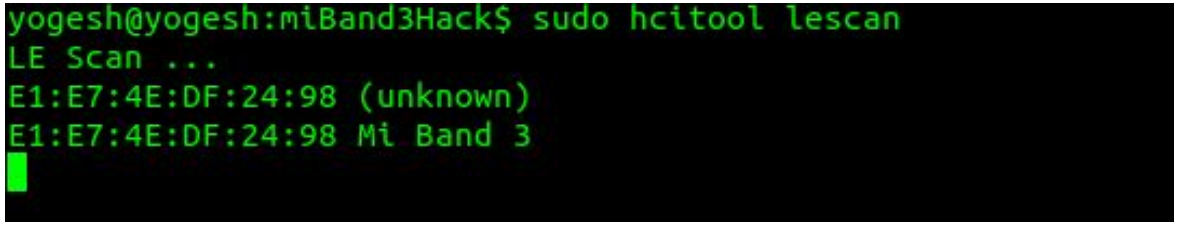
0. Selecting the target

Goal: Finding the BLE devices near the vicinity

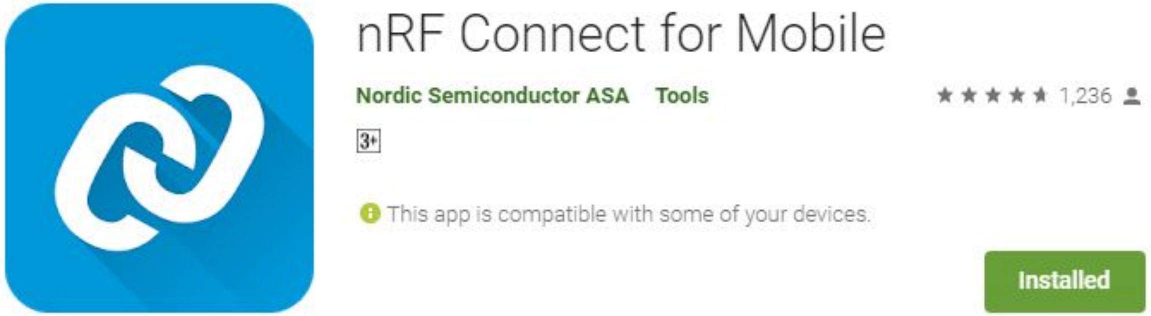
Tools Used: **Bluez, hcitool, gatttool**

Install Bluez: $ sudo apt-get install bluez

Install Hcitool: hcitool comes **preinstalled with bluez stack**



```
yogesh@yogesh:miBand3Hack$ sudo hcitool lescan
LE Scan ...
E1:E7:4E:DF:24:98 (unknown)
E1:E7:4E:DF:24:98 Mi Band 3
```

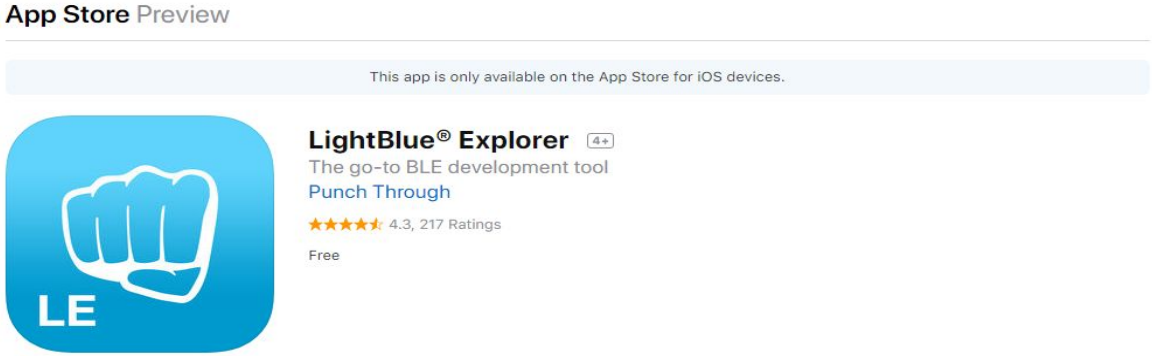


nRF Connect for Mobile

Nordic Semiconductor ASA Tools ★★★★★ 1,236

3+

This app is compatible with some of your devices.



App Store Preview

This app is only available on the App Store for iOS devices.

LightBlue® Explorer 4+

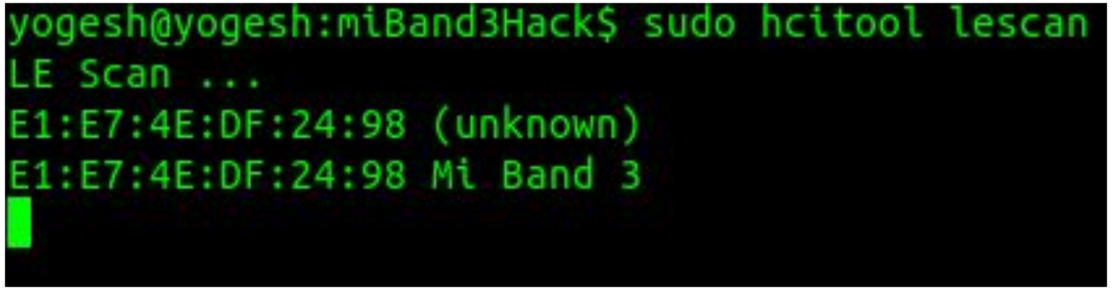
The go-to BLE development tool

Punch Through

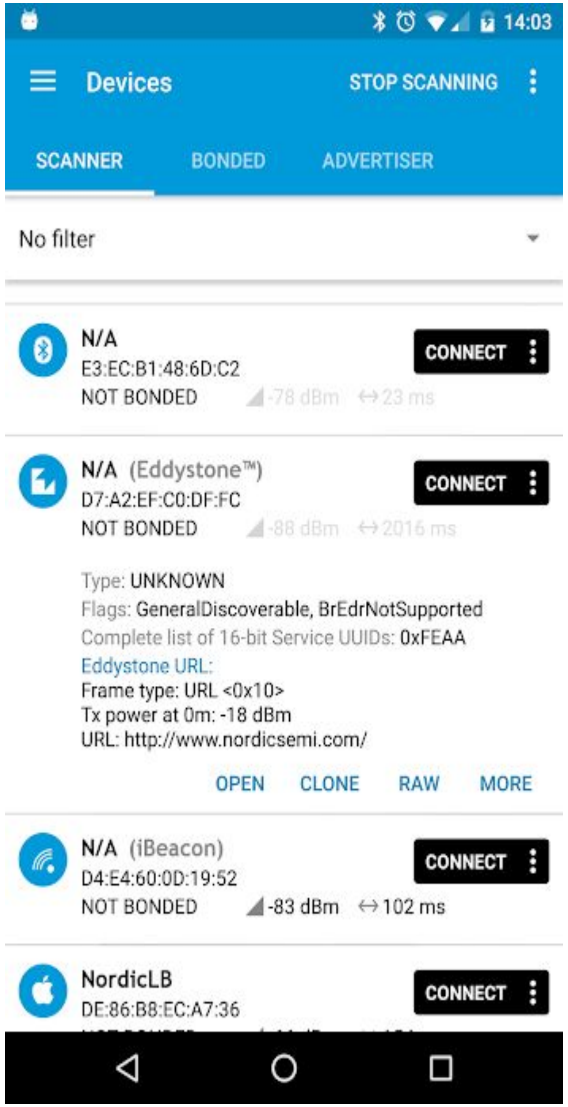
★★★★★ 4.3, 217 Ratings

Free

Scanning for BLE Devices



```
yogesh@yogesh:miBand3Hack$ sudo hcitool lescan
LE Scan ...
E1:E7:4E:DF:24:98 (unknown)
E1:E7:4E:DF:24:98 Mi Band 3
```



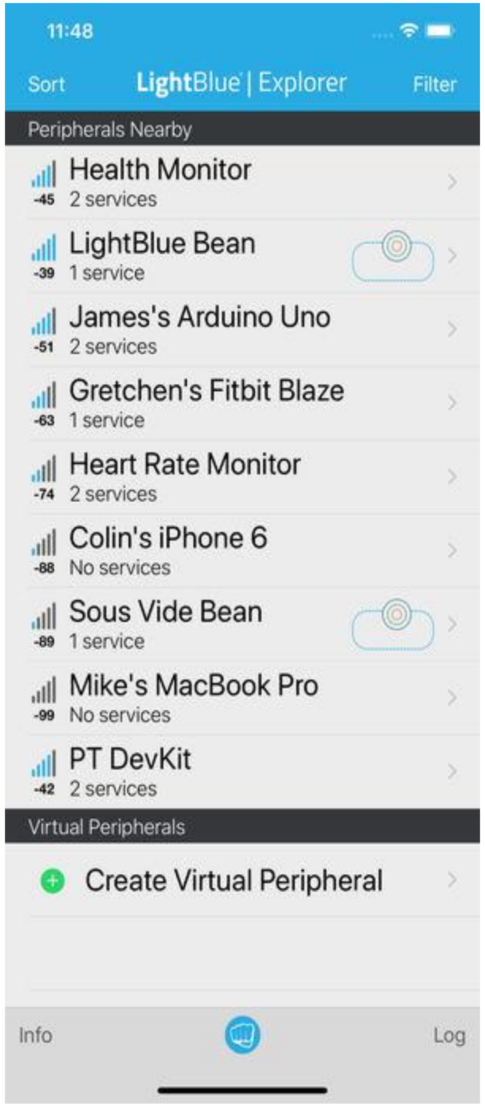
N/A
E3:EC:B1:48:6D:C2
NOT BONDED -78 dBm ↔23 ms
CONNECT

N/A (Eddystone™)
D7:A2:EF:C0:DF:FC
NOT BONDED -88 dBm ↔2016 ms
CONNECT

Type: UNKNOWN
Flags: GeneralDiscoverable, BrEdrNotSupported
Complete list of 16-bit Service UUIDs: 0xFEAA
Eddystone URL:
Frame type: URL <0x10>
Tx power at 0m: -18 dBm
URL: http://www.nordicsemi.com/

N/A (iBeacon)
D4:E4:60:0D:19:52
NOT BONDED -83 dBm ↔102 ms
CONNECT

NordicLB
DE:86:B8:EC:A7:36
CONNECT



Peripherals Nearby

-45	Health Monitor 2 services
-39	LightBlue Bean 1 service
-51	James's Arduino Uno 2 services
-63	Gretchen's Fitbit Blaze 1 service
-74	Heart Rate Monitor 2 services
-88	Colin's iPhone 6 No services
-89	Sous Vide Bean 1 service
-99	Mike's MacBook Pro No services
-42	PT DevKit 2 services

Virtual Peripherals

Create Virtual Peripheral

1. Enumerate the services and characteristics

```
sudo gatttool -b <BLE ADDRESS> -I

      >connect
```

List down all primary services

```
      > primary
```

List down all characteristics

```
      > characteristics
```

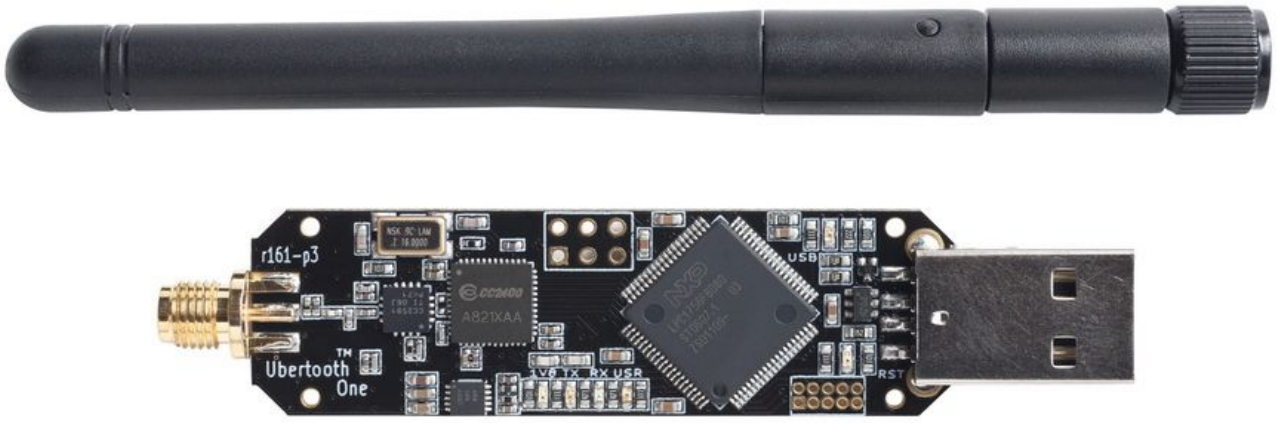
Sniffing BLE Packets

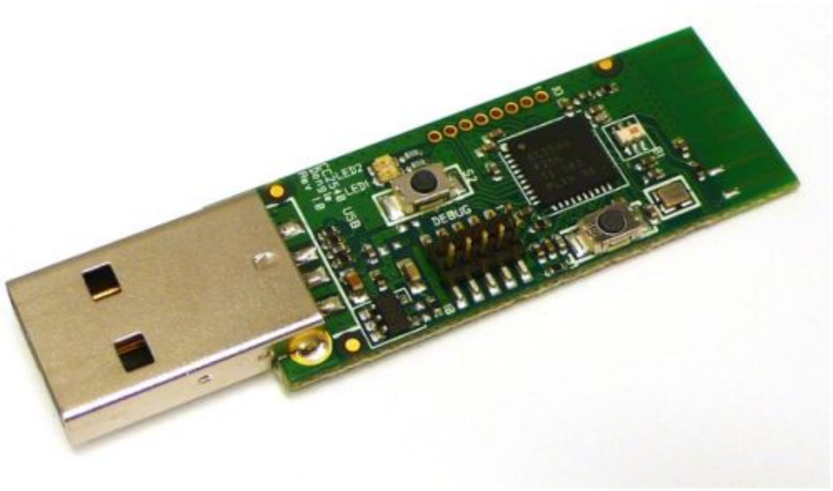
Ubertooth

- Works great for both Classic and BLE
- Open Source Hardware/Software
- About $100

CC2540

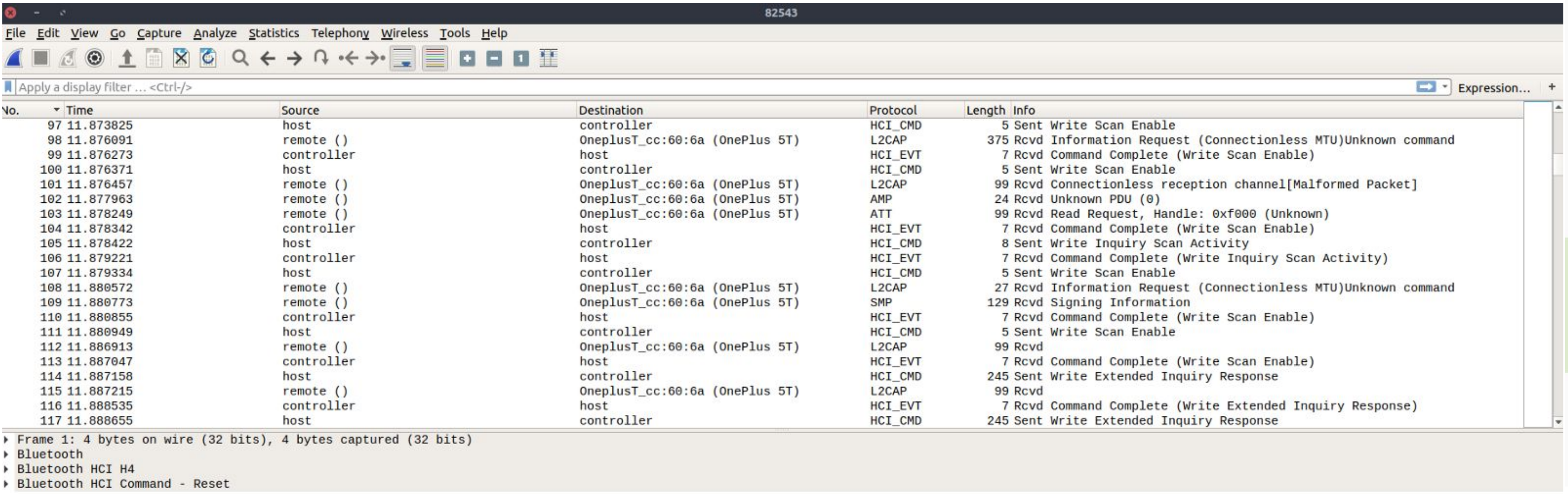
- Cheaper but limited configuration
- About $50





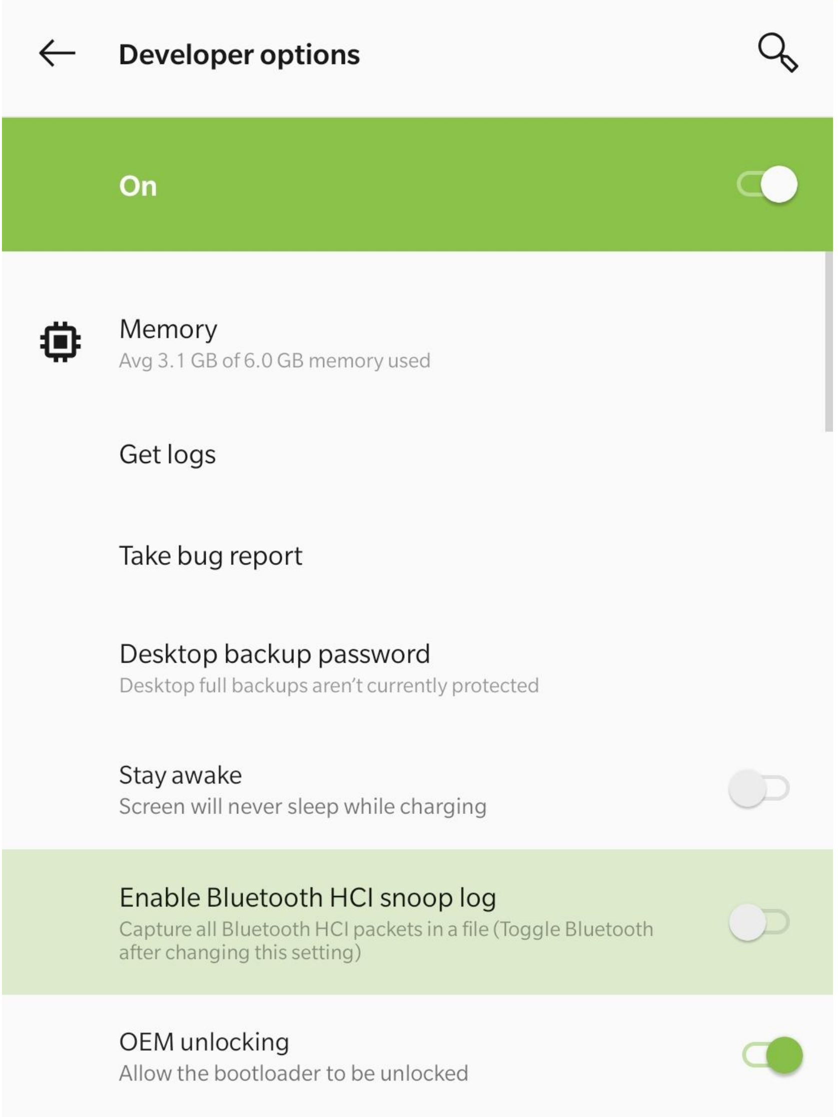
Alternate to Sniffers

- Enable Developer Option
- Enable Bluetooth HCI Snoop Log
- $ adb pull /sdcard/btsnoop_hci.log


82543
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...
+

```
No.  Time       Source       Destination                   Protocol  Length Info
 97 11.873825  host         controller                    HCI_CMD        5 Sent Write Scan Enable
 98 11.876091  remote ()    OneplusT_cc:60:6a (OnePlus 5T) L2CAP        375 Rcvd Information Request (Connectionless MTU)Unknown command
 99 11.876273  controller   host                          HCI_EVT        7 Rcvd Command Complete (Write Scan Enable)
100 11.876371  host         controller                    HCI_CMD        5 Sent Write Scan Enable
101 11.876457  remote ()    OneplusT_cc:60:6a (OnePlus 5T) L2CAP         99 Rcvd Connectionless reception channel[Malformed Packet]
102 11.877963  remote ()    OneplusT_cc:60:6a (OnePlus 5T) AMP           24 Rcvd Unknown PDU (0)
103 11.878249  remote ()    OneplusT_cc:60:6a (OnePlus 5T) ATT           99 Rcvd Read Request, Handle: 0xf000 (Unknown)
104 11.878342  controller   host                          HCI_EVT        7 Rcvd Command Complete (Write Scan Enable)
105 11.878422  host         controller                    HCI_CMD        8 Sent Write Inquiry Scan Activity
106 11.879221  controller   host                          HCI_EVT        7 Rcvd Command Complete (Write Inquiry Scan Activity)
107 11.879334  host         controller                    HCI_CMD        5 Sent Write Scan Enable
108 11.880572  remote ()    OneplusT_cc:60:6a (OnePlus 5T) L2CAP         27 Rcvd Information Request (Connectionless MTU)Unknown command
109 11.880773  remote ()    OneplusT_cc:60:6a (OnePlus 5T) SMP          129 Rcvd Signing Information
110 11.880855  controller   host                          HCI_EVT        7 Rcvd Command Complete (Write Scan Enable)
111 11.880949  host         controller                    HCI_CMD        5 Sent Write Scan Enable
112 11.886913  remote ()    OneplusT_cc:60:6a (OnePlus 5T) L2CAP         99 Rcvd
113 11.887047  controller   host                          HCI_EVT        7 Rcvd Command Complete (Write Scan Enable)
114 11.887158  host         controller                    HCI_CMD      245 Sent Write Extended Inquiry Response
115 11.887215  remote ()    OneplusT_cc:60:6a (OnePlus 5T) L2CAP         99 Rcvd
116 11.888535  controller   host                          HCI_EVT        7 Rcvd Command Complete (Write Extended Inquiry Response)
117 11.888655  host         controller                    HCI_CMD      245 Sent Write Extended Inquiry Response

▸ Frame 1: 4 bytes on wire (32 bits), 4 bytes captured (32 bits)
▸ Bluetooth
▸ Bluetooth HCI H4
▸ Bluetooth HCI Command - Reset
```



← Developer options

On ●

Memory
Avg 3.1 GB of 6.0 GB memory used

Get logs

Take bug report

Desktop backup password
Desktop full backups aren't currently protected

Stay awake ○
Screen will never sleep while charging

Enable Bluetooth HCI snoop log ○
Capture all Bluetooth HCI packets in a file (Toggle Bluetooth after changing this setting)

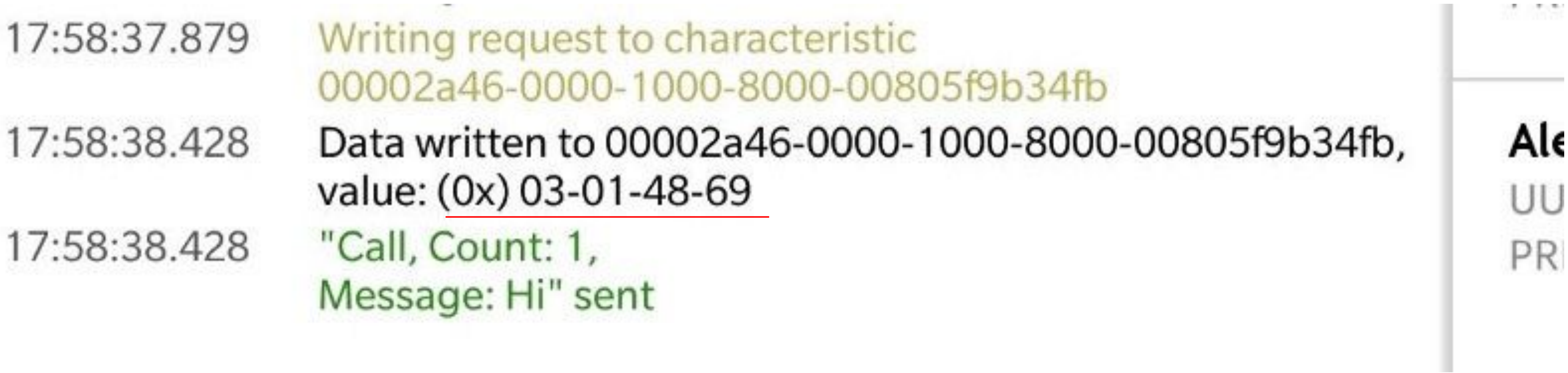
OEM unlocking ●
Allow the bootloader to be unlocked

Authentication in BLE devices

3 devices, out of 5 devices that I tested, did not implement link layer encryption.

2 devices, out of 5 devices that I tested, did not have authentication!!!

Send some Notification? ;)



```
17:58:37.879    Writing request to characteristic
                00002a46-0000-1000-8000-00805f9b34fb
17:58:38.428    Data written to 00002a46-0000-1000-8000-00805f9b34fb,
                value: (0x) 03-01-48-69
17:58:38.428    "Call, Count: 1,
                Message: Hi" sent
```

First Two Byte is Notification Type

01 -> Email

03 -> Call

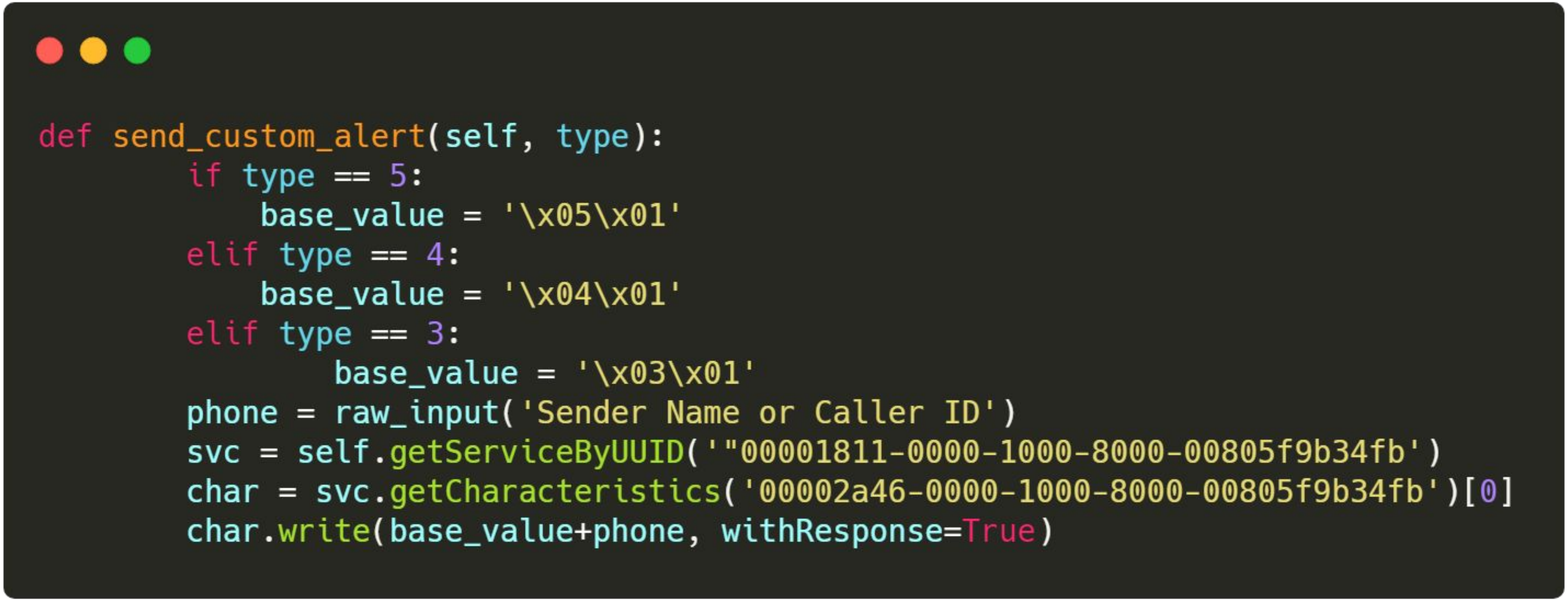
04 -> Missed Call

05 -> SMS/MMS

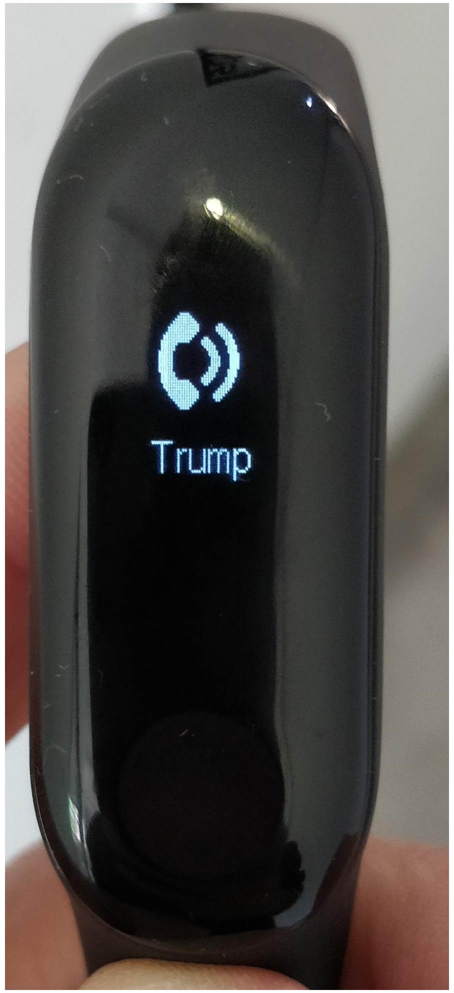
Next Two Byte is numbers of notification

And remaining is the hex value of the notification title that you are sending.

Send some Notification? ;)



```
def send_custom_alert(self, type):
        if type == 5:
            base_value = '\x05\x01'
        elif type == 4:
            base_value = '\x04\x01'
        elif type == 3:
                base_value = '\x03\x01'
        phone = raw_input('Sender Name or Caller ID')
        svc = self.getServiceByUUID('"00001811-0000-1000-8000-00805f9b34fb')
        char = svc.getCharacteristics('00002a46-0000-1000-8000-00805f9b34fb')[0]
        char.write(base_value+phone, withResponse=True)
```


Trump

Firmware

My aim was to display this!

Firmware

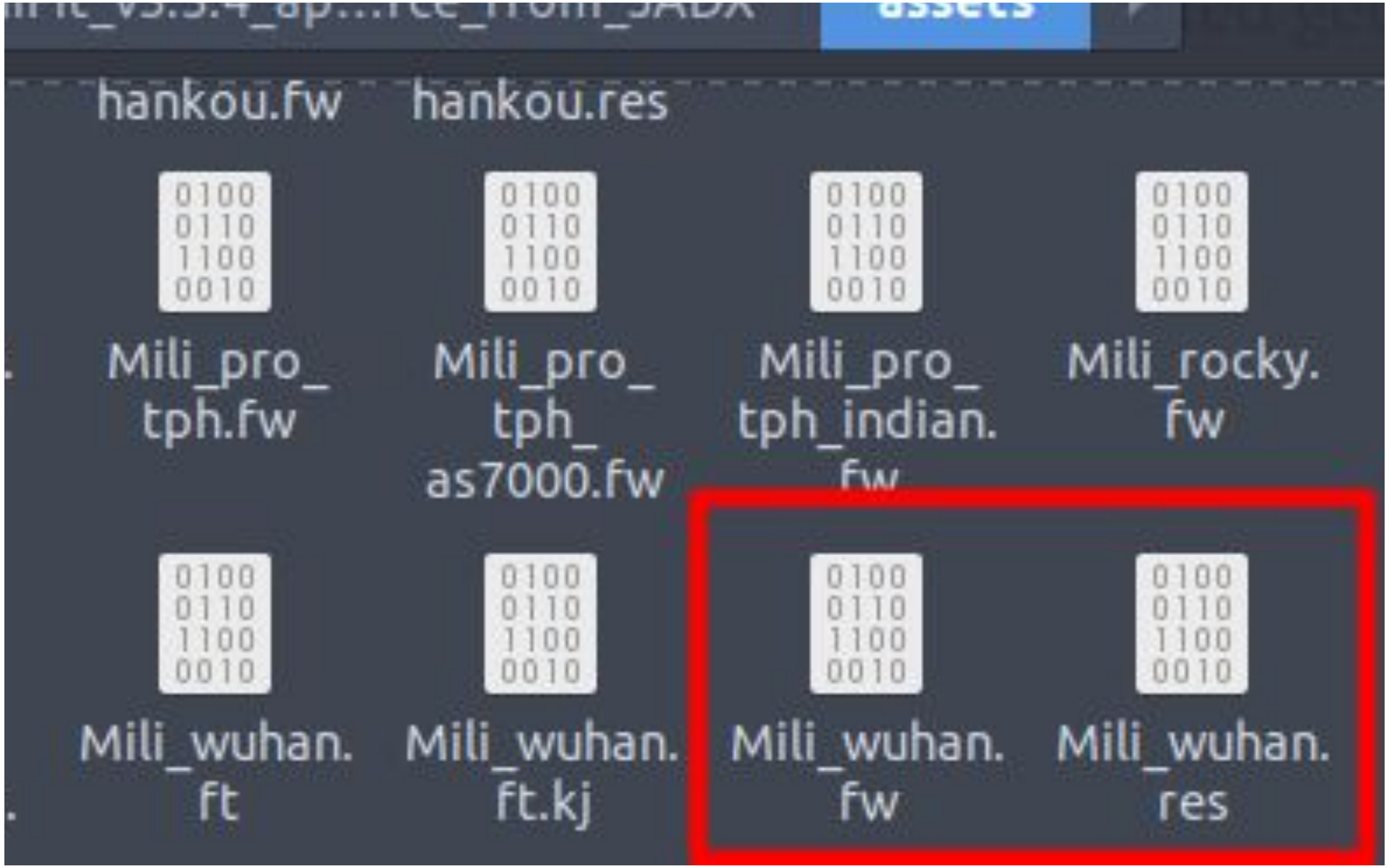
A **firmware** is a piece of Software that runs on embedded CPU!

How do I get firmware?

Reverse Engineering the Mobile application maybe? Or during the DFU update?

Let’s reverse engineer the mobile application!

```
$ apktool d cool_app.apk
```



hankou.fw hankou.res

Mili_pro_tph.fw

Mili_pro_tph_as7000.fw

Mili_pro_tph_indian.fw

Mili_rocky.fw

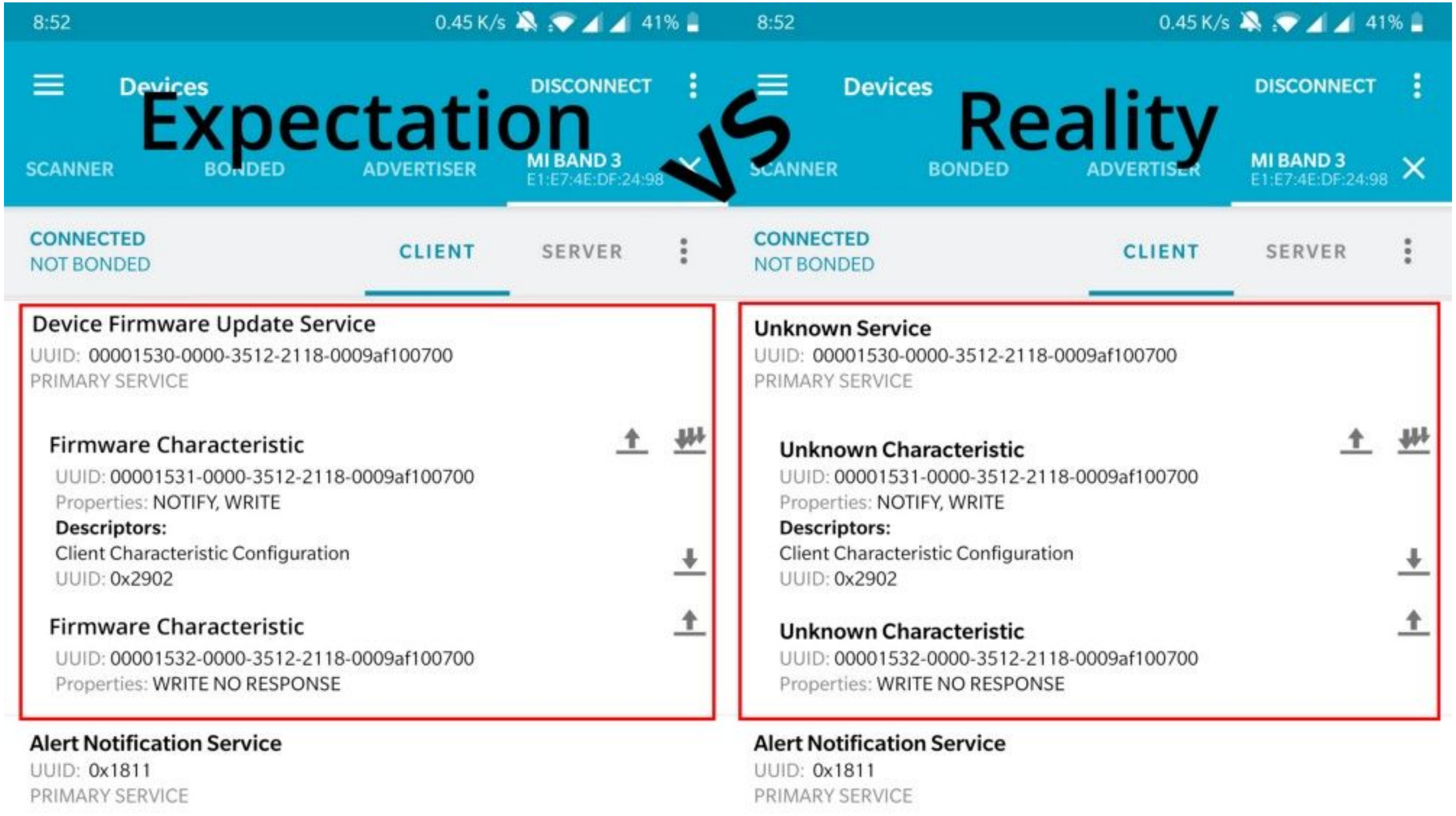
Mili_wuhan.ft

Mili_wuhan.ft.kj

Mili_wuhan.fw

Mili_wuhan.res

Uploading the firmware



Device Firmware Update Service
UUID: 00001530-0000-3512-2118-0009af100700
PRIMARY SERVICE

Firmware Characteristic
UUID: 00001531-0000-3512-2118-0009af100700
Properties: NOTIFY, WRITE
Descriptors:
Client Characteristic Configuration
UUID: 0x2902

Firmware Characteristic
UUID: 00001532-0000-3512-2118-0009af100700
Properties: WRITE NO RESPONSE

Alert Notification Service
UUID: 0x1811
PRIMARY SERVICE

Unknown Service
UUID: 00001530-0000-3512-2118-0009af100700
PRIMARY SERVICE

Unknown Characteristic
UUID: 00001531-0000-3512-2118-0009af100700
Properties: NOTIFY, WRITE
Descriptors:
Client Characteristic Configuration
UUID: 0x2902

Unknown Characteristic
UUID: 00001532-0000-3512-2118-0009af100700
Properties: WRITE NO RESPONSE

Alert Notification Service
UUID: 0x1811
PRIMARY SERVICE

How does firmware upload works?
For this fitness tracker

- Initialize the firmware/resource Update On Characteristic 1531 with write command of 4-byte
- **\x01** + fileSize in Hex(3-byte)
- But, for the resource, its **5-byte**!
  \x01 + **fileSize** in Hex(3-byte) + **\x02**
- Last byte **\x02** is for letting the firmware update service know that it's a resource and not the firmware file.

Doesn’t accept 0x5EFAC but accepts 0xAcEF05

How does firmware upload works? For this fitness tracker

What is **Checksum?**

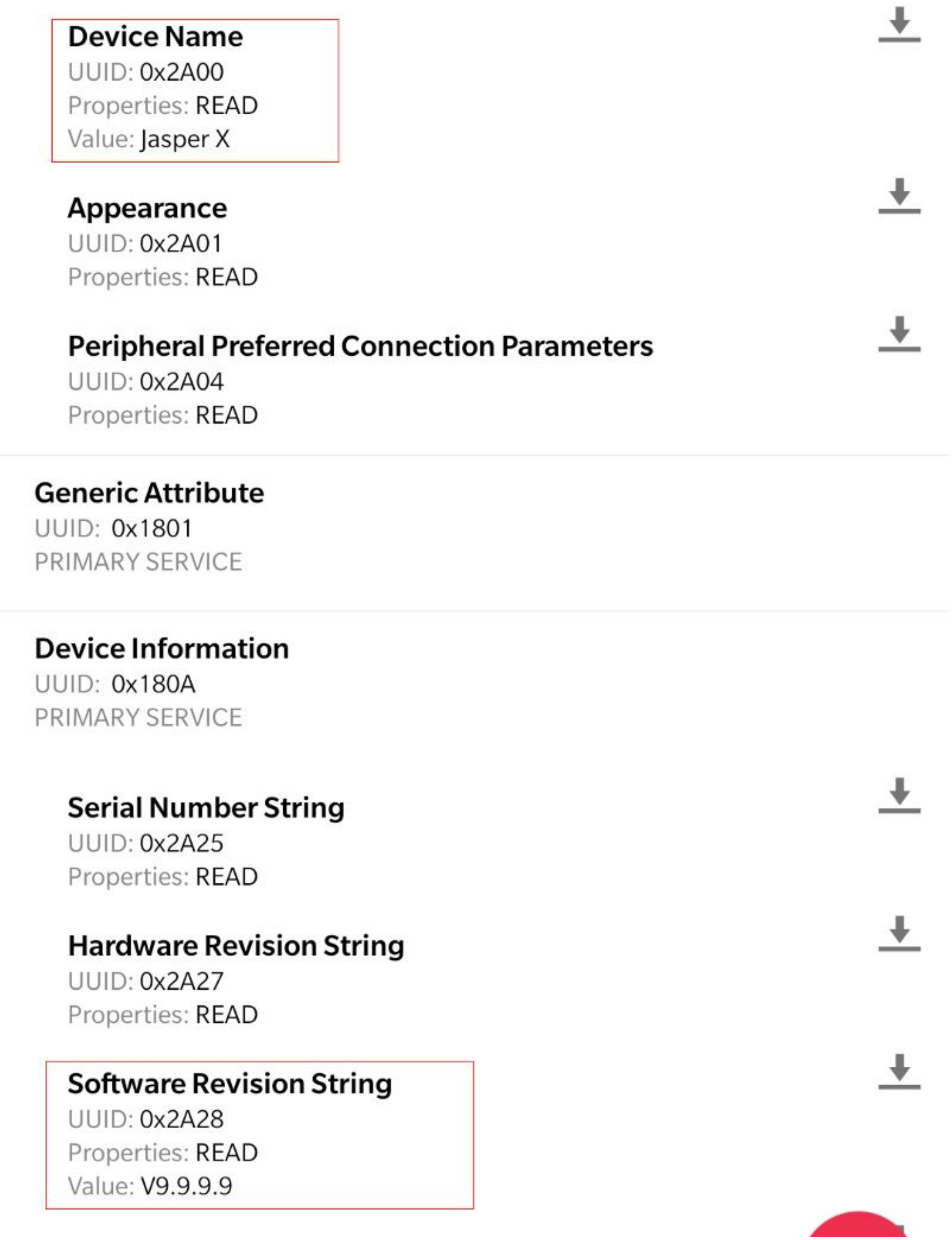
Calculated value that is used to determine the integrity of data during the transmission.

BLE does not perform error correction but can only perform error detection. Bluetooth 5.0 introduces error correction.

How does firmware upload works?
For this fitness tracker

Once the CRC is calculated, write the checksum to Characteristic “XXXX” of 3 bytes.
The checksum must begin with \x04 and your checksum value
\x04 + checksum
If the checksum matches the resource will be accepted and updated. But for firmware, you need to send reboot command as well.
On Characteristic “XXXX” send \x05 for the reboot.

And yes, the firmware update is done!



Device Name
UUID: 0x2A00
Properties: READ
Value: Jasper X

Appearance
UUID: 0x2A01
Properties: READ

Peripheral Preferred Connection Parameters
UUID: 0x2A04
Properties: READ

Generic Attribute
UUID: 0x1801
PRIMARY SERVICE

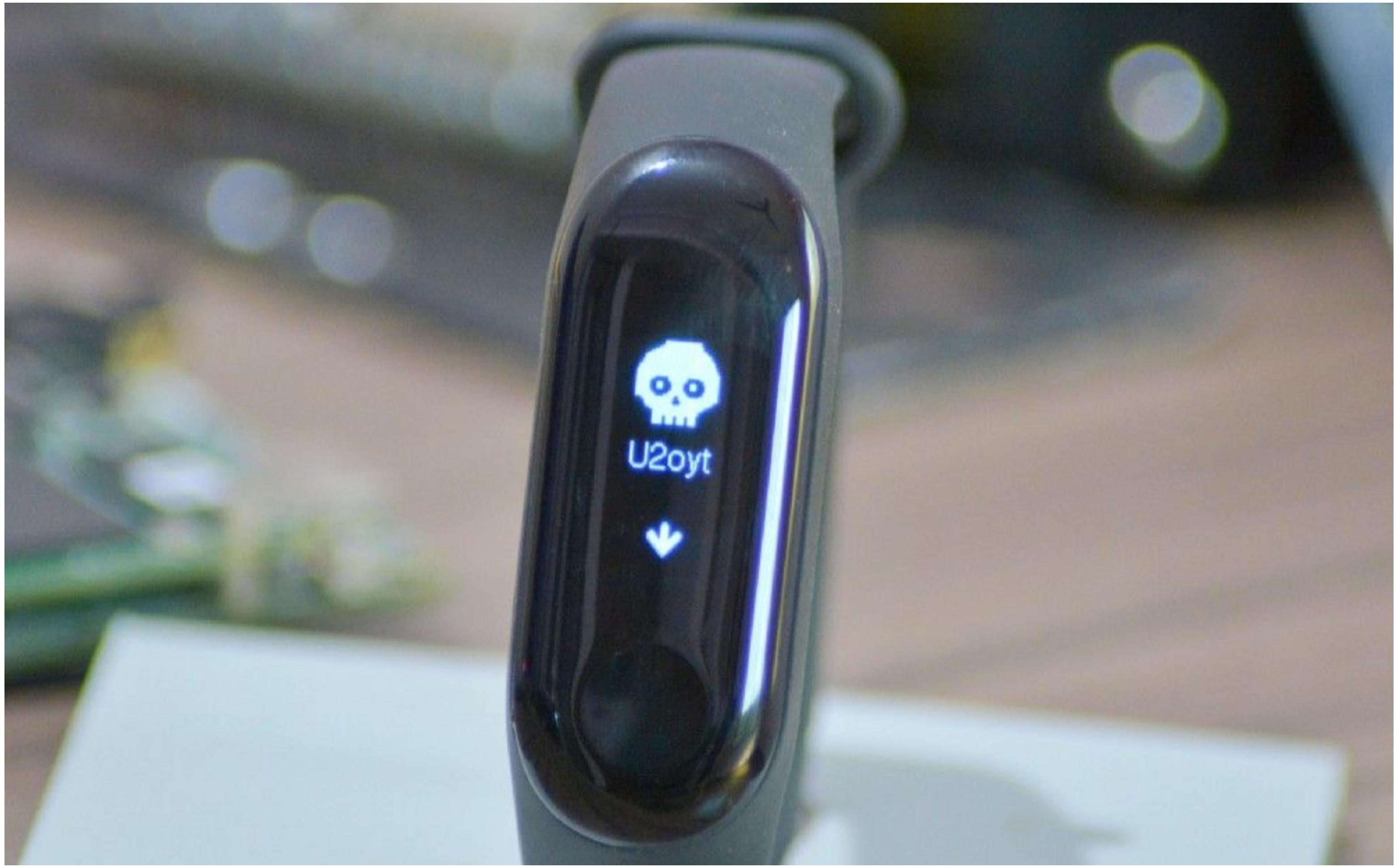
Device Information
UUID: 0x180A
PRIMARY SERVICE

Serial Number String
UUID: 0x2A25
Properties: READ

Hardware Revision String
UUID: 0x2A27
Properties: READ

Software Revision String
UUID: 0x2A28
Properties: READ
Value: V9.9.9.9

What about the skull Icon? ;)

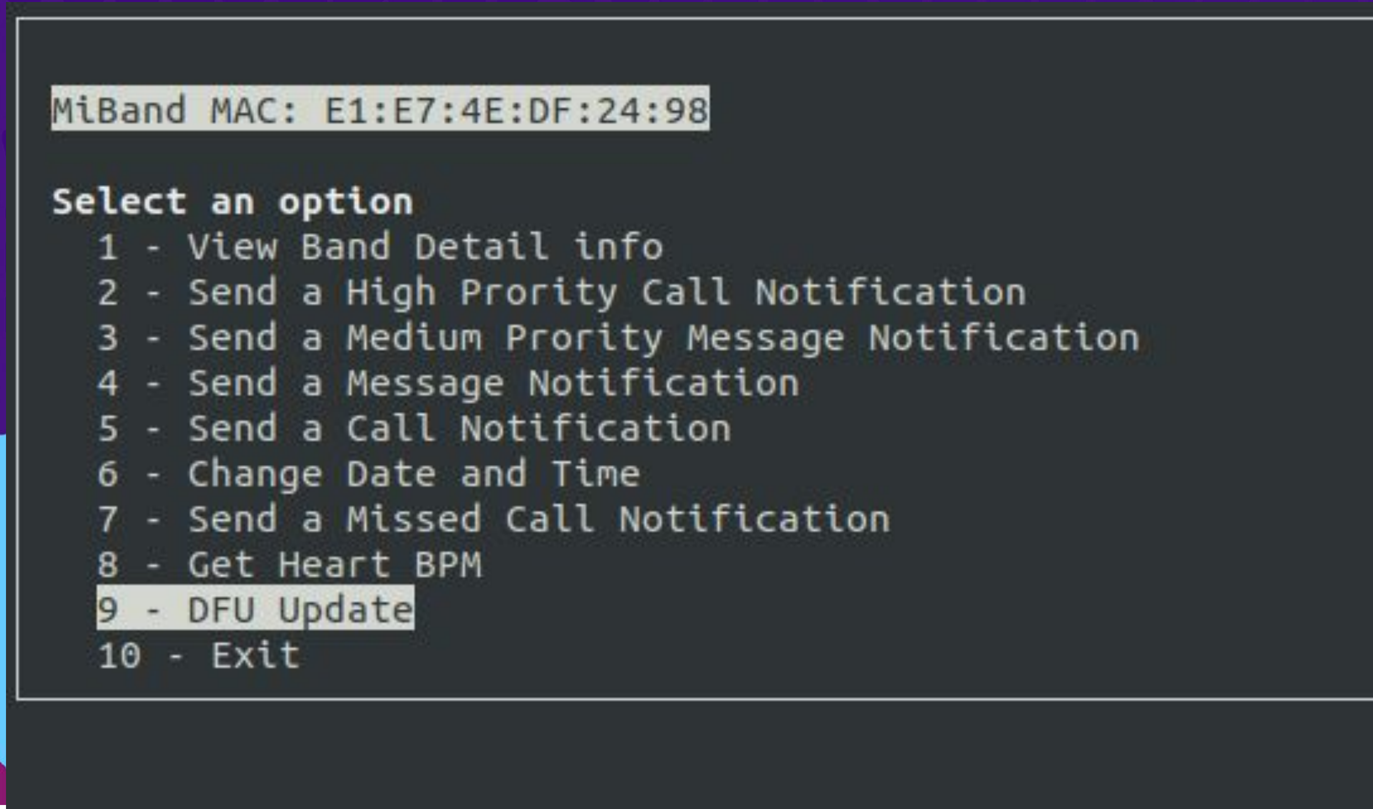


Q&A

More about this hack is on Medium & Github!

https://medium.com/@yogeshojha

https://github.com/yogeshojha/MiBand3/



```
MiBand MAC: E1:E7:4E:DF:24:98

Select an option
  1 - View Band Detail info
  2 - Send a High Prority Call Notification
  3 - Send a Medium Prority Message Notification
  4 - Send a Message Notification
  5 - Send a Call Notification
  6 - Change Date and Time
  7 - Send a Missed Call Notification
  8 - Get Heart BPM
  9 - DFU Update
  10 - Exit
```