Bluetooth, Suffering, and You
Part II: Developing Painful Experience

Paul A. Wortman, PhD
Mauddib28

January 21, 2025

Table of contents

Whoami

- PhD
- Bluetooth Security Researcher
- Research Scientist

Whatisthis?

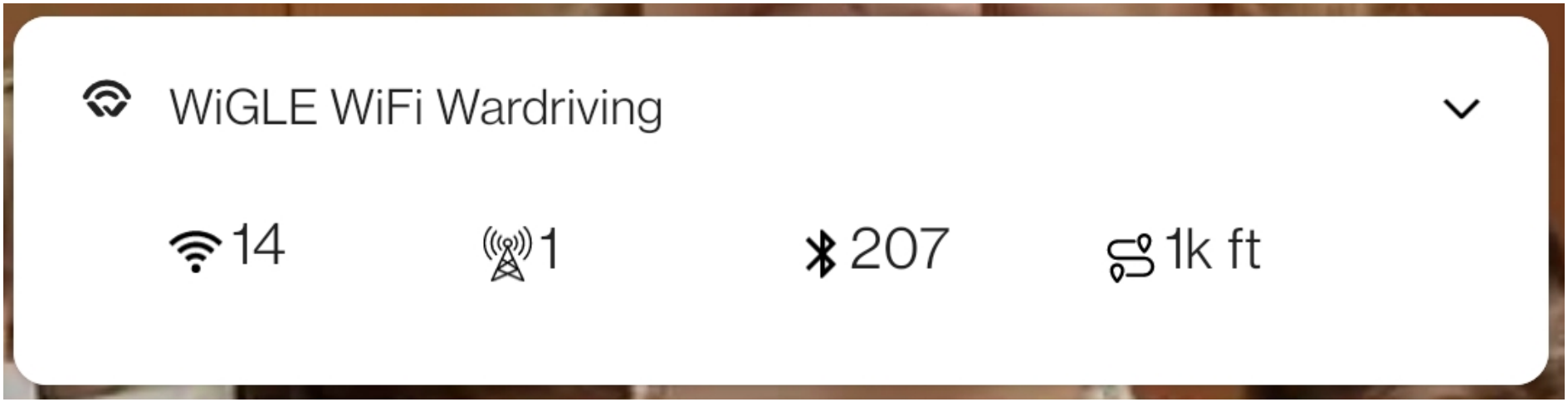


Figure 1: Wiggle War Drive - Landing Strip

“Lightning” Headphones That Require Bluetooth

Josh Whiton:

> A crazy experience — I lost my earbuds in a remote town in Chile, so tried buying a new pair at the airport before flying out. But the new wired, iPhone, lightning-cable headphones didn’t work. Strange.

Why is Bluetooth Important?

- Ubiquity of devices in the wild
- What is Bluetooth purposed for?
  - Replace Wires... Wirelessly...
- Embedded Systems? Internet of Things!
  - Ex: Medical, Home, Space, Towns, Roads, Sensors, etm.

BLE - Must Know Basics - High-level BLE Structure

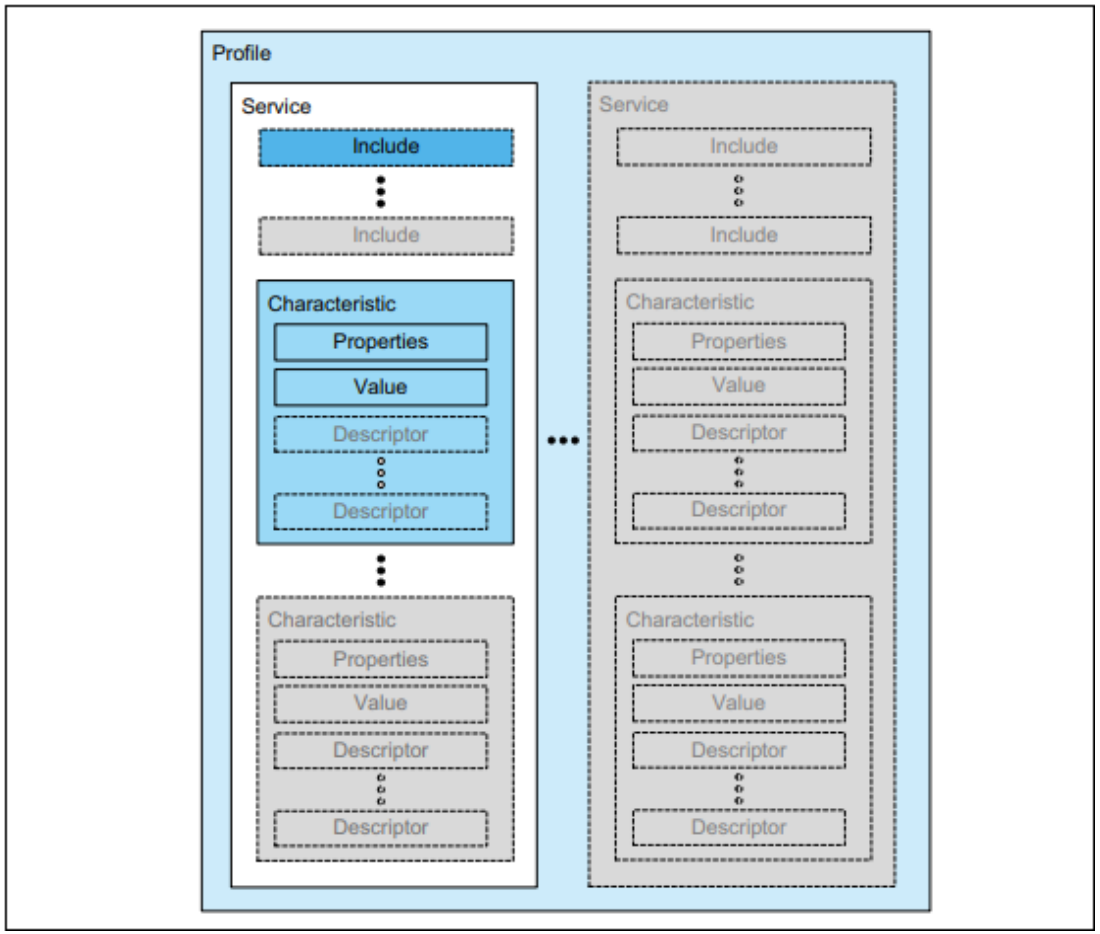
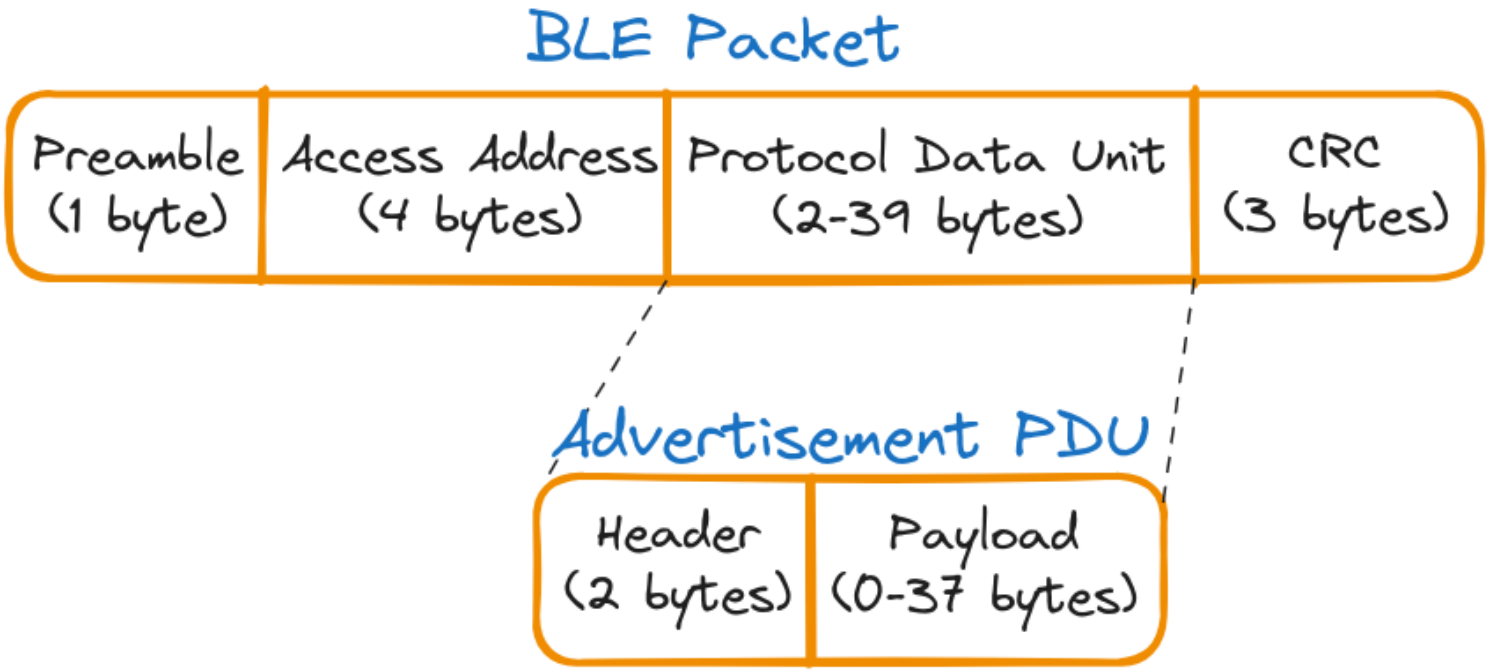


Figure 3: GATT-Based Profile hierarchy [14]

- Device, Services, Characteristics, Descriptors
- Characteristics have to be read once to populate data
  - Second read allows determining the data

BLE - Must Know Basics - Time to Advertise



Purpose of Device's Advertisement Packet

- Make self known to nearby devices
- Pass device address DE:AD:BE:EF:00 for identification
- Additional Services information

Listing Libraries

What libraries are out there you say?

- btzen
- μpython bluetooth / aioble
- bleak
- pybluez
- simplepyble
- NoBLE
- BLEno
- PyBT
- PyGATT

Note: A generally found reference is ukBaz Bluetooth Writings

What Code Libraries Exist

- Lots.... Bleak, BlueZ, SimpleBLE, microPython bluetooth, micropython aioble, pybluez, btzen
- Settled on BlueZ library
  - Everyone has issues.... Lack of examples, code, implementation...
  - Due to lessons learned..... Smallest scale granularity = Winner!

Where to Start?

The Bluetooth State Machine

Basic of Bluetooth (found via blood, sweat, and lots of tears)

- How does it work?
- The state machine
- Issues that get it

BLE SM - Constraints

Embedded Bluetooth devices essentially run under the following constraints:

- BLE device acts as a single processor device
- Multiplexing is the “workaround” for parallelism / threading
- Precognition of what is being searched for

BLE SM - Specifics

Method of Examining BLE Devices as a State Machine:

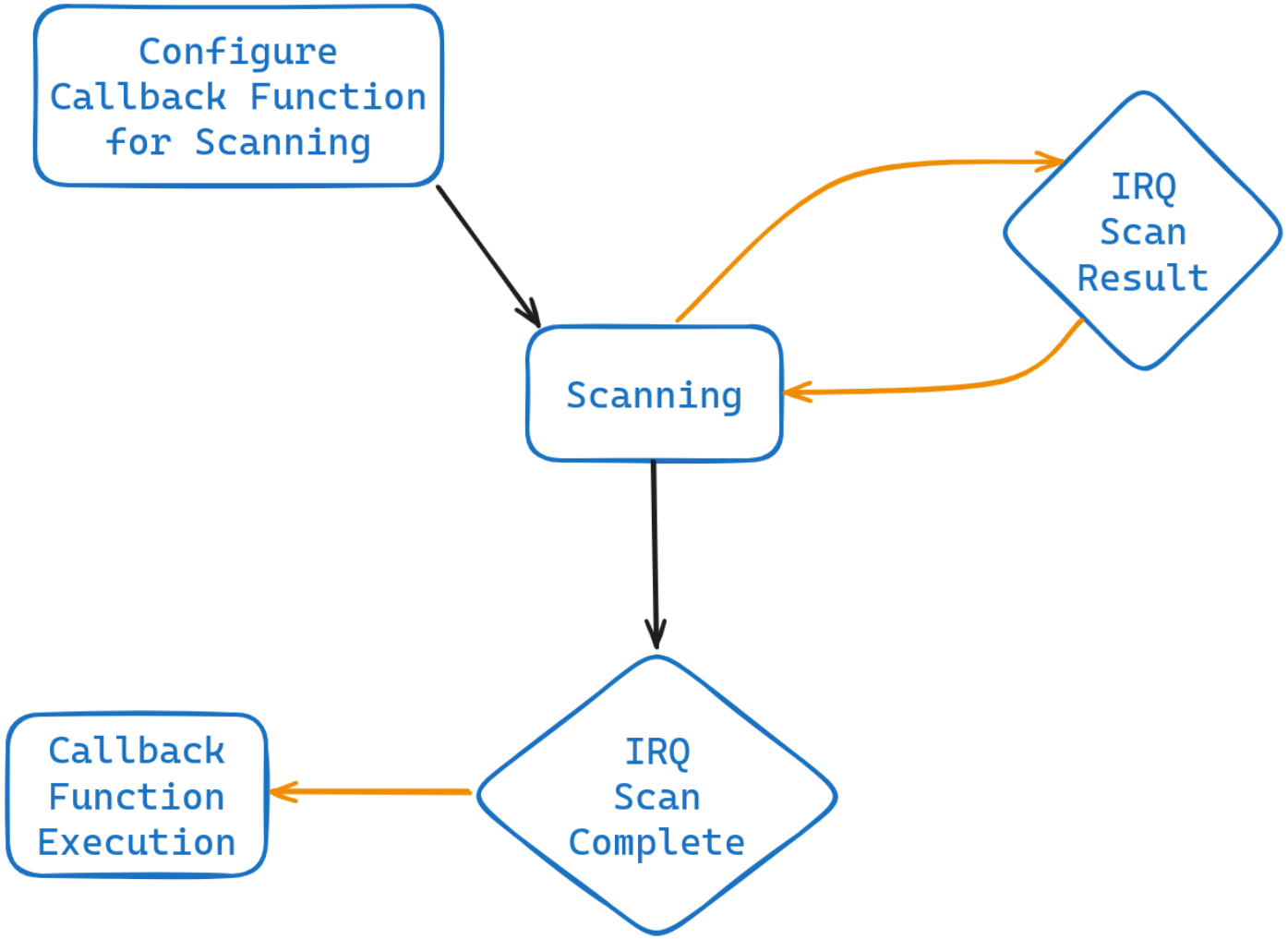
- Tasks must be performed in a specific order
- Too much nesting leads to stalling/loss
- Memory access is a tricky process....
  - May require creating copies of data
  - Leads to potential resource limits / hanging

Note: There is a notable difference between a *Central* and *Peripheral* device

- E.g. GATTC, GATTS
- Negligible for higher level review

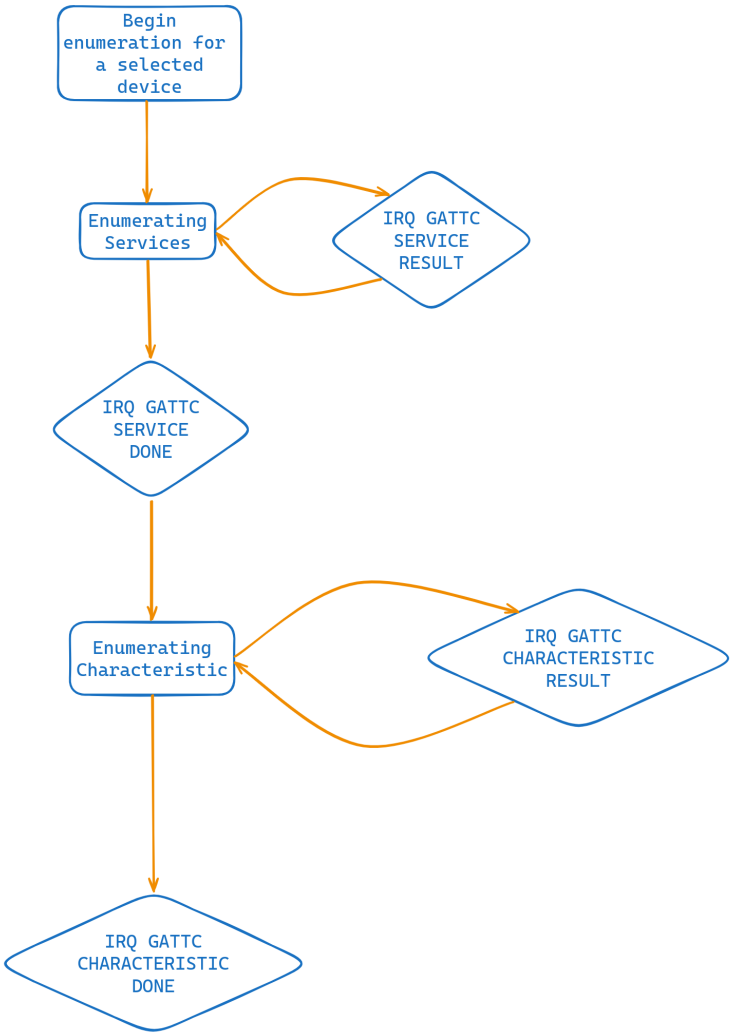
BLE SM - Scanning

Scanning State Machine:



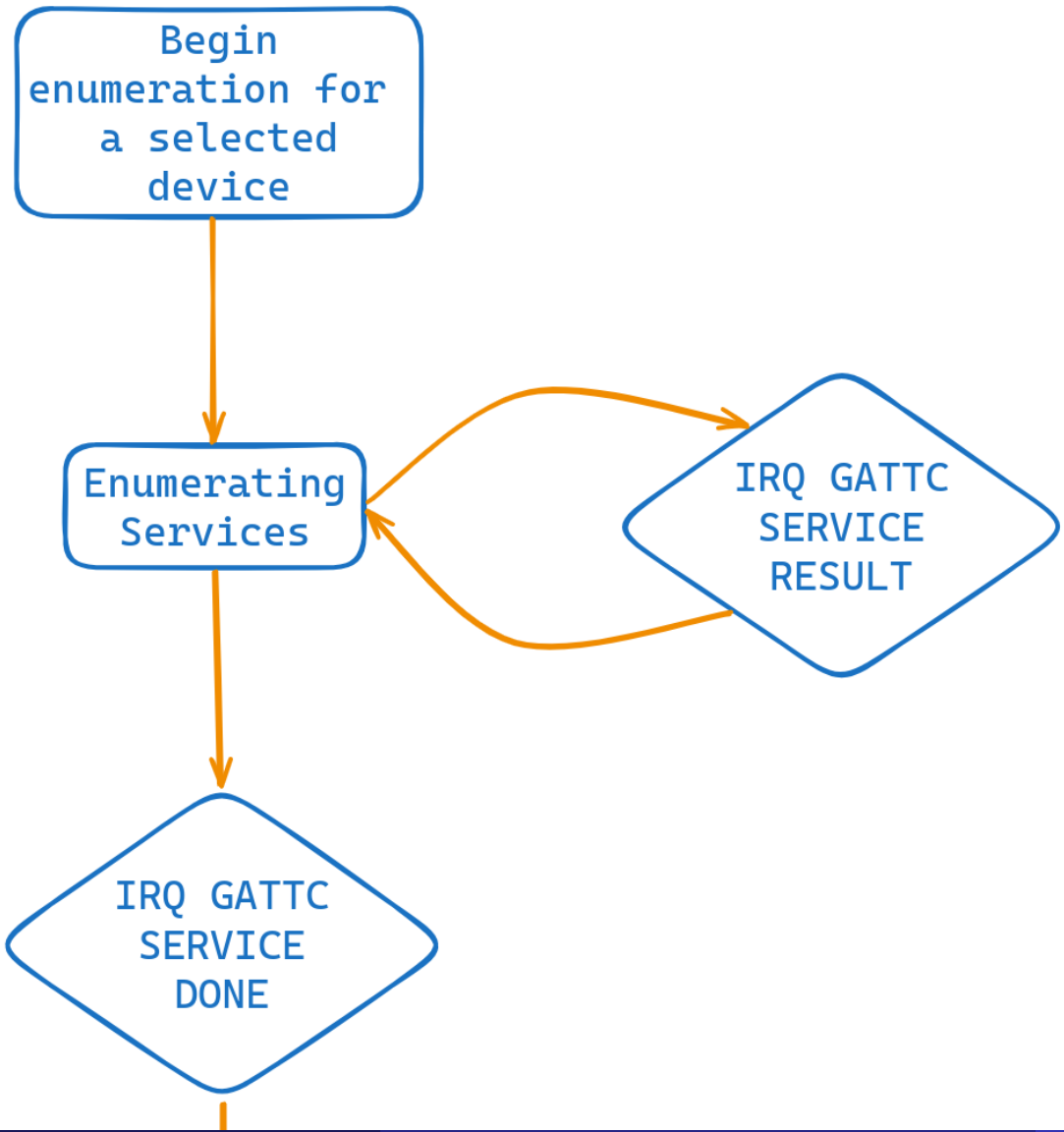
BLE SM - Enumeration

Enumeration State Machine:



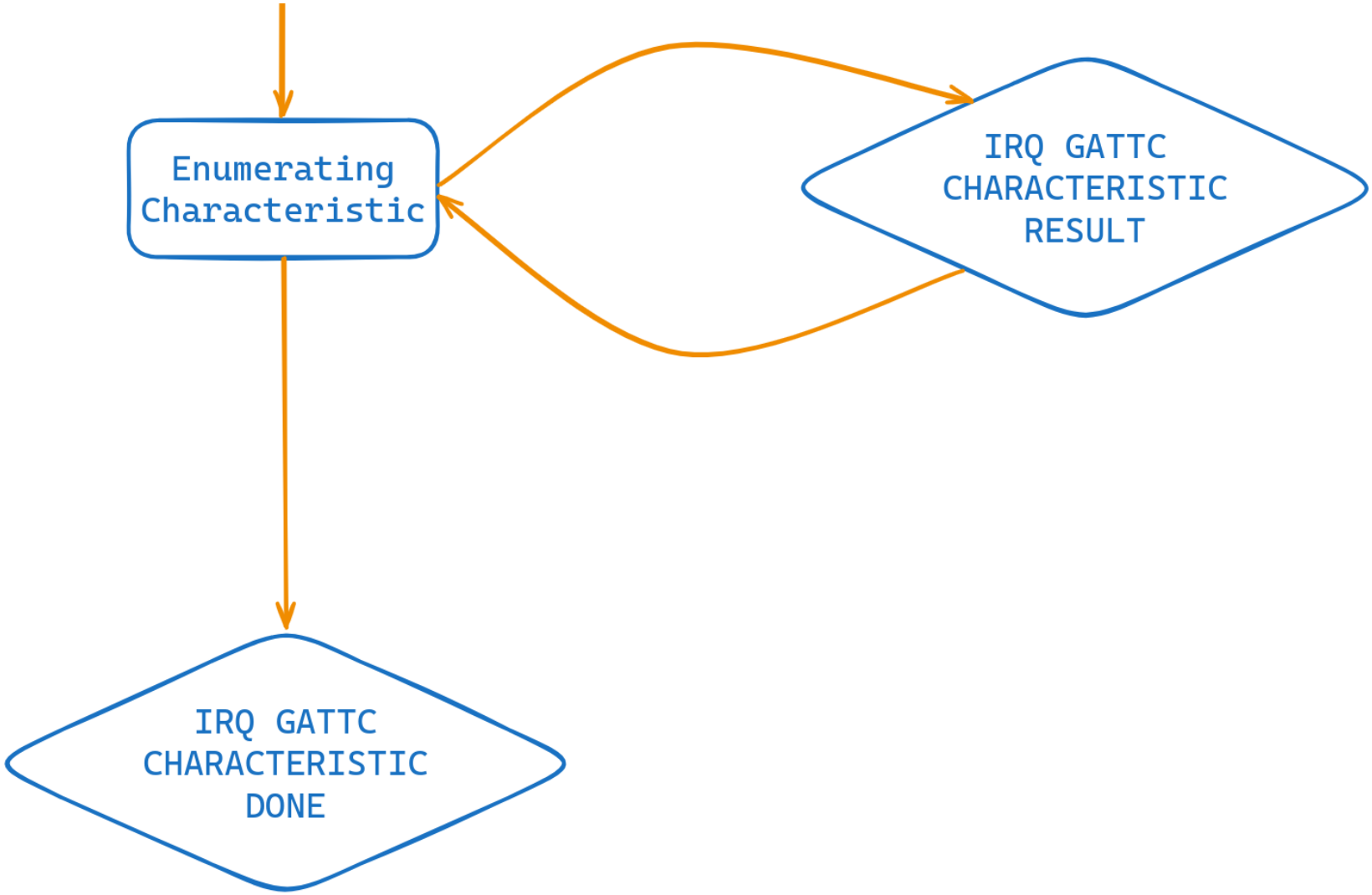
BLE SM - Enumeration - Services

Enumeration of Services:



BLE SM - Enumeration - Characteristics

Enumeration of Characteristics:



Handles Handles Everywhere

Handles vs. UUIDs vs. Short UUIDs Multiple ways of referencing the same Service/Characteristic/Descriptor:

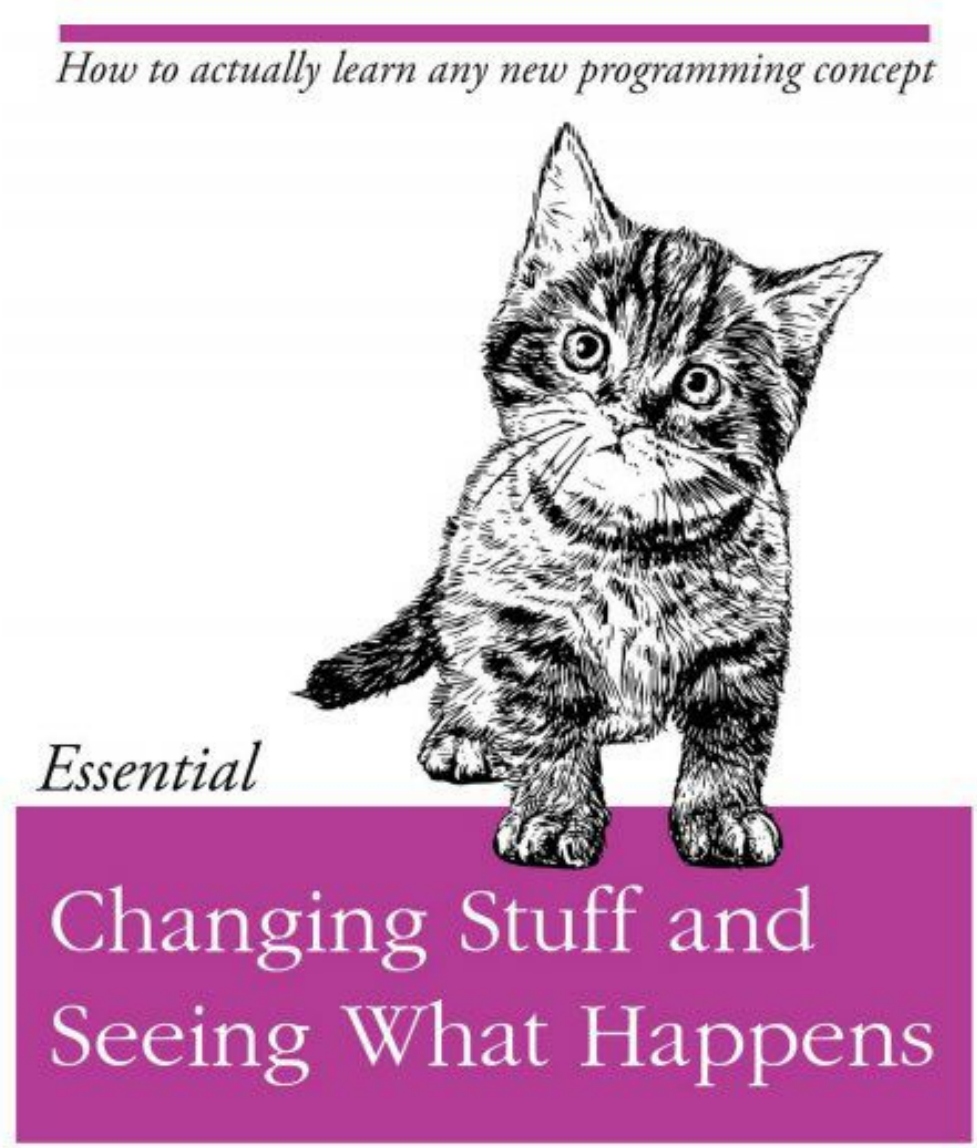
- UUID
  - Short (0x5678)
  - Long (12345678-0000-1111-2222-4444-55555)
- Handles
  - Start, End, Value
  - Note: Value is within the Start - End range
- Handles are a representation of the UUIDs; although Handles can appear differently between CLI tools and D-Bus structure returns

Moar Restrictions?!?!

Larger Limitations:

- Only a small space of UUIDs is standardized
  - Specific format for Bluetooth SIG (`XXXXXXXX-0000-1000-8000-00805f9b34fb`)
  - Some “provided” as vendor specific
  - Wild West
- ABSTRACTIONS!?!?!!!
  - Always abstractions of something lower
  - Eventually hit H-File bedrock....
  - Keeping the secrets

How To Develop



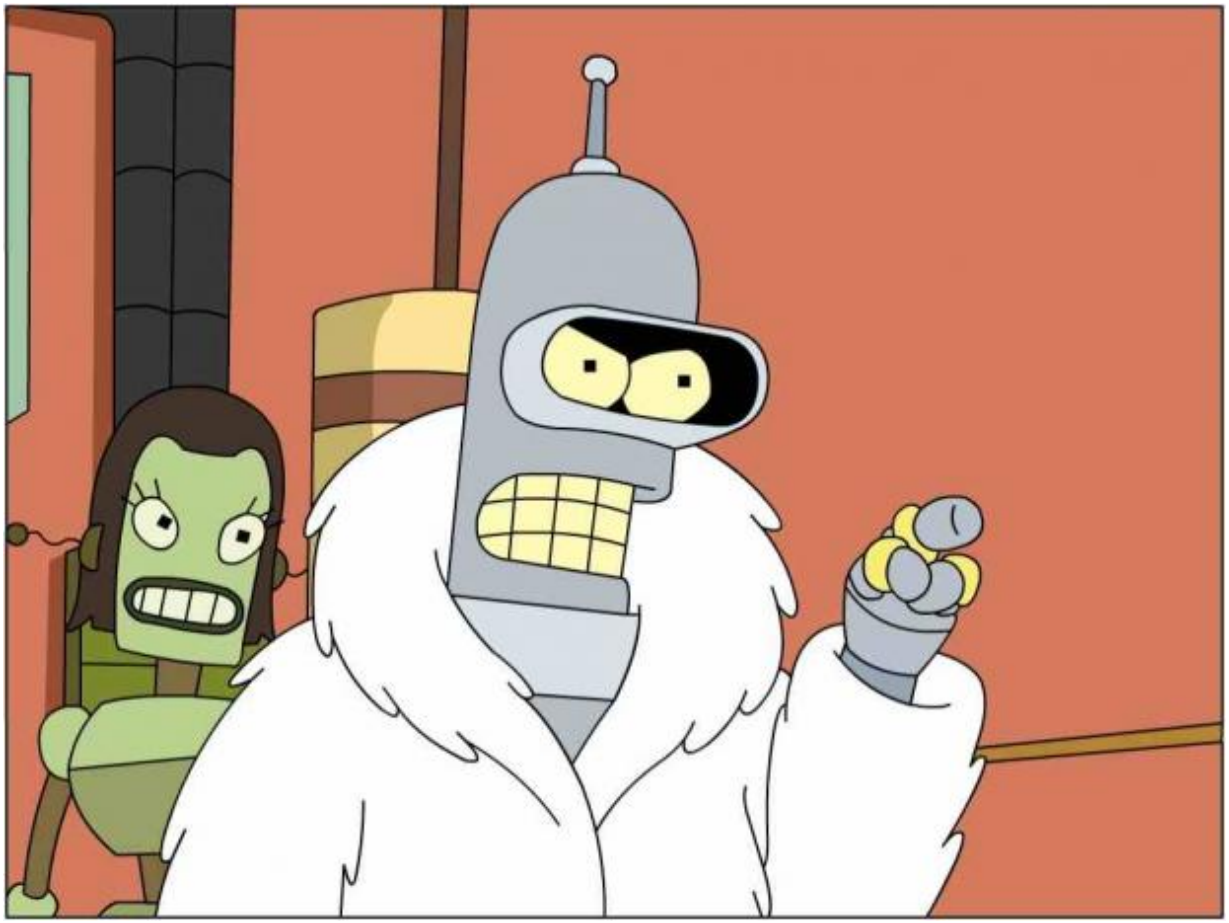
Basics with Arduino

Start with something “simple”

- How to get working:
  - Connection
  - Reads / Writes
  - Build GATT Structure
- Why do things break?
  - Not understanding the underlying technology.... RTFM....
  - Arduino just breaks when writing to Descriptors....????

BLE C[w]TF to Pico W - Signals Suck....

- Not seeing BLE CTF signals!?!?!!!
- Make a newer, better BLE server!



Figuring out a Pico W

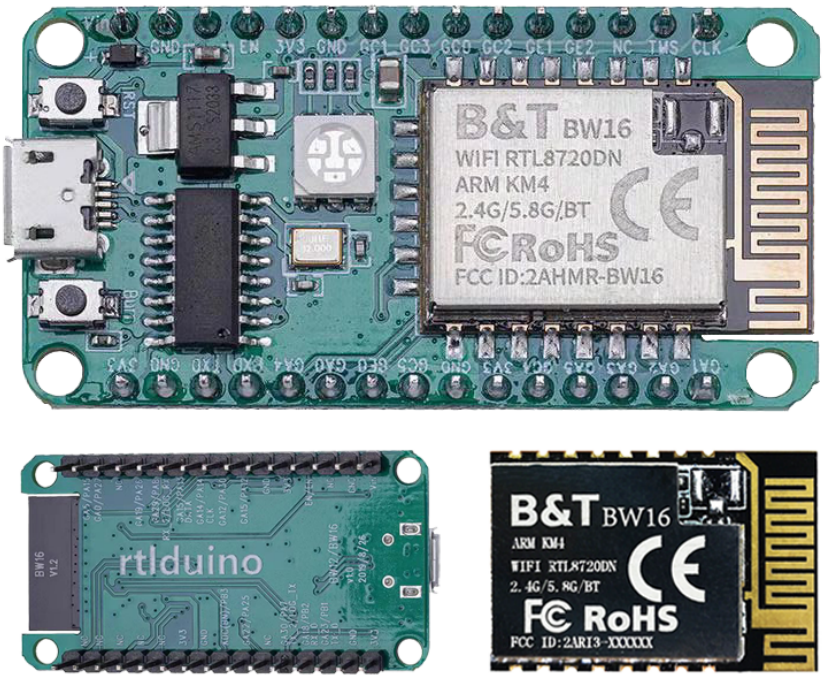
Most difficult was determining the order of operations for the embedded system

- Performing too many nested calls cause a stall
- Interacting with “ephemeral” information can cause a stall/crash
- Some information can only be accessible when the main loops are “paused”

Getting the BW-16 Working

Requirements, Costs, and Limitations:

- Planning of components, wires, breadboard vs. soldering, power
- Approximately `$9.00` computer, limited memory/storage
- Where to find documentation; getting the basics started



Resources Technical Fidelity Manuals

Always have a good reference to build the foundation

- `https://www.amebaiot.com/en/ameba-arduino-summary/`
- `https://www.amebaiot.com/en/amebapro2-amb82-mini-arduino-getting-started/`

Pre-requisites

Great pre-requisite file found for handling advertisements:

- `https://github.com/micropython/micropython/blob/master/examples/bluetooth/ble_advertising.py`
- Provides low-level dissection and unpacking for decoding information
- Includes example `demo()` to illustrate functionality

Note: *Focus is on creating a Peripheral; separate from a Central*

Advertising Basics - Imports

First one needs to import the `advertising_payload` function from the aforementioned `ble_advertising.py` file

```
import bluetooth
from ble_advertising import advertising_payload

from micropython import const
```

Advertising Basics - Definitions Part I

```
# Definition of Internal Request Queries (???);
    Used to Signal Internal States
_IRQ_CENTRAL_CONNECT = const(1)
_IRQ_CENTRAL_DISCONNECT = const(2)
_IRQ_GATTS_WRITE = const(3)
```

Advertising Basics - Definitions Part II

```
# Definition of Constants used to Designate
    Specific Service/Characteristic/Descriptor
    Functional Flags
_FLAG_READ = const(0x0002)
_FLAG_WRITE_NO_RESPONSE = const(0x0004)
_FLAG_WRITE = const(0x0008)
_FLAG_NOTIFY = const(0x0010)
```

Advertising Basics - Definitions Part III

```
# Definition of the UUIDs and tuples for
    Establishing Services and Characteristics (e.g
    . TX and RX Characteristics for the UART
    Service)
_UART_UUID = bluetooth.UUID("6E400001-B5A3-F393-
    E0A9-E50E24DCCA9E")
_UART_TX = (
        bluetooth.UUID("6E400003-B5A3-F393-E0A9-
            E50E24DCCA9E"),
        _FLAG_READ | _FLAG_NOTIFY,
)
_UART_RX = (
        bluetooth.UUID("6E400002-B5A3-F393-E0A9-
            E50E24DCCA9E"),
        _FLAG_WRITE | _FLAG_WRITE_NO_RESPONSE,
)
```

Advertising Basics - Definitions Part IV

```
_UART_SERVICE = (
        _UART_UUID,
        (_UART_TX, _UART_RX),
)
```

Advertising Basics - Class Definition Part I

```
class BLESimplePeripheral:
        def __init__(self, ble, name="mpy-uart"):
                self._ble = ble
                self._ble.active(True)
                ...
                self._connections = set()
                ...
                self._payload =
                    advertising_payload(name=name,
                     services=[_UART_UUID])
                self._advertise()

        ...
```

Advertising Basics - Class Definition Part II

```
...

def send(self, data):
        for conn_handle in self.
            _connections:
                self._ble.gatts_notify(
                    conn_handle, self.
                    _handle_tx, data)

...

def _advertise(self, interval_us=500000):
        print("Starting␣advertising")
        self._ble.gap_advertise(
            interval_us, adv_data=self.
            _payload)
```

Advertising Basics - Main Function

```
def demo():
        led_onboard = Pin("LED", Pin.Out)
        ble = bluetooth.BLE()
        p = BLESimplePeripheral(ble)

        ...

if __name__ = "__main__":
        demo()
```

Advertising Basics

Now you too can Advertise!!

Connect / Disconnect

Next one needs the capability to connect/disconnect from any device

Connect / Disconnect - Imports

```
import time
```

Connect / Disconnect - Class Definition - Part I

```
class BLESimplePeripheral:
        def __init__(self, ble, name="mpy-uart"):
                ...
                self._ble.irq(self._irq)
                ...
                self._connections = set()
```

Connect / Disconnect - Class Definition - Part II

```
def _irq(self, event, data):
        if event == _IRQ_CENTRAL_CONNECT:
    conn_handle, _, _ = data
    print("New connection", conn_handle)
    self._connections.add(conn_handle)
elif event == _IRQ_CENTRAL_DISCONNECT:
    conn_handle, _, _ = data
    print("Disconnected", conn_handle)
    self._connections.remove(conn_handle)
    self._advertise()
...
```

Connect / Disconnect - Class Definition - Part III

```
...

def is_connected(self):
return len(self._connections) > 0

...
```

Connect / Disconnect - Main Function

```
def demo():
        led_onboard = Pin("LED", Pin.OUT)
        ...

        while True:
                if p._is_connected():
                        led_onboard.on()
                        ...
                time.sleep_ms(100)
```

Reading I/O

- Reading requires expansion of the Class Definition
- Additions to the main function

Reading I/O - Class Definition - Part I

```
class BLESimplePeripheral:
        def __init__(self, ble, name="mpy-uart"):
                ...
                self._ble.irq(self._irq)
                ((self._handle_tx, self.
                   _handle_rx),) = self._ble.
                   gatts_register_services((
                   _UART_SERVICE,)
                ...
                self._read_callback = None

        def read_from_attribute(self,
            attribute_handle):
        data = self._ble.gatts_read(
            attribute_handle)
        return data
```

Reading I/O - Class Definition - Part II

```
...

    def on_read(self, callback):
            self._read_callback = callback

    ...
```

Reading I/O - Main Function

```
def demo():
        ...
        read_output = "< No Output >"

        def read_action(v):
                data_val = p.read_from_attribute(
                   p._handle_tx)
                print("TX", data_val)

        p.on_read(read_action)
        ...
```

Writing I/O

Writing requires similar extension of the Class

Writing I/O - Class Definition - Part I

```
class BLESimplePeripheral:
        def __init__(self, ble, name="mpy-uart"):
                ...
                self._ble.irq(self._irq)
        ((self._handle_tx, self._handle_rx),) =
            self._ble.gatts_register_services((
            _UART_SERVICE,))
        ...
        self._write_callback = None


        ...
```

Writing I/O - Class Definition - Part II

```
...

def _irq(self, event, data):
        ...
        elif event == _IRQ_GATTS_WRITE:
    conn_handle, value_handle = data
    value = self._ble.gatts_read(
        value_handle)
    if value_handle == self._handle_rx
        and self._write_callback:
        self._write_callback(value)

...
```

Writing I/O - Class Definition - Part III

```
...

def on_write(self, callback):
self._write_callback = callback
```

Writing I/O - Main Function

```
def demo():
        ...
        p = BLESimplePeripheral(ble)
        ...
        def on_rx(v):
        print("RX", v)
        p._ble.gatts_write(p._handle_rx, v)

        p.on_write(on_rx)
        ...
```

Notify / Indicate

Notify is similar edits to Read and Write, but requires some additional considerations

- Note: When capturing signals the use of multiplexing or other workaround required to circumvent single process limitation

Notify / Indicate - Class Definition

```
class BLESimplePeripheral:
        def __init__(self, ble, name="mpy-uart"):
                ...
                ((self._handle_tx, self.
                    _handle_rx),) = self._ble.
                    gatts_register_services((
                    _UART_SERVICE,))
                ...

        ...

        def send(self, data):
        for conn_handle in self._connections:
            self._ble.gatts_notify(conn_handle,
                self._handle_tx, data)
```

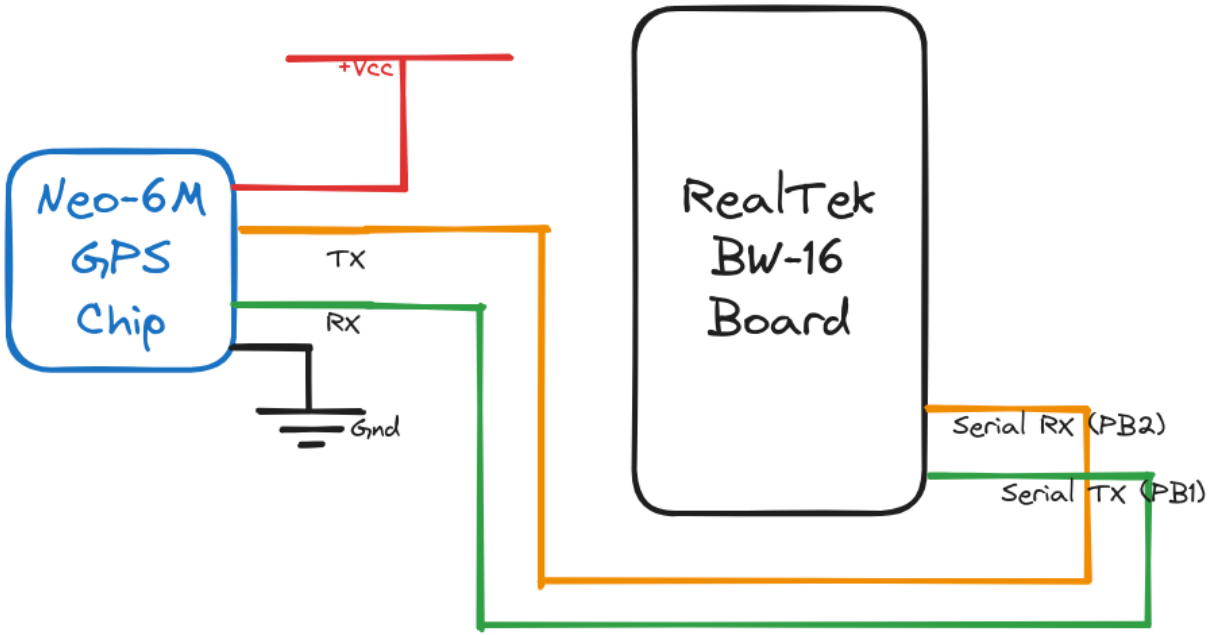
Notify / Indicate - Main Function

```
def demo():
        ...
        i = 0
    while True:
        if p.is_connected():
            led_onboard.on()
            for _ in range(3):
                data = str(i) + "_"
                print("TX", data)
                p.send(data)
                i += 1
        time.sleep_ms(100)
```

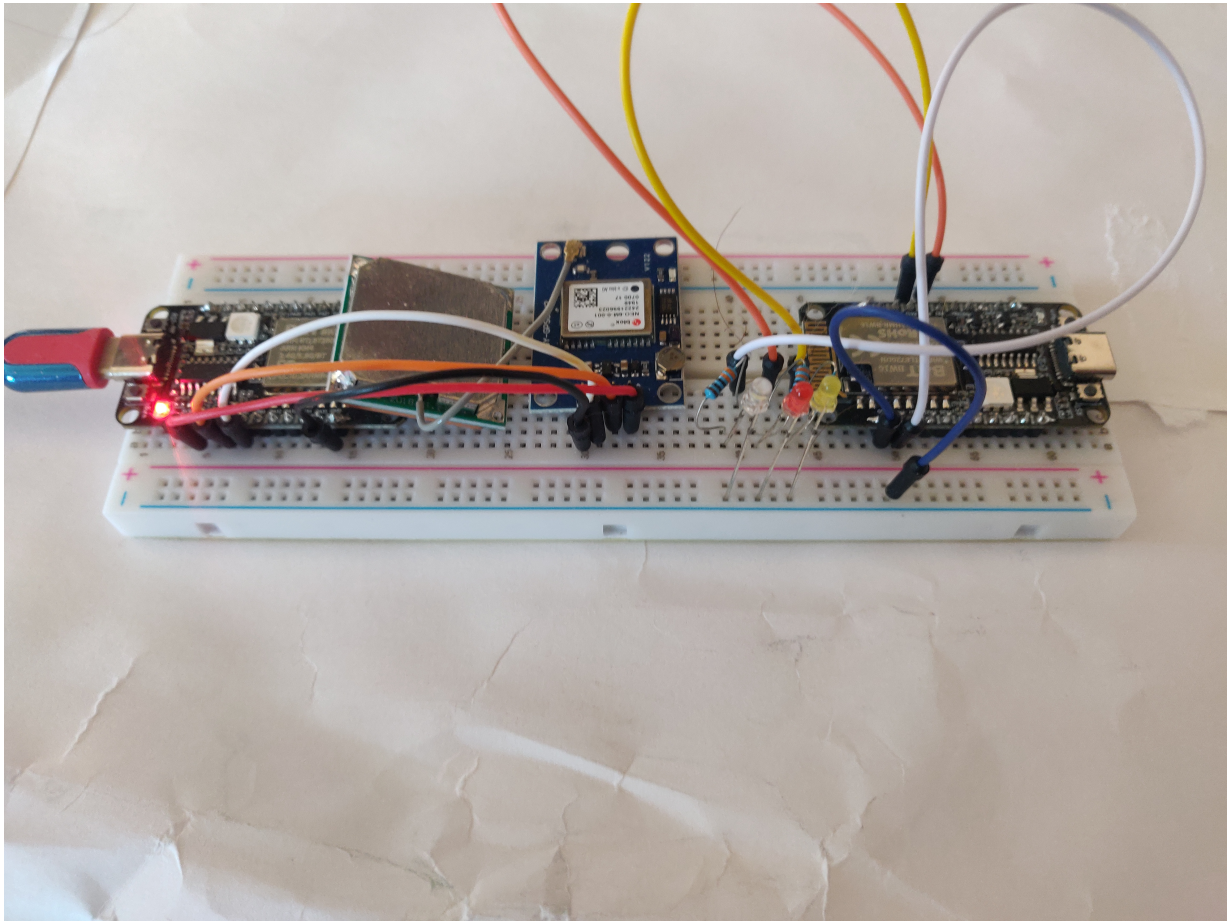
Where to find the Examples

No need to write this all yourself!

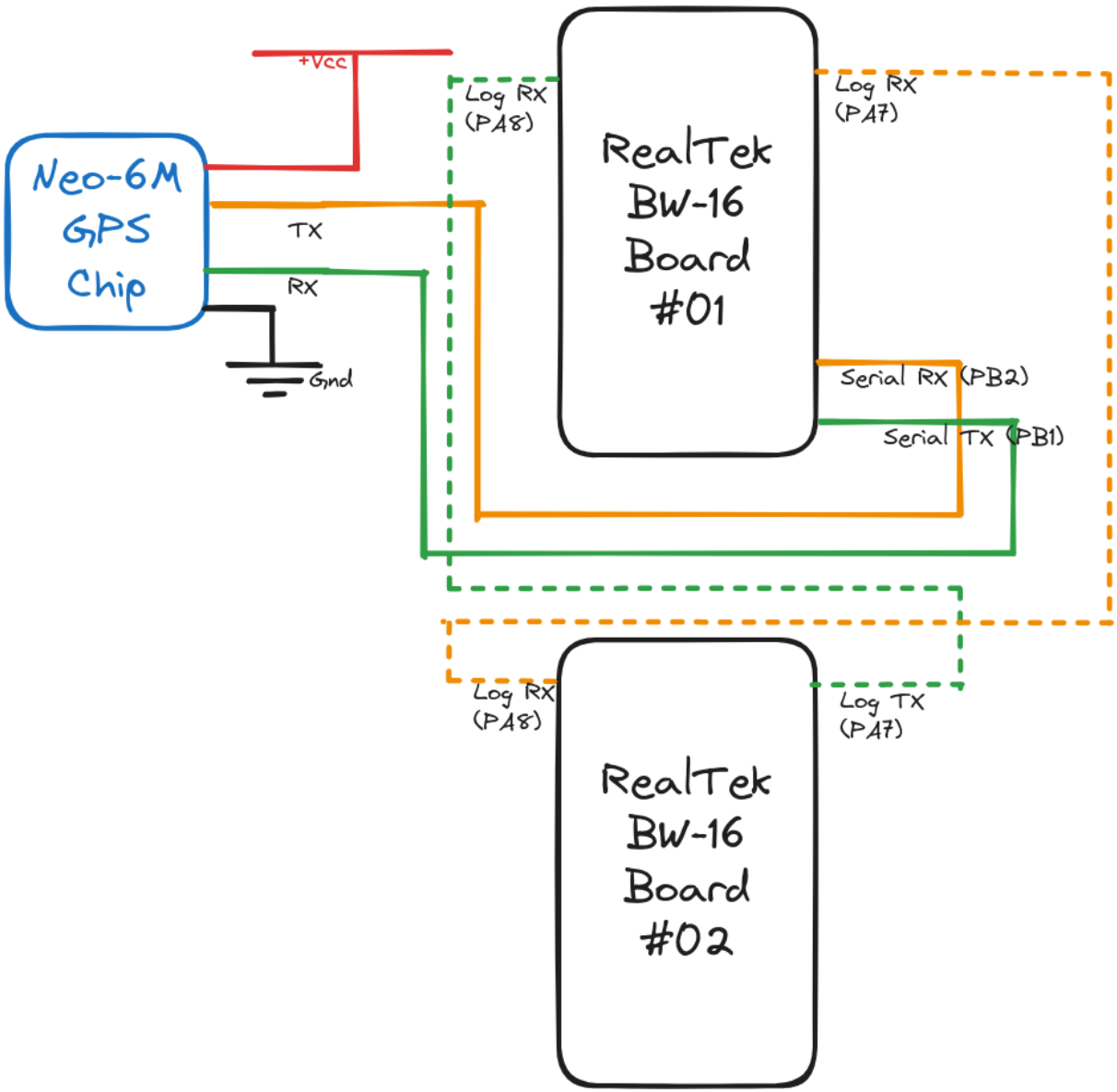
- Github for RealTek BW-16 Development / Testing
- URL: `https://github.com/Mauddib28/BW-16--BLE/tree/main`



GPS - Example - IRL



GPS - Example - Mk.II



Pain Points

Pain Points #001

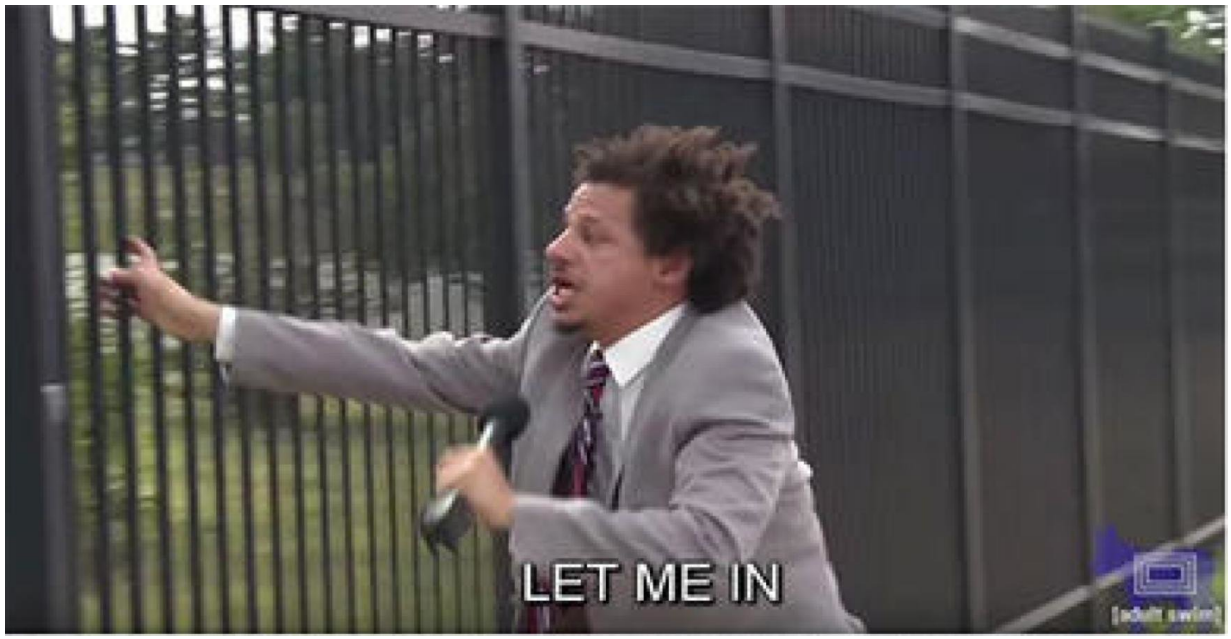
Why all the secrets?

- Hard to find good examples
  - Intended operation
  - Security is **Magical** and etherial
- If **ANYONE** find a good security/pairing example, *PLEASE* tell me!!!

Pain Points #002

Abstract this!

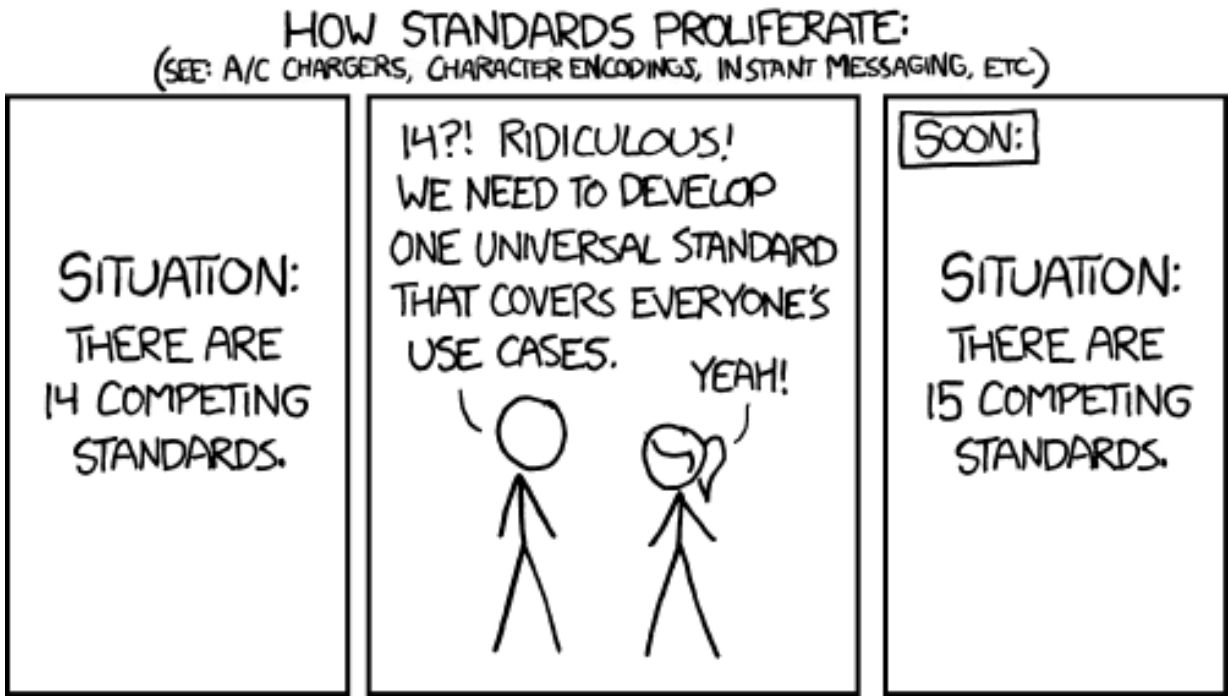
- Eventually hit bedrock
  - The rabbit hole has a wall in it
  - Lack of transparency in lower-levels
- Time for more Reverse Engineering!



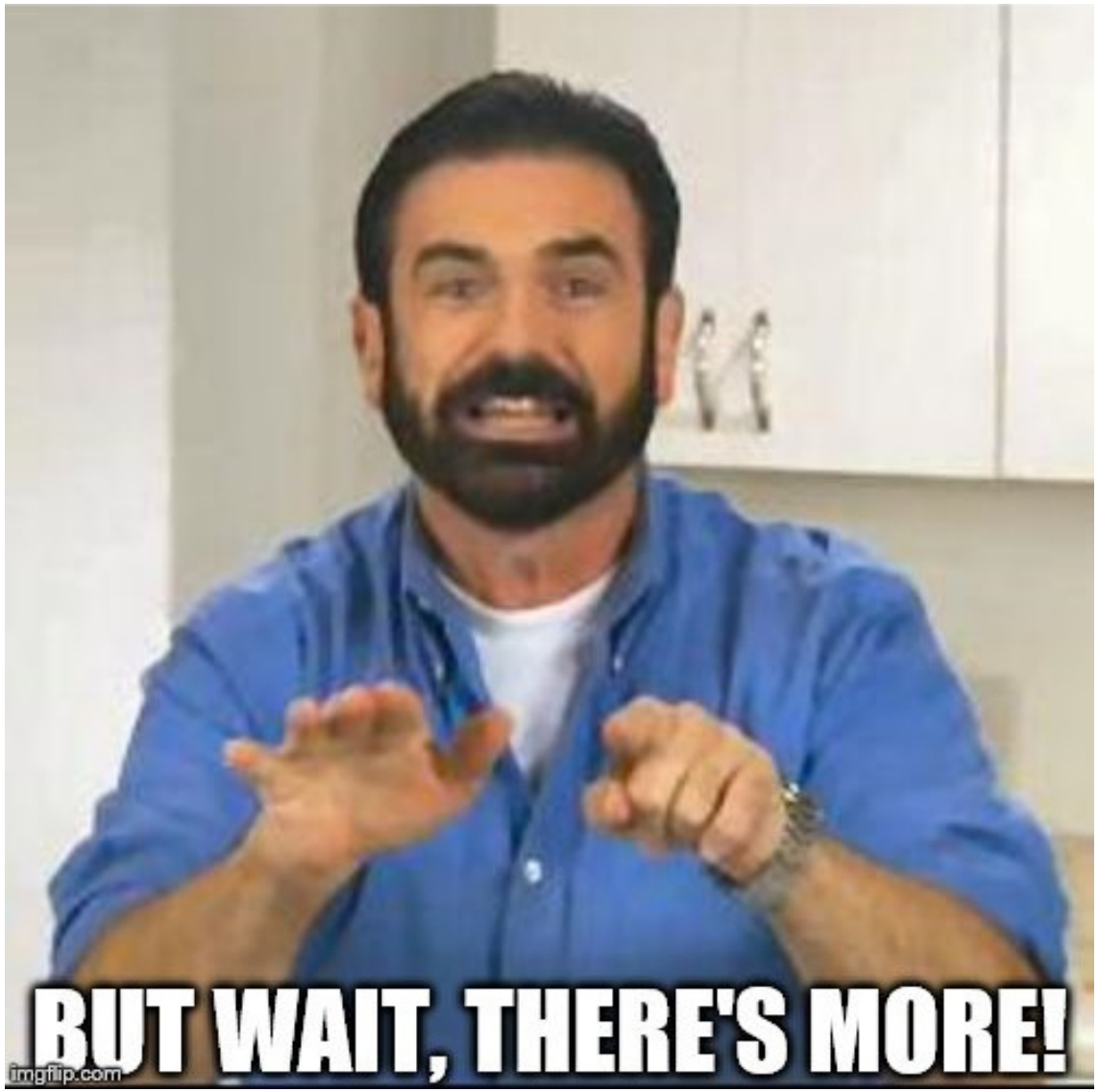
Pain Points #003

Everyone is the cutting edge developer?

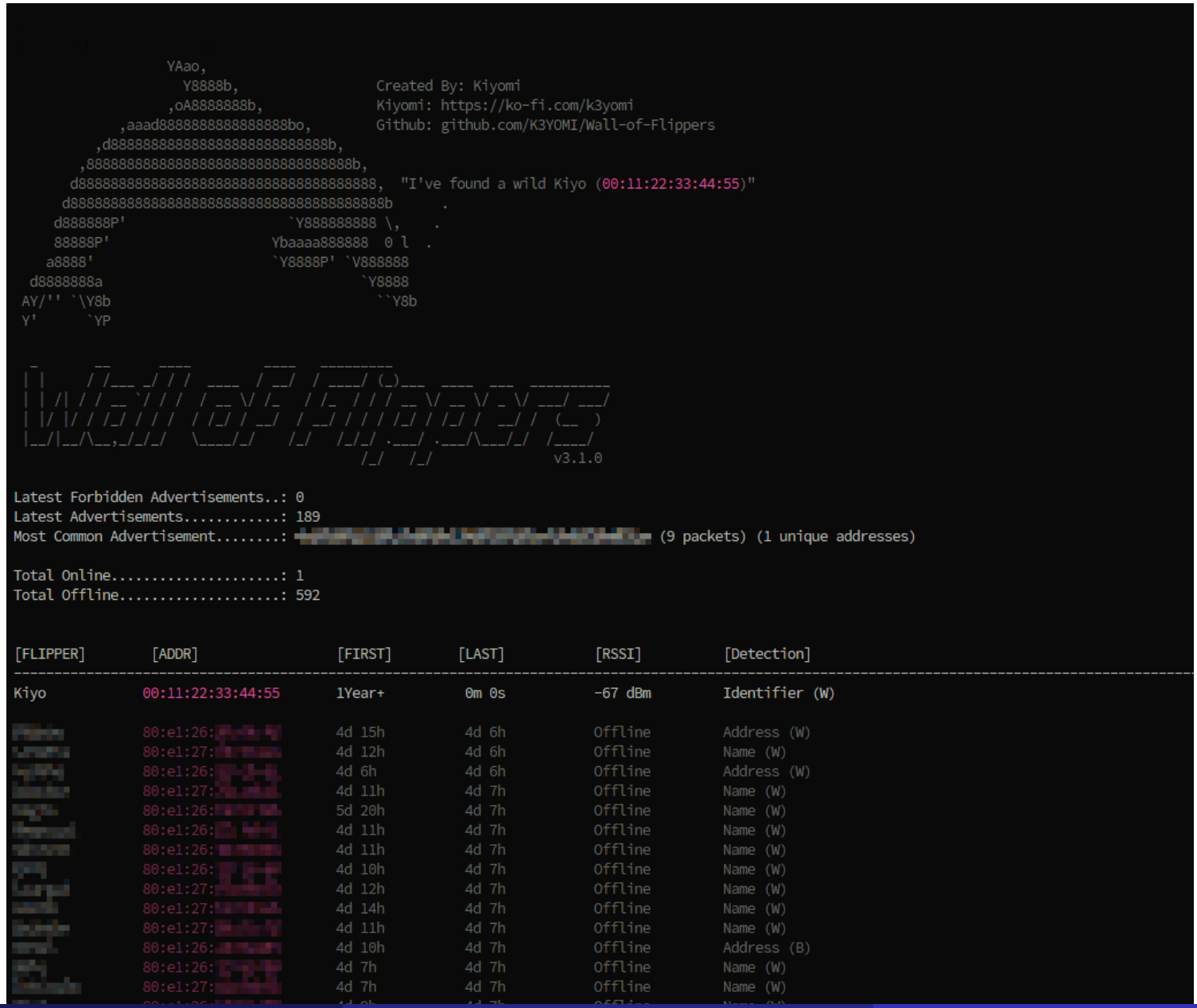
- Everyone can do whatever they want
- Even `btmon` returns Vendor Specific or Unknown
- Time to start cataloging...



What About the Hackers?



Wall of Flippers



Wall of Flippers

Identifies Flipper Devices

- Examines the Advertisement Packets
- Extracts the Device Address to determine if a now Flipper MAC

This project successfully funded on December 19, but you can still Late Pledge for available rewards.

BLEShark Nano: A Compact Wireless Multi-Tool for Hackers

A portable device for testing Bluetooth and Wi-Fi vulnerabilities—including games, apps, auto updates, and a smooth interface.

$81,540

pledged of **$3,482 goal**

1,709

backers

Back th

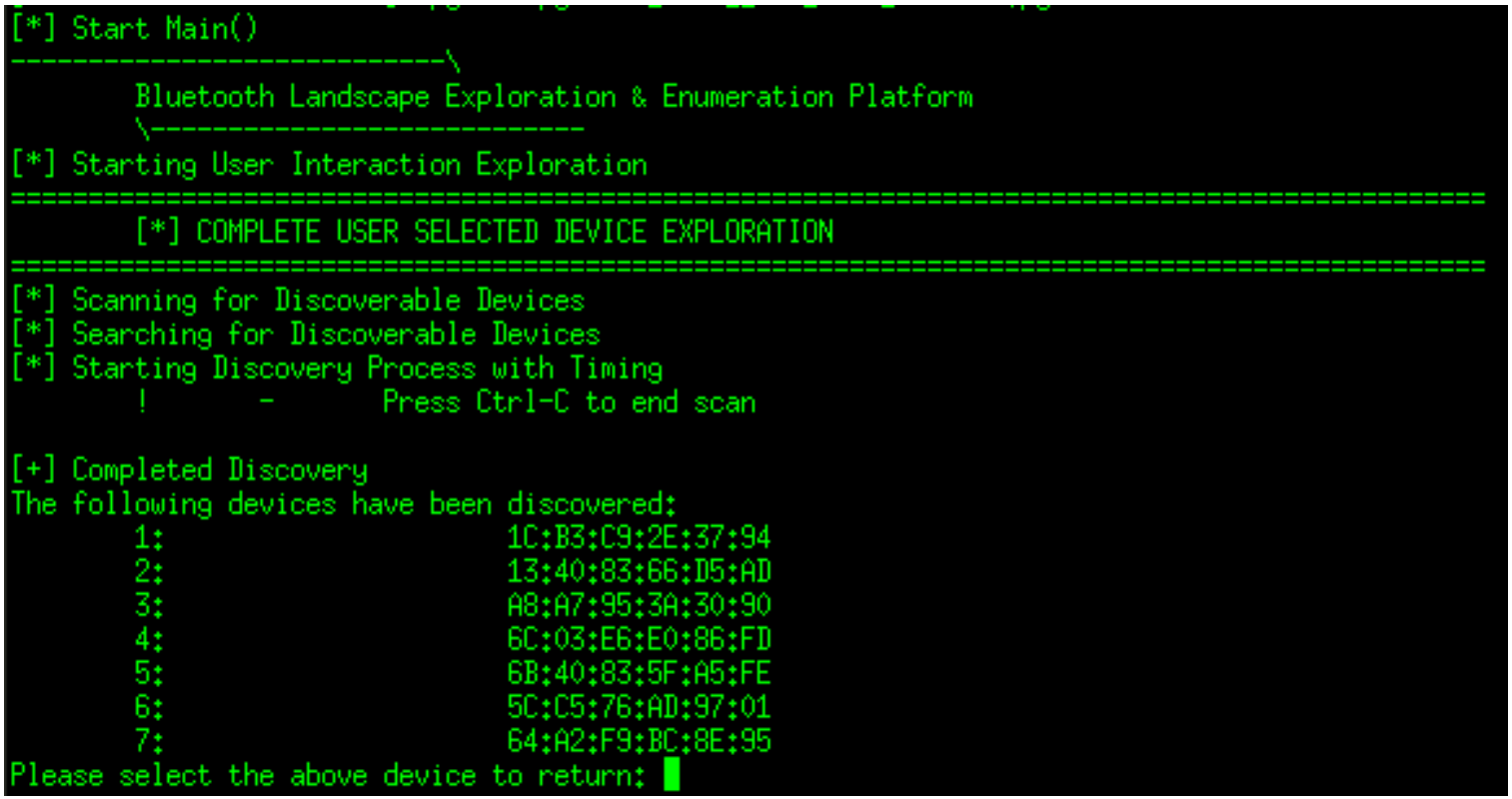
Save

BLE Nano Shark

- Combines several existing tools into one toy
- Leverages pairing capability to display requests
  - Most likely due to secondary or tertiary thread/process to handle pairing
  - Agent? Agent Manager?
  - Note: MOST interested in dissecting how this was done

Bluetooth Landscape Exploration & Enumeration Platform

- Git Repo: https://github.com/Mauddib28/bleep-tool



Safari Time - Spotted Zebra - First Half

```
[ DEVICE        -       4C:24:98:15:36:68 ]
Service -       service000c
-       Handle:                 None
-       UUID:                   00001801-0000-1000-8000-00805f9b34fb
        Characteristic  -       char000d
        -       UUID:                   00002a05-0000-1000-8000-00805f9b34fb
        -       Handle:                 None
        -       Flags:                  ['indicate']
        -       Value:                  []
        -       ASCii:                  -=!=- UNKNOWN -=!=-
                Descriptor      -                       desc000f
                -       UUID:                   00002902-0000-1000-8000-00805f9b34fb
                -       Flags:                  None
                -       Value:                  []
Service -       service0010
-       Handle:                 None
-       UUID:                   0000180a-0000-1000-8000-00805f9b34fb
        Characteristic  -       char0011
        -       UUID:                   00002a24-0000-1000-8000-00805f9b34fb
        -       Handle:                 None
        -       Flags:                  ['read']
        -       Value:                  [90, 68, 52, 49, 48]
        -       ASCii:                  ZD410
        Characteristic  -       char0013
        -       UUID:                   00002a25-0000-1000-8000-00805f9b34fb
        -       Handle:                 None
        -       Flags:                  ['read']
        -       Value:                  [53, 48, 74, 49, 57, 51, 49, 48, 54, 49, 48, 56]
        -       ASCii:                  50J193106108
        Characteristic  -       char0015
        -       UUID:                   00002a26-0000-1000-8000-00805f9b34fb
        -       Handle:                 None
        -       Flags:                  ['read']
        -       Value:                  [86, 56, 52, 46, 50, 48, 46, 49, 53, 90]
        -       ASCii:                  V84.20.15Z
        Characteristic  -       char0017
        -       UUID:                   00002a27-0000-1000-8000-00805f9b34fb
        -       Handle:                 None
        -       Flags:                  ['read']
        -       Value:                  [90, 68, 52, 49, 72, 50, 50, 45, 68, 48, 49, 69, 48, 48, 69, 90]
        -       ASCii:                  ZD41H22-D01E00EZ
```

Safari Time - Spotted Zebra - Second Half

```
        -       Flags:                  ['read']
        -       Value:                  [90, 68, 52, 49, 72, 50, 50, 45, 68, 48, 49, 69, 48, 48, 69, 90]
        -       ASCii:                  ZD41H22-D01E00EZ
        Characteristic  -       char0019
        -       UUID:                   00002a28-0000-1000-8000-00805f9b34fb
        -       Handle:                 None
        -       Flags:                  ['read']
        -       Value:                  [53, 46, 50]
        -       ASCii:                  5.2
        Characteristic  -       char001b
        -       UUID:                   00002a29-0000-1000-8000-00805f9b34fb
        -       Handle:                 None
        -       Flags:                  ['read']
        -       Value:                  [90, 101, 98, 114, 97, 32, 84, 101, 99, 104, 110, 111, 108, 111, 103, 105, 101, 115]
        -       ASCii:                  Zebra Technologies
        Characteristic  -       char001d
        -       UUID:                   00002a50-0000-1000-8000-00805f9b34fb
        -       Handle:                 None
        -       Flags:                  ['read']
        -       Value:                  [1, 241, 1, 28, 1, 17, 1]
        -       ASCii:                  b'\x01\xf1\x01\x1c\x01\x11\x01'
Service -       service001f
-       Handle:                 None
-       UUID:                   38eb4a80-c570-11e3-9507-0002a5d5c51b
        Characteristic  -       char0020
        -       UUID:                   38eb4a81-c570-11e3-9507-0002a5d5c51b
        -       Handle:                 None
```

Figure 5: Safari Time - Zebra Spotted - Second Half

Questions

Questions? Demo??

Bibliography References I

Freedesktop

What is D-Bus, Published 2022

https://www.freedesktop.org/wiki/Software/dbus/

Last Accessed: 2024-03-28 22:43:19 EST

GNU

Knowing the Details of D-Bus Services

https://www.gnu.org/software/emacs/manual/html_node/dbus/Introspection.html

Last Accessed: 2024-04-01 19:48:34 EST

Programiz

Python Decorators

https://www.programiz.com/python-programming/decorator

Last Accessed: 2024-04-01 19:52:34 EST

Bibliography References II

hbldh

characteristic.py

https://github.com/hbldh/bleak/blob/63adefa24cb6ed11c8cf154fa41f51ecff1df98c/bleak/backends/characteristic.py

Last Accessed: 2024-04-01 19:56:34 EST

elsampsa

Bluetooth + DBus + gobject demo

https://github.com/elsampsa/btdemo

Last Accessed: 2024-04-01 19:54:52 EST

Freedesktop

DbusTools, Published May 07 2021

https://www.freedesktop.org/wiki/Software/DbusTools/

Last Accessed: 2024-03-28 22:57:29 EST

Bibliography References III


Bluetooth SIG
assigned_numbers
https://bitbucket.org/bluetooth-SIG/public/src/main/assigned_numbers/
Last Accessed: 2024-04-01 21:00:29 EST


Bluetooth SIG
public bitbucket
https://bitbucket.org/bluetooth-SIG/public/src/main/
Last Accessed: 2024-04-02 15:13:33 EST


Archlinux Forum
Activation via systemd failed for unit dbus-org.bluez.service
https://bbs.archlinux.org/viewtopic.php?id=155714
Last Accessed: 2024-04-02 15:09:42 EST

Bibliography References IV

oscaracena

gattlib.h

https://github.com/oscaracena/pygattlib/blob/7d08c0805313201b2ab12628e19544bb180218a8/src/gattlib.h

Last Accessed: 2024-04-02 15:09:42 EST

Bluetooth SIG

Bluetooth for Linux Developers Study Guide - Versions 1.0, 1.0.1

https://www.bluetooth.com/bluetooth-resources/bluetooth-for-linux/

Last Accessed: 2021-12-29 10:26:27 EST, 2022-10-18 18:54:02 EST

Bibliography References V

Bluetooth SIG

Developer Study Guide Bluetooth Internet Gateways - Version 2.0.0

https://www.bluetooth.com/blog/the-bluetooth-internet-gateway-study-guide/

Last Accessed: 2021-07-21 14:46:56 EST

Bluetooth SIG

Bluetooth LE Developer Study Guide - Version 5.2.0

https://www.bluetooth.com/bluetooth-resources/bluetooth-le-developer-starter-kit/

Last Accessed: 2023-02-16 11:36:57 EST

Bibliography References VI

Bluetooth SIG

Bluetooth Core Specification - Versions 5.3, 5.4

https://www.bluetooth.com/specifications/specs/core-specification-5-[3—4]/

Last Accessed: 2023-12-19 13:24:22 EST, 2023-12-16 11:33:37 EST

Bluetooth SIG

Generic Attribute Profile (GATT)

https://www.bluetooth.com/wp-content/uploads/Files/Specification/HTML/Core-54/out/en/host/generic-attribute-profile–gatt-.html

Last Accessed: 2023-12-16 11:22:03 EST

Marcel Holtmann, Maxim Krasnyansky, Qualcomm

BlueZ - Bluetooth protocol stack for Linux

https://git.kernel.org/pub/scm/bluetooth/bluez.git/tree/doc

Last Accessed: 2024-04-11 10:39:23 EST