OWASP SEASIDES BLE - E

•••

It’s about changing the things

BLE - E

BLE Protocols

- HCI - Host Controller Interface
- L2CAP - Logical Link Control And Adaptation Protocol
- RFCOMM - Radio Frequency communication protocol
- SDP - Service Discovery Protocol
- BNEP - Bluetooth Network Encapsulation Protocol
- ATT - Attribute Protool
- SMP - Security Manager Protocol

BLE Profiles

- GAP - Generic Access Profile
- SPP - Serial Port Profile
- PAN - Personal Area Network
- HSP - HeadSet Profile
- HFP - Hands Free Profile
- GAP LE - Generic Access Protocol Low Energy
- GATT -- Generic Attribute Profile

https://en.wikipedia.org/wiki/List_of_Bluetooth_profiles

???????? Target for today

- HCI - Host Controller Interface
- SDP - Service Discovery Protocol
- GAP LE - Generic Access Protocol Low Energy
- GATT -- Generic Attribute Profile
- L2CAP - Logical Link Control And Adaptation Protocol
- RFCOMM - Radio Frequency communication protocol

Bluetooth Versions

LMP	Bluetooth Version
0	Bluetooth 1.0b
1	Bluetooth 1.1
2	Bluetooth 1.2
3	Bluetooth 2.0 + EDR
4	Bluetooth 2.1 + EDR
5	Bluetooth 3.0 + HS
6	Bluetooth 4.0
7	Bluetooth 4.1
8	Bluetooth 4.2
9	Bluetooth 5
10	Bluetooth 5.1

Demo - Identify the version of Your BLE Device

#hcitool scan or lescan

#hcitool info or leinfo

```
v@iotpentest:~$ sudo hcitool info [redacted]
Requesting information ...
        BD Address:  [redacted]
        OUI Company: Cambridge Executive Limited ([redacted])
        Device Name: [redacted]
        LMP Version: 4.2 (0x8) LMP Subversion: 0x2fb8
        Manufacturer: Cambridge Silicon Radio (10)
        Features page 0: 0xff 0xff 0x8f 0xfe 0xdb 0xff 0x5b 0x87
                <3-slot packets> <5-slot packets> <encryption> <slot offset>
                <timing accuracy> <role switch> <hold mode> <sniff mode>
                <park state> <RSSI> <channel quality> <SCO link> <HV2 packets>
                <HV3 packets> <u-law log> <A-law log> <CVSD> <paging scheme>
                <power control> <transparent SCO> <broadcast encrypt>
                <EDR ACL 2 Mbps> <EDR ACL 3 Mbps> <enhanced iscan>
                <interlaced iscan> <interlaced pscan> <inquiry with RSSI>
                <extended SCO> <EV4 packets> <EV5 packets> <AFH cap. slave>
                <AFH class. slave> <LE support> <3-slot EDR ACL>
                <5-slot EDR ACL> <sniff subrating> <pause encryption>
                <AFH cap. master> <AFH class. master> <EDR eSCO 2 Mbps>
                <EDR eSCO 3 Mbps> <3-slot EDR eSCO> <extended inquiry>
                <LE and BR/EDR> <simple pairing> <encapsulated PDU>
                <non-flush flag> <LSTO> <inquiry TX power> <EPC>
```

Lets get hands dirty a little ... Not so Fast

Requirements to test BLE

Hardware

1. CSR 4.0 & Small Dongles
2. Ubertooth
3. Good configuration laptop
4. Any Cheap or Vulnerable device buy from the robu or banggood
5. ESP32 -- Microcontroller - Wifi and BLE

UD100

- Supports Bluetooth stack v4.0
- USB 2.0
- Supports Bluetooth DUN, FAX, SPP, HID, FTP, OPP, SDP, HCRP, LAN, OBEX FTP, OBEX OPP, OBEX BIP, BIP, AVRCP, A2DP, HSP, HFP, PAN, BPP, Headset, AVCTP, AVDTP, HDP, Find Me, Proximity, Health Thermometer, Heart Rate, HID OVER GATT profiles
- Supports up to 7 simultaneous connections
- Easy to use Windows configuration tool available
- Bluetooth driver needed (Bluesoleil driver)
- Easy to use Windows configuration tool available
  Working distance (In an open field): Normally 300 meters, up to 600 meters using 5 dipole antenna

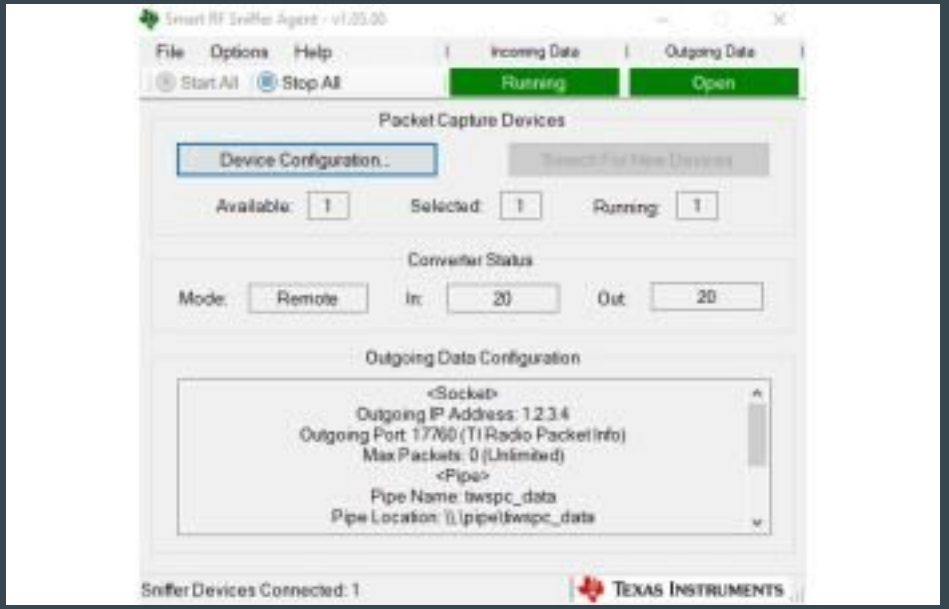
http://www.senanetworks.com/ud100-g03.html

Ubertooth and NRF52840

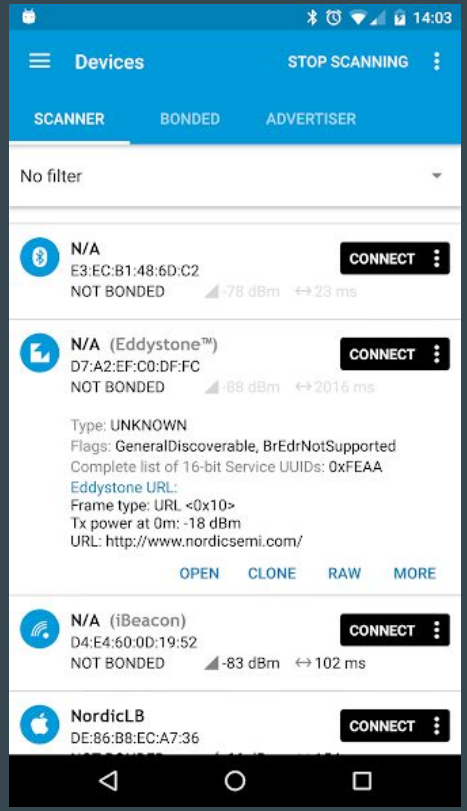
- 2.4 GHz transmit and receive.
- Transmit power and receive sensitivity comparable to a Class 1 Bluetooth device.
- standard Cortex Debug Connector (10-pin 50-mil JTAG).
- In-System Programming (ISP) serial connector.
- expansion connector: intended for inter-Ubertooth communication or other future uses.
- six indicator LEDs.

CC2540 sniffer board USB

Smart Rf sniffer agent



NRF Connect APP - Android



Required tools to test the BLE

1. Bluez (hcitool)
2. Gatttool
3. Btproxy
4. Bettercap
5. Wireshark
6. Btlejack
7. Btle juice
8. NRF Connect APP
9. Gattacker
10. sdptool
11. Etc

Depends on requirement we can install the tools

What we need to test

Core concepts in BLE

Core concepts in BLE

There are two basic concepts in BLE.

- GAP - Generic Access Profile
- GATT - Generic Attribute Protocol

Core Concepts in BLE

Generic Access Profile (GAP)

This is responsible for the connections and advertising in BLE. GAP is responsible for the visibility of a device to the external world and also plays a major role in determining how the device interacts with other devices.

The following two concepts are integral to GAP:

Peripheral devices :

These are small and low energy devices that can connect with complex, more powerful central devices. Heart rate monitor is an example of a peripheral device.

Central devices :

These devices are mostly cell phones or gadgets that have an increased memory and processing power.

Generic Attribute Profile

Making use of a generic data protocol known as Attribute Protocol, ATT determines how two BLE devices exchange data with each other using concepts -

- Characteristics
- Services

Services:

A service can have many characteristics. Each service is unique in itself with a universally unique identifier (UUID) that could either be 16 bit in size for official adapted services or 128 bit for custom services.

Characteristics:

Characteristics are the most fundamental concept within a GATT transaction. Characteristics contain a single data point and akin to services, each characteristic has a unique ID or UUID that distinguishes itself from the other characteristic. For example HRM sensor data from health bands etc.

GAP (Advertising and Connections)

Table 3-1. Modes and their applicable procedures

Mode	Applicable Role(s)	Applicable Peer Procedure(s)
Broadcast	Broadcaster	Observation
Non-discoverable	Peripheral	N/A
Limited discoverable	Peripheral	Limited and General discovery
General discoverable	Peripheral	General discovery
Non-connectable	Peripheral, broadcaster, observer	N/A
Any connectable	Peripheral	Any connection establishment

Conversely, Table 3-2 shows the modes that the peer needs be in to perform each of the listed GAP procedures.

Table 3-2. Procedures and their required modes

Procedure	Applicable Role(s)	Applicable Peer Mode(s)
Observation	Observer	Broadcast
Limited discovery	Central	Limited discoverable
General discovery	Central	Limited and General discoverable
Name discovery	Peripheral, central	N/A
Any connection establishment	Central	Any connectable
Connection parameter update	Peripheral, central	N/A

https://www.bluetooth.com/specifications/assigned-numbers/generic-access-profile/

GATT (Services and Characteristics)

Services:

GATT services group conceptually related attributes in one common section of the attribute information set in the GATT server.

Characteristics:

You can understand characteristics as containers for user data. They always include at least two attributes: the *characteristic declaration* (which provides metadata about the actual user data) and the *characteristic value* (which is a full attribute that contains the user data in its value field).

Understanding Bluetooth security


THANK GOD.
FINALLY.
memegenerator.net

One of the best communication platform for the IoT devices to share and communicate and for operate device is Bluetooth low energy protocol

- Bluetooth standard – Non Secure one
- Bluetooth Low Energy – is Secure one
- Bluetooth 4.0 – vulnerable
- 4.1 – vulnerable
- 4.2 – vulnerable
- 5 , 5.1 – current in market (no - 5.0)

Pairing in bluetooth

Phase One:

Attribution Protocol (ATT) values. These live at layer 4 with L2CAP, and are typically not ever encrypted

Phase Two

The purpose is to generate a Short Term Key (STK). This is done with the devices agreeing on a Temporary Key (TK) mixed with some random numbers which gives them the STK.

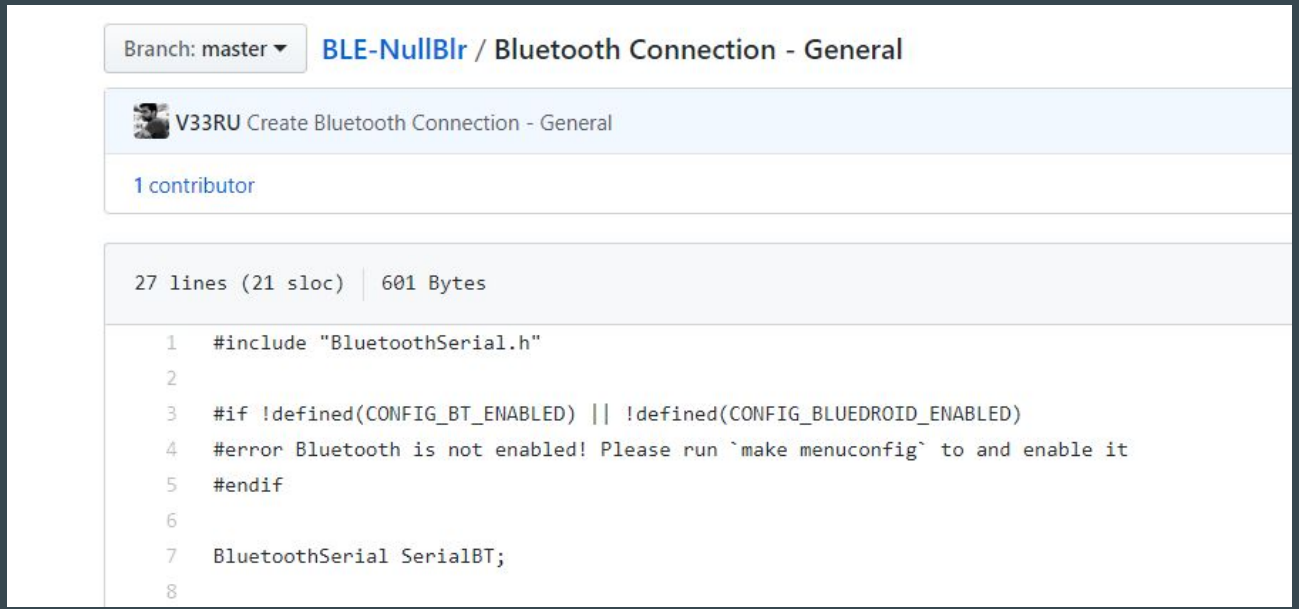
Phase Three

If an LTK wasn't generated in phase two, one is generated in phase three. Data like the Connection Signature Resolving Key (CSRK) for data signing and the Identity Resolving Key (IRK) for private MAC address generation and lookup are generated in this phase.

Demo - Let's try to connect with ESP 32

Code available in the github

Try with the Basic Connection of Bluetooth



Demo - ESP32 with BLE-Security

Key Pairing Code from the

Branch: master ▾ BLE-NullBlr / BLE- PassKey

V33RU Create BLE- PassKey

1 contributor

76 lines (63 sloc) | 2.41 KB Raw Blame

```
/*
    Based on Neil Kolban example for IDF: https://github.com/nkolban/esp32-snippets/blob/master/cpp_utils/tests
    Ported to Arduino ESP32 by Evandro Copercini
*/

#include <BLEDevice.h>
#include <BLEUtils.h>
#include <BLEServer.h>

// See the following for generating UUIDs:
// https://www.uuidgenerator.net/
```

flags? & property? & handle? & UUID?


Damn it...
Not again.
quickmeme.com

An example responses of primary and characteristics

```
[LE]> primary
attr handle: 0x0001, end grp handle: 0x0005 uuid: 00001801-0000-1000-8000-00805f9b34fb
attr handle: 0x0014, end grp handle: 0x001e uuid: 00001800-0000-1000-8000-00805f9b34fb
attr handle: 0x0028, end grp handle: 0xffff uuid: ade3d529-c784-4f63-a987-eb69f70ee816
[LE]> characteristics
handle: 0x0002, char properties: 0x20, char value handle: 0x0003, uuid: 00002a05-0000-1000-8000-00805f9b34fb
handle: 0x0015, char properties: 0x02, char value handle: 0x0016, uuid: 00002a00-0000-1000-8000-00805f9b34fb
handle: 0x0017, char properties: 0x02, char value handle: 0x0018, uuid: 00002a01-0000-1000-8000-00805f9b34fb
handle: 0x0029, char properties: 0x22, char value handle: 0x002a, uuid: e9241982-4580-42c4-8831-95048216b256
handle: 0x002b, char properties: 0x0a, char value handle: 0x002c, uuid: ad7b334f-4637-4b86-90b6-9d787f03d218
```

BLE FLAGS -

Very Very Very Important

Used to set limited or general discovery mode

- 0x00 Display Only
- 0x01 Display Yes/No (both a display and a way to designate yes or no)
- 0x02 Keyboard Only
- 0x03 No Input/No Output (e.g. headphones)
- 0x04 Keyboard Display (both a keyboard and a display screen)
- 0x05-0xFF Reserved

Property

Read or Write

Broadcast

Write without response

Notify

Indicate

Signed with write command

Queed write

Write auxiliaris

Handle

The attribute handle is a unique 16-bit identifier for each attribute on a particular GATT server.

```
handle = 0x0015, char properties = 0x02, char value handle = 0x0016, uuid = 00002a00-0000-1000-8000-00805f9b34fb
```

Value

The attribute value holds the actual data content of the attribute.

```
handle = 0x0019, char properties = 0x02, char value handle = 0x001a, uuid = 00002aa6-0000-1000-8000-00805f9b34fb
```

Characteristic Descriptors

GATT characteristic descriptors (commonly called simply *descriptors*) are mostly used to provide the client with *metadata* (additional information about the characteristic and its value).

```
v@mr-iot:~$ sudo gatttool -b 24:0A:C4:30:F0:6A --char-read -a 0x0016
Characteristic value/descriptor: 32 62 30 30 30 34 32 66 37 34 38 31 63 37 62 30 35 36 63 34 62 34 31 30 64 32 38 66 33 33 63 66
```

UUID

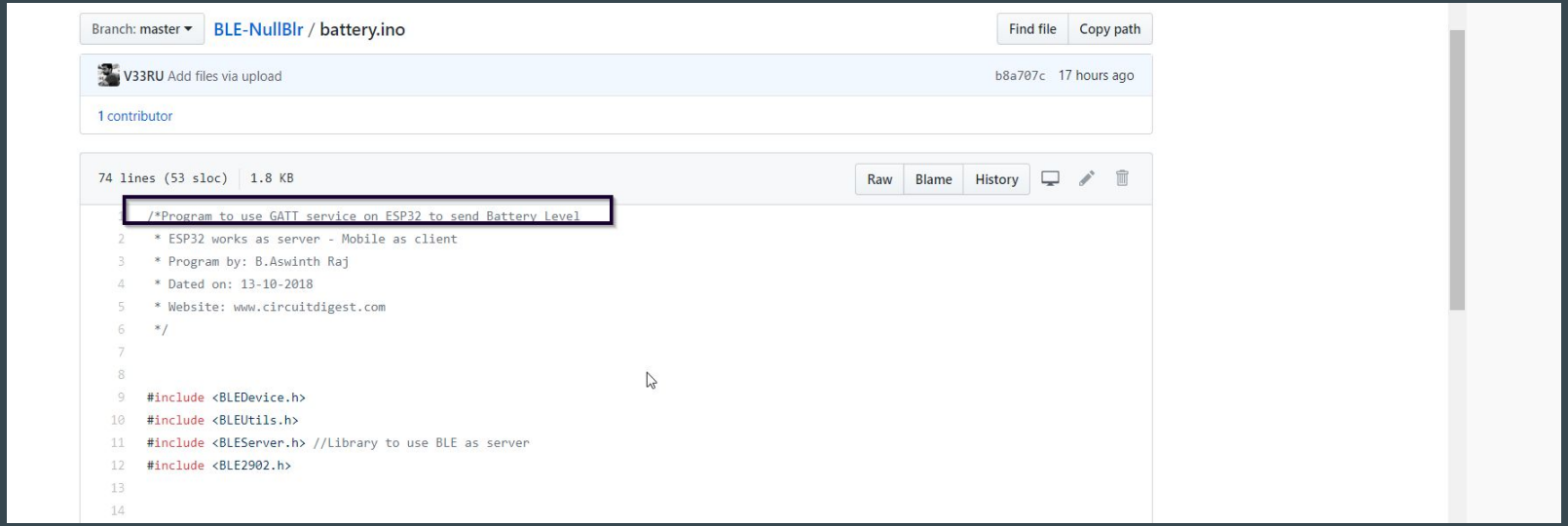
A universally unique identifier (UUID) is a 128-bit (16 bytes) number that is guaranteed

```
uuid: 00001801-0000-1000-8000-00805f9b34fb
```

Identify the service from the UUID

Demo - Battery Indicator

- Bluetooth headset is having the Battery indication and call functionality
- After flashing ESP32 check in NRF connect APP



Lets install the tools

1. -- apt install bluez (hci tools and gatttool is installed)(http://www.bluez.org/)
2. -- bettercap (https://github.com/bettercap)
3. -- btlejuice (https://github.com/DigitalSecurity/btlejuice)
4. -- btlejack (https://github.com/virtualabs/btlejack)

Which important will install first remaining later for the practice

Tools which is going to use

hcitool:

It makes use of the host controller interface in a laptop to communicate and read/write changes to BLE devices. hcitool is therefore, useful in finding out the available victim BLE device that advertises, and then in changing the values after connection.
The values/data can only be changed if one knows the service and characteristic the data is coming from. In order to find out the relevant services and characteristics, one may use a gatttool.

gatttool:

As mentioned in the previous paragraph, gatttool is mainly helpful in finding out the services and characteristics of an available BLE device so that the victim's data can be read/written according to the attacker.

Walkthrough Commands

--- hcitool -h and man hcitool

--- gatttool -h and man gatttool

Lets get little understand about the commands

Usage

hciconfig : Used to list all the attached BLE adapters.

hciconfig hciX up : Enable the BLE adapter named hciX.

hciconfig hciX down : Disable the BLE adapter named hciX.

hcitool lescan : Scan for BLE devices in the vicinity.

gatttool -I : Launches gatttool in an interactive REPL like mode where the user can various issue commands as listed below.

connect <addr> : Connect to the BLE device with the specified address.

gatttool -t random -b <addr> -I : Connect to the device using a random address.

Primary

Characteristics

Start scan devices

. turn on the vulnerable device (smart band or smart watch)

-- run the below command

##hcitool lescan

Note the MAC address of the device

BLE Exploitation

Try to connect the device

Try to get the information about the device

BLE Exploitation

Connect with gatttool

##gatttool -I connect <ble address>

##primary

##characteristics

BLE Exploitation

Identify the read/write characteristics

##char-desc

Filter displayed handles

##char-desc 01 05

Find read characteristic

##char-read-hnd <handle>

BLE Exploitation

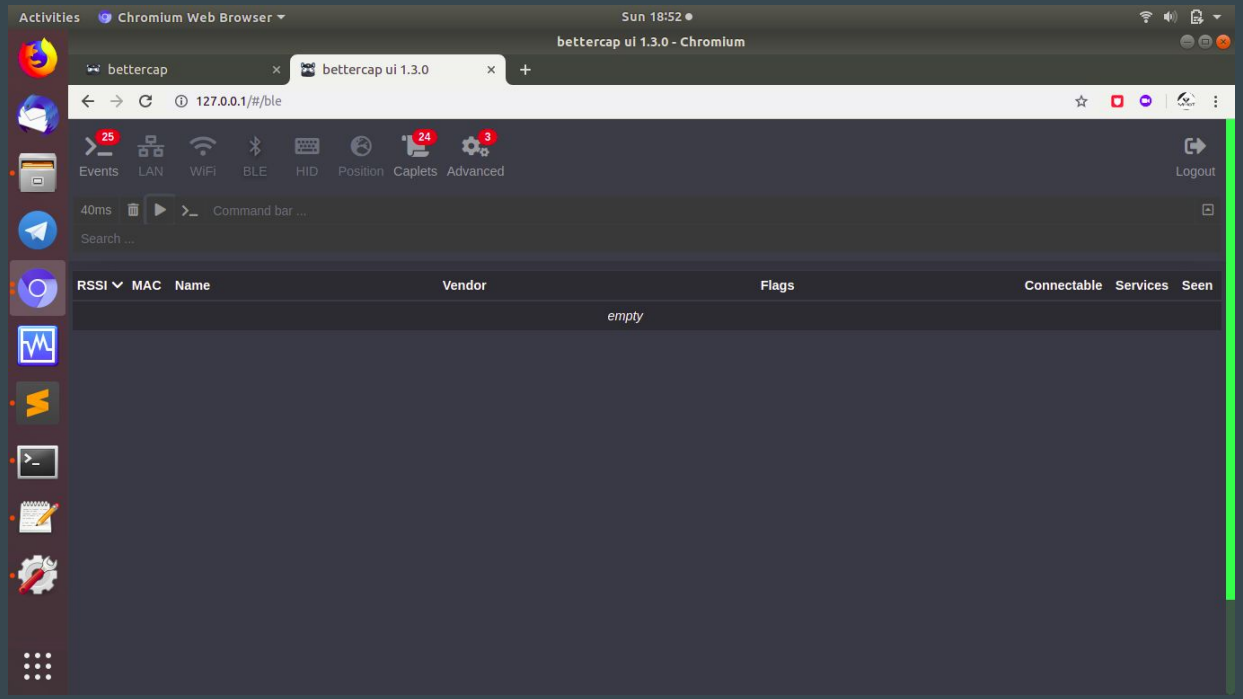
Write the data to characteristic

##char-write-req (or) char-write-cmd

A Successful write request shows hack a vulnerable device

Bettercap With UI

sudo bettercap -caplet http-ui



Useful commands

sudo hcitool lescan --duplicate

sudo hcidump --raw

sudo gatttool -t random -b C7:17:1D:43:39:03 -I

sudo gatttool -b 20:FA:BB:0C:50:EB --char-write-req -a 0x111e -n 121

sudo sdptool browse --l2cap --tree 68:27:37:44:7B:47

Completed task as per the session details

Missing something

What about these areas

Android (code in app)
iOS (code in app)
Hardware chips (physical access)
L2cap (communication)
RFCOMM (communication)

App Reversing for the BLE

Use the tools like apkpure(to download the apk)

apktool to reverse engineering

Check Manifest file

BluetoothAdapter, startLeScan, LeScanCallback, BluetoothGatt

CTF Time

Flash your ESP32 with little modified Lab of @hackgnar

Thank You

FYI

https://github.com/nayarsystems/virkey

https://randomnerdtutorials.com/esp32-bluetooth-low-energy-ble-arduino-ide/

https://eprint.iacr.org/2013/309.pdf

https://medium.com/rtone-iot-security/deep-dive-into-bluetooth-le-security-d2301d640bfc

https://labs.mwrinfosecurity.com/assets/BlogFiles/mwri-android-bluetooth-pairing-bypass-2016-04-12.pdf