# BLE Workshop

## Learning to Hack Bluetooth Low Energy with BLE CTF

Ryan Holeman - DEF CON 33 - Aug 2025

# Ryan Holeman

- CISO @ Stability AI
- PhD (ABD) student at DSU
- Austin, Texas
- Dad
- Hobby addict (🛹🏂🏄‍♂️)
- Regular speaker
- Hacker

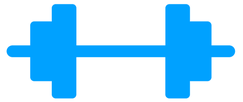
# Agenda

**BLE Basics**

- Protocols
- Stacks
- Hardware
- Software

**Workshop Essentials**

- GATT
- BLE CTF
- Tools



**Training**

- +20 exercises

# Before We Start

- Many things I say during this class will not be 100% accurate
- I use a lot of analogies and comparisons to accelerate your understanding
- Setup: https://github.com/hackgnar/ble_ctf/blob/master/docs/workshop_setup.md
- **Install Ubuntu 20.04, 22.04, or 24.04 if you want to be able to complete the exercises without major heartache or coding**
- Feel free to use LLMs for the workshop
- Be responsible with what you learn

# Here We Go!

- Things are going to get a bit technical in the next few slides
- Don’t worry if you don’t understand it all
- You don’t need to understand wifi and tcp in order to do web application hacking

# Bluetooth - BLE vs Basic Rate

**BLE (Smart, 4.x, 5.x)**

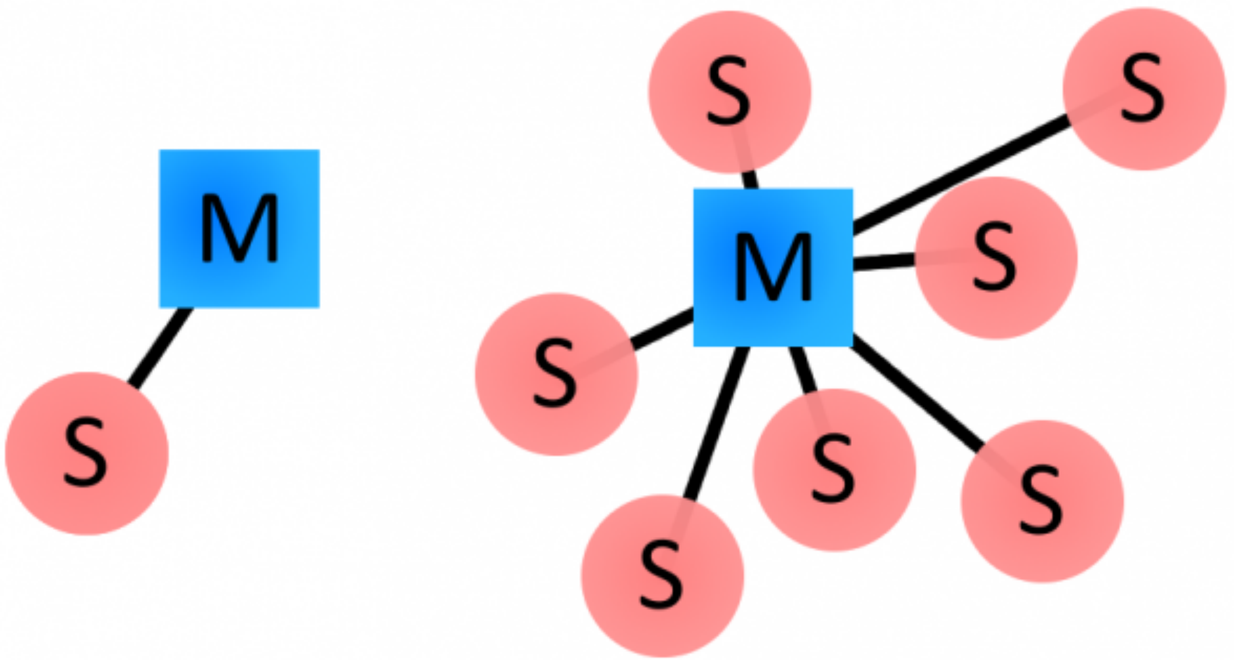
- Our focus for today
- More prevalent now-a-day
- Less channels - 32
- Easier to sniff

**Basic Rate (aka Classic, 2.0)**

- More channels - 89
- Focus area of tools, talks & hardware older than 10-15 years ago
- Harder to sniff and discover
- Still in use today - Devices with bigger batteries, keyboards, cars, etc

# Client Server Topology

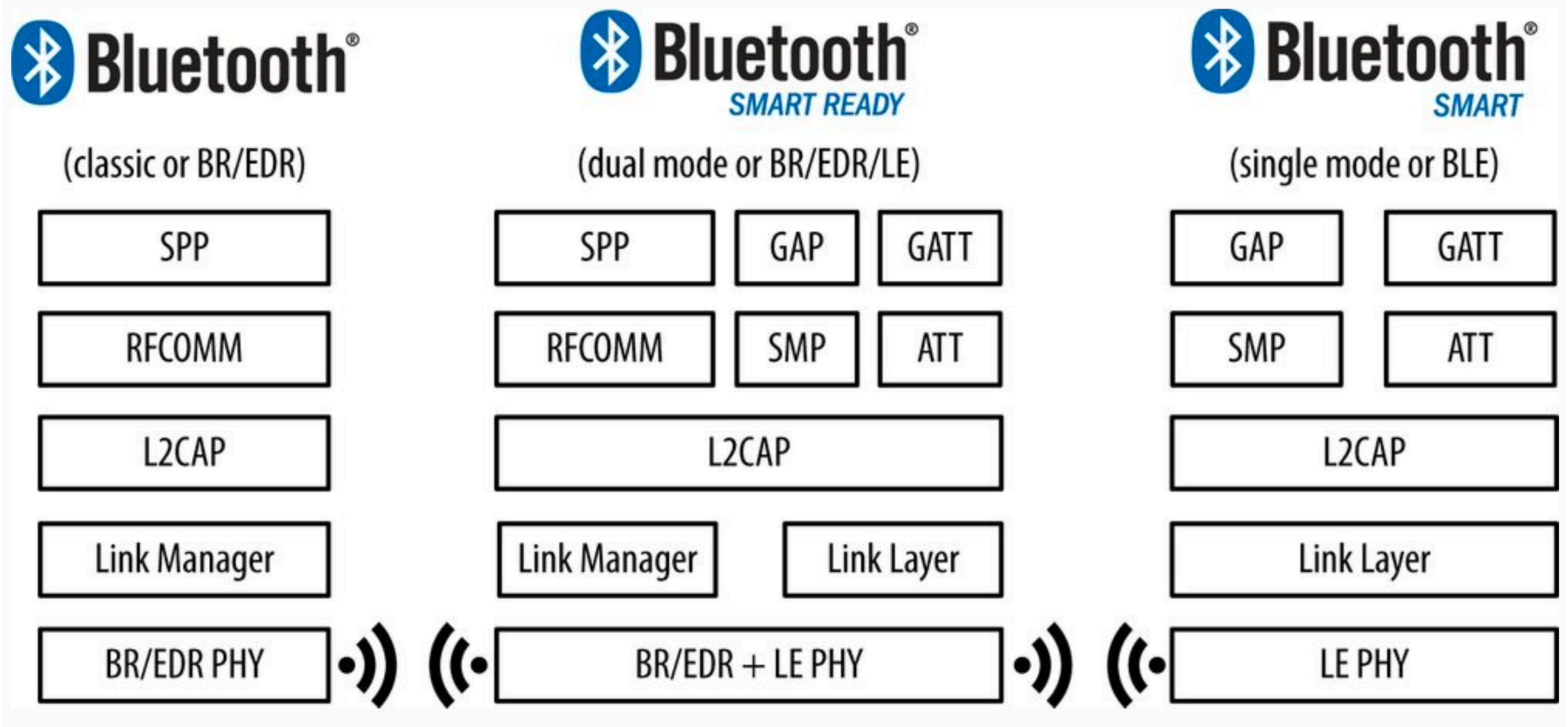
- Master
  - computer or phone
- Slave
  - watch, earphones, mouse, keyboard, heart rate monitor, etc



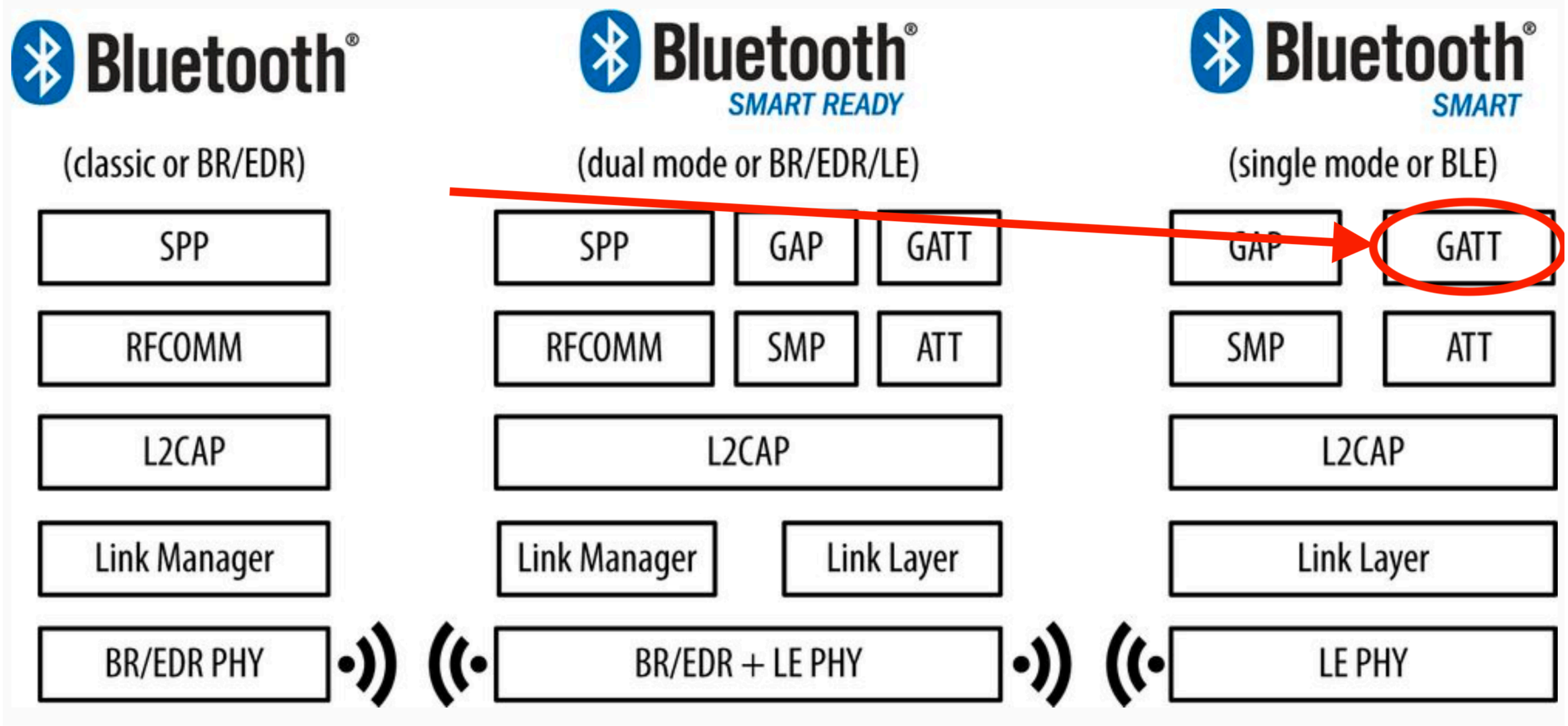
# Connection Types

- Paired vs Unpaired
- Authentication
  - In band
  - Out of band
- Encryption
  - Most of this is typically handled via OS abstraction
  - It is also limited by the master’s service implementation

# Bluetooth Stacks



# Bluetooth Stacks



# Hardware

- One of the main questions I get from people
- Also one of the most misunderstood areas of people getting into Bluetooth hacking
- You don’t need all sorts of fancy tools $$$

# Hardware

## Various devices

- What do they all do?
- What do I really need?

# Bluetooth Devices

## Categories

- **Host devices** - Computers and phones with BT support
- **BT Peripheral & IoT devices** - Keyboards, headphones, microcontrollers (i.e Esp32s, nordic chips, etc )
- **Sniffers** - Software defined radio & microcontrollers w/ custom defined baseband protocols (i.e. cc111x, nrf51x, etc)
- **Hybrids** - sniffer microcontrollers with BT firmware or custom BT stacks

# Bluetooth Devices

## Host devices

- Likely all most people need
- Allows you to host bluetooth services or connect to bluetooth devices over the standard bluetooth protocol
- Some support different protocols (i.e. 3.0, 4.0, BTBR, BLE, etc)
- Some have different ranges
  - Class 1-3
- Some support external antennas
  - UD100

# Bluetooth Devices

## Host device tools

- Gatttool & bluetoothctl
  - The curl of bluetooth
- NRF Connect
  - An iOS and Android application for interacting with BT devices
- Various others
  - WHAD, Bettercap, Bluez tools, code libraries, and more

# Bluetooth Devices

## Peripherals & IoT devices

- Can host firmware to act as standard BT clients or servers
- Some can be used as sniffers
  - Nordic 4x & 5x based chipsets
- Firmware libraries depend on chipsets
- Capabilities vary based on firmware api support
- Mostly all C code based

# Bluetooth Devices

## Peripherals & IoT devices tools

- Mostly just SDKs
  - Nordic
  - Expressif
  - Bluedroid
  - Many more
- You are basically just creating firmware with BT support

# Bluetooth Devices

## Sniffers & Hybrids

- Allow you to passively sniff bluetooth traffic
- Can be used for various types of injection or BT protocol simulation
- Can not **typically** be used as standard Bluetooth hosts or clients. This is becoming less true as tools mature.
- Operate at the PHY layer
- Require custom firmware & host software
- **Not needed for this workshop**


# Bluetooth Devices

## Sniffer & Hybrid tools

- WHAD - Wireless HAcking Devices
  - DEF CON 32 - Romain Cayre & Damien Cauquil
- Ubertooth-btle
  - Mike Ossman, Dominic Spill, Mike Ryan & many others
- Adafruit_BLESniffer_Python
- Various others
  - SDR software, btlejack, radiobit, & more

# Section Summary

- Bluetooth is crazy!
- We will only be focusing on the least crazy today
  - Standard BT software
  - GATT
- In the next sections I will make some horribly untrue comparisons of GATT and HTTP

# 5 Minute Break

- Make sure you are following the setup instructions on https://github.com/hackgnar/ble_ctf/blob/master/docs/workshop_setup.md

- Either install NRF Connect on your phone or continue to get your Linux setup to work. If you are using Linux, hopefully you can run the following command by now:

```
rholeman@locallocal:~/src/bluez$ hciconfig -a
hci0:   Type: Primary  Bus: USB
        BD Address: 11:22:33:44:55:66  ACL MTU: 310:10  SCO MTU: 64:8
        UP RUNNING
        RX bytes:688 acl:0 sco:0 events:49 errors:0
        TX bytes:3163 acl:0 sco:0 commands:48 errors:0
        Features: 0xff 0xff 0x8f 0xfe 0xdb 0xff 0x5b 0x87
        Packet type: DM1 DM3 DM5 DH1 DH3 DH5 HV1 HV2 HV3
        Link policy: RSWITCH HOLD SNIFF PARK
        Link mode: SLAVE ACCEPT
        Name: 'locallocal #1'
        Class: 0x0c010c
        Service Classes: Rendering, Capturing
        Device Class: Computer, Laptop
        HCI Version: 4.0 (0x6)  Revision: 0x2031
        LMP Version: 4.0 (0x6)  Subversion: 0x2031
        Manufacturer: Cambridge Silicon Radio (10)
```

# What’s Next?

## Agenda

- Dirty GATT overview
- BLE CTF overview
- Tools primer

# GATT

**Lies**

- The HTTP of Bluetooth
- Think of a GATT server as a web site

**Technicalz**

- The most typical type of service hosted in BLE
- When you scan & connect to a BT device you are typically connecting to a GATT server
- Only one client can connect to a GATT server at a time
- Most do not require authentication & encryption
- Some will require auth and encryption access functionality

# GATT Characteristics

- The URLs of GATT
- They are represented by UUIDs on the GATT server
- Also denoted by handles in most Linux apps
- Characteristics come in 2 forms
  - **Predefined** by bluetooth standards - i.e. battery status, names, device types, etc
  - **Custom -** Custom code underneath that developers created specifically for their GATT application - i.e. change your riding mode on an electric skateboard
- Most devices typically host 5-10 characteristics\handles

# GATT Characteristics

HTTP URLS

| |
|---|
| /jobs/ |
| /jobs/job-id |
| /jobs/job-id/status |
| /jobs/job-id/files |
| /jobs/job-id/results |
| /jobs/job-id |
| /jobs/job-id/stop |

GATT Characteristics

| Handles | Service > Characteristics |
|---|---|
| 0001 -> 0005 | Generic Attribute ( 00001801-0000-1000-8000-00805f9b34fb ) |
| 0003 | Service Changed ( 00002a05-0000-1000-8000-00805f9b34fb ) |
| 0014 -> 001c | Generic Access ( 00001800-0000-1000-8000-00805f9b34fb ) |
| 0016 | Device Name ( 00002a00-0000-1000-8000-00805f9b34fb ) |
| 0018 | Appearance ( 00002a01-0000-1000-8000-00805f9b34fb ) |
| 001a | Central Address Resolution ( 00002aa6-0000-1000-8000-00805f9b34fb ) |
| 0028 -> ffff | 00ff ( 000000ff-0000-1000-8000-00805f9b34fb ) |
| 002a | ff01 ( 0000ff01-0000-1000-8000-00805f9b34fb ) |
| 002c | ff02 ( 0000ff02-0000-1000-8000-00805f9b34fb ) |
| 002e | ff03 ( 0000ff03-0000-1000-8000-00805f9b34fb ) |
| 0030 | ff04 ( 0000ff04-0000-1000-8000-00805f9b34fb ) |

# GATT Methods

- **Read**
  - The HTTP GET method of Bluetooth
  - curl http://google.com
  - `gatttool -b 11:22:33:44:55:66 --char-read -a 0x0011`
- **Write**
  - The HTTP POST method of Bluetooth
  - `curl -d "param1=value1&param2=value2" -X POST http://localhost:3000/data`
  - `gatttool -b 11:22:33:44:55:66 --char-write -a 0x0011 -n 0x1337`
- **Notify**
  - Streams data when you subscribe or listen to it
  - `gatttool -b 11:22:33:44:55:66 --char-read -a 0x0011 --listen`
- **Indicate**
  - Much like notify, but requires acks

# GATT Methods

- If you look at a method map for an HTTP server, it would looks something like

| Method | URL path | Description |
|---|---|---|
| **GET** | /jobs/ | List of the current user's jobs |
| **GET** | /jobs/job-id | Details for the specified job |
| **GET** | /jobs/job-id/status | Status code for the job (e.g. running) |
| **GET** | /jobs/job-id/files | List of links to job directory files |
| **GET** | /jobs/job-id/results | Results of the job |
| **DELETE** | /jobs/job-id | Release the job (terminate if still running) |
| **DELETE** | /jobs/job-id/stop | Stop the current job |

# GATT Methods

- If you look at a method map for a GATT server, it would looks something like

| Handles | Service > Characteristics | Properties |
|---|---|---|
| 0001 -> 0005 | Generic Attribute ( 00001801-0000-1000-8000-00805f9b34fb ) | |
| 0003 | Service Changed ( 00002a05-0000-1000-8000-00805f9b34fb ) | INDICATE |
| 0014 -> 001c | Generic Access ( 00001800-0000-1000-8000-00805f9b34fb ) | |
| 0016 | Device Name ( 00002a00-0000-1000-8000-00805f9b34fb ) | READ |
| 0018 | Appearance ( 00002a01-0000-1000-8000-00805f9b34fb ) | READ |
| 001a | Central Address Resolution ( 00002aa6-0000-1000-8000-00805f9b34fb ) | READ |
| 0028 -> ffff | 00ff ( 000000ff-0000-1000-8000-00805f9b34fb ) | |
| 002a | ff01 ( 0000ff01-0000-1000-8000-00805f9b34fb ) | READ |
| 002c | ff02 ( 0000ff02-0000-1000-8000-00805f9b34fb ) | READ **WRITE** |
| 002e | ff03 ( 0000ff03-0000-1000-8000-00805f9b34fb ) | READ |
| 0030 | ff04 ( 0000ff04-0000-1000-8000-00805f9b34fb ) | READ |
| 0032 | ff05 ( 0000ff05-0000-1000-8000-00805f9b34fb ) | READ **WRITE** |
| 0034 | ff06 ( 0000ff06-0000-1000-8000-00805f9b34fb ) | READ **WRITE** |
| 0036 | ff07 ( 0000ff07-0000-1000-8000-00805f9b34fb ) | READ **WRITE** |
| 0038 | ff08 ( 0000ff08-0000-1000-8000-00805f9b34fb ) | READ |
| 003a | ff09 ( 0000ff09-0000-1000-8000-00805f9b34fb ) | **WRITE** |
| 003c | ff0a ( 0000ff0a-0000-1000-8000-00805f9b34fb ) | READ **WRITE** |
| 003e | ff0b ( 0000ff0b-0000-1000-8000-00805f9b34fb ) | READ |
| 0040 | ff0c ( 0000ff0c-0000-1000-8000-00805f9b34fb ) | NOTIFY READ **WRITE** |
| 0042 | ff0d ( 0000ff0d-0000-1000-8000-00805f9b34fb ) | READ |
| 0044 | ff0e ( 0000ff0e-0000-1000-8000-00805f9b34fb ) | READ INDICATE **WRITE** |
| 0046 | ff0f ( 0000ff0f-0000-1000-8000-00805f9b34fb ) | NOTIFY READ **WRITE** |
| 0048 | ff10 ( 0000ff10-0000-1000-8000-00805f9b34fb ) | READ |

# Workshop

# BLE CTF

- You will be doing exercises on in a CTF to learn concepts we discussed today
- Built by yours truly
  - A series of BLE GATT exercises in CTF format
- Built on the ESP32
  - Super cheap microcontrollers
  - Nice C API
  - BLE, WiFi, USB stuff, Blinky LEDs
- Custom firmware

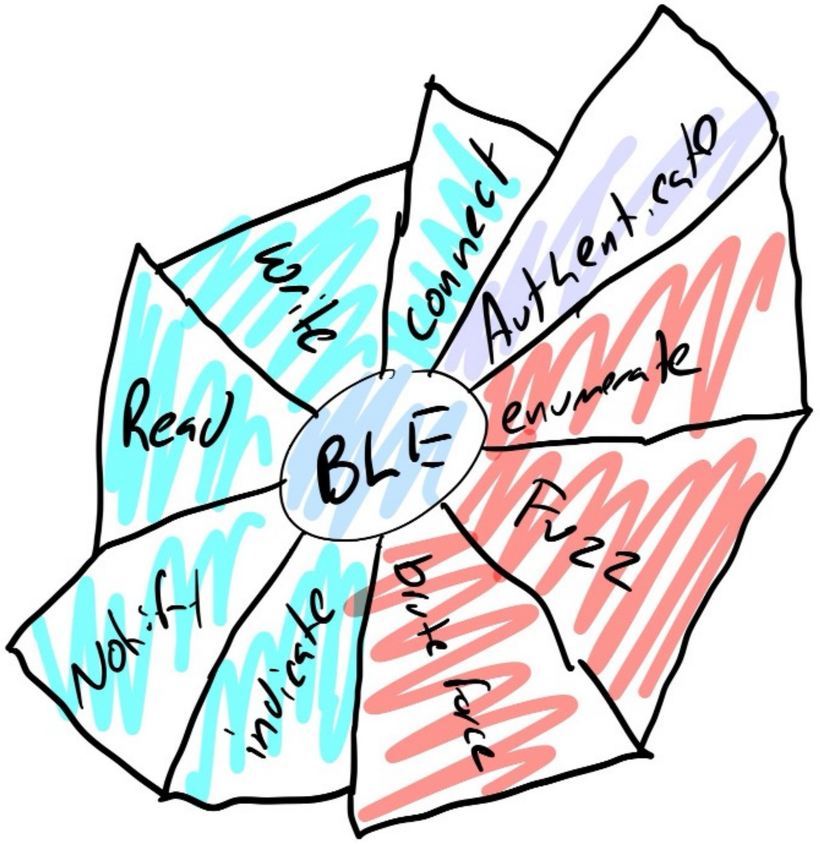
# BLE CTF

- Why did I build this???
- There were no great resources for learning BLE
- Low cost of entry
- Get more people involved with BLE
- I had never written GATT servers before and wanted to try

# BLE CTF

## Version 1

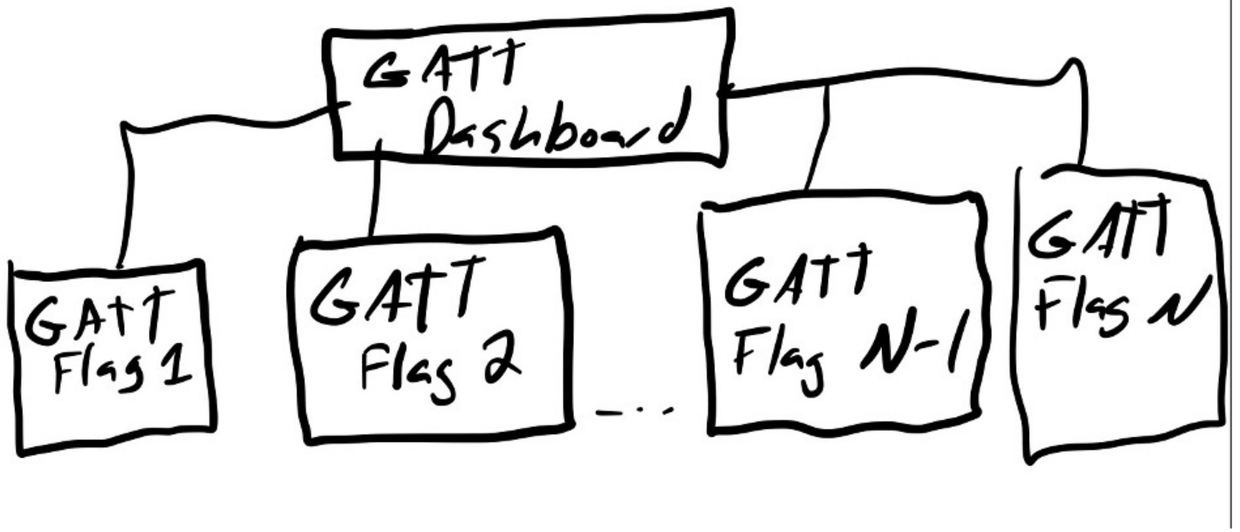
- What we are using today!
- Released last year
- Teaches the basics of BLE
- No Bluetooth experience required
- Only requires a Linux box and standard Bluetooth connectivity
- Very monolithic in nature
- Has like 30ish characteristics
- Firmware is not very modular
- Does not allow for more advanced challenges



# BLE CTF

## Version 2 - Infinity

- Advanced BLE concepts
- Modular
- Firmware created by code generatic
- Each flag is a different GATT server
- Design allows for more complex connection and authentication challenges
- Persistent state storage



# BLE CTF

## Fun facts

- There is a fantasy RPG port of BLE CTF created by Stephen Stockman called BLECTF_FUN
- There is a chapter in the book Practical IoT Hacking about BLE CTF Infinity
- At DEF CON 31 there was a hacker group badge add-on based on BLE CTF
- Various conference workshops use the project
- Claude and ChatGPT can be used to help solve most of the exercises

# Tools

## Things you will need for today

- Linux box or NRF Connect on your phone
- For Linux users, you will need:
  - a compatible bluetooth module in your computer or a USB dongle
  - Bluetooth software
    - Gatttool or bluetoothCTL
    - Hcitool
  - Bash Commands
    - Xdd
    - Echo
    - Md5sum
    - Tr
    - For loops

# Tools

## Linux Setup

- If you run the following and see an hcix device, you are gtg
  - `hciconfig -a`
- If you are on kali you will have to undo rfkill
  - `rfkill unblock all`
  - `hciconfig hci0 up`
- If you are on Windows or OSX
  - Install an Ubuntu or Kali VM and pass through a bluetooth device

# Tools

## Hcitool

- Hcitool is great for scanning for connectable devices
- You will typically use the following to scan for BLE
  - `hcitool lescan`
- Some versions of hcitool dedup results, some dont
- For versions that don’t it’s useful to pipe results though grep if you know a BT mac address or BT device name
  - `hcitool lescan |grep –i ctf`
- We will be supplying you with MAC addresses to use so you don’t need to scan to find your device
- If Bluez tools are not on your Linux system, you can scan for devices with bluetoothctf or btmgnt

# Tools

## Hcitool

```
rholeman@locallocal:~/src/ble_ctf$ sudo hcitool lescan
LE Scan ...
24:C4:3C:90:A5:5F (unknown)
32:50:C1:3A:C5:C6 (unknown)
80:7D:3A:C4:1C:8A BLE_CTF_SCORE
80:7D:3A:C4:1C:8A BLE_CTF_SCORE
80:7D:3A:C4:1C:8A (unknown)
7A:10:46:C3:1C:48 (unknown)
60:FD:C2:7B:99:2A (unknown)
60:FD:C2:7B:99:2A (unknown)
80:7D:3A:C4:1C:8A BLE_CTF_SCORE
32:50:C1:3A:C5:C6 (unknown)
```

# Tools

## Gatttool

- The OG
- Is now deprecated in most modern Linux distributions
- If you want to use it, try Ubuntu releases 24.04 or earlier
- Tried and true for over a decade
- Easy to script
- Aligns with the BLE CTF documentation
- 1000s of users have completed the full CTF with this
- Doesn’t have an enumeration function but you can use a script to do this that I provide in the BLE CTF repository called gatttool_enum.sh
  - https://github.com/hackgnar/ble_ctf/blob/master/gatttool_enum.sh

# Tools

## Gatttool

- gatttool is great for connecting to GATT servers to enumerate characteristics, do read, do writes, etc
- To **list characteristics**/handles of a GATT server
  - `gatttool -b 11:22:33:44:55:66 --characteristics`
- To **read** a characteristic/handle value
  - `gatttool -b 11:22:33:44:55:66 --char-read -a 0x0011`
- To **write** a characteristic/handle value
  - `gatttool -b 11:22:33:44:55:66 --char-write -a 0x0011 -n 0x1337`
- You can also do **persistent connections** to a GATT server
  - `gatttool -b 11:22:33:44:55:66 -I`
- --help-all is your friend
  - `gatttool --help-all`

# Tools

## BluetoothCTL

- The modern linux replacement for gatttool
- Doesn’t have an enumeration function
- Have to write to attributes/characteristics/handles with UUIDs which can be lengthy
- You can theoretically do all of the CTF with this but if you are using interactive mode, you will have issues with hex and ascii conversion on writes
- The read methods do have auto conversations functionality for reading in hex and ascii
- Do keep in mind that this is a persistent connection tool and you will not be able to connect back to the dongle with another tool while connected with this tool
- LLMs can help you convert gatttool commands to associated bluetoothctl commands

# Tools

## Bluetoothctl

- Can’t one shot commands like gatttool
- To **list characteristics**/handles of a GATT server
  - `bluetoothctl`
  - `scan le`
  - `connect 11:22:33:44:55:66`
  - `gatt.list-attributes`
- To **read** a characteristic/handle value
  - `gatt.select-attribute UUID`
  - `gatt.read`
- To **write** a characteristic/handle value
  - `gatt.write “value”`
- help is your friend
  - `command –help`

# Tools

## Bash commands

- **Xdd**
  - Useful for converting hex => ascii and vise versa in gatttool
  - For hex to ascii use `xxd -r -p`
  - For ascii to hex use `xxd -ps`
  - `gatttool -b de:ad:be:ef:be:f1 --char-read -a 0x002a|awk -F':' '{print $2}'|tr -d ' '|xxd -r -p;printf '\n'`
- **Echo**
  - Nothing crazy here, just remember that the -n flag strips newlines. This is useful for sending flag values
- **Tr, awk** & for loops
  - Nothing crazy here either… just useful for managing strings and connection loops

# Tools

## NRF Connect

- Its a phone app, just fire it up and use it
- The fallback for people with linux problems
- This is an iPhone or Android application you can use for interacting with generic bluetooth devices
- You can do about 70% of the CTF with this but will require some tedious repudiative tasks for some of the exercises
- This is a great tool to use if you hare having issues with a flag
- Do keep in mind that this is a persistent connection tool and you will not be able to connect back to the dongle with another tool while connected with this tool

# Tools

## Optional - WHAD

- This tool is amazing
- It is my favorite tool in existence and is far superior to everything out there
- Requires the most setup and has a steeper learning curve
- Can be used with multiple hardware devices such as standard HCI BT devices, NRF chips, ubertooth, and even ESP32s
- Is lacking a full enumeration capability to read the values of all characteristics/handles
- Requires you to use linux based handle numbers like in gatttool or BLEH.  The index of these may not align with those from gatttool or the BLE CTF documentation
- You can theoretically do all of the exercises with this but I have found some inconsistencies with ascii and hex based writes.  TBD if this is a bug in WHAD or in the firmware of the CTF
- If you use an NRF dongle, you can play around with passive packet scanning, and even injection which presents the most full featured experience of learning to hack BLE
- Do keep in mind that this is a persistent connection tool and you will not be able to connect back to the dongle with another tool while connected with this tool

# Tools

## Optional - Bettercap

- Has a nice enumeration feature to see all of the contents of read characteristics on a device
- Has limited ble method functionality such as write, read, notification and indication
- You could theoretically do 25-50% of the excercises with it
- Mostly nice for device scanning
- Do keep in mind that this is a persistent connection tool and you will not be able to connect back to the dongle with another tool while connected with this tool

# Flag 1

Flag one is a gift! You can only obtain it by reading this document or peaking at the source code. In short, this flag is to get you familiar with doing a simple write to a BLE handle. Do the following to get your first flag. Make sure you replace the MAC address in the examples below with your devices Mac address!

First, check out your score:

```
gatttool -b de:ad:be:ef:be:f1 --char-read -a 0x002a|awk -F':' '{print $2}'|
tr -d ' '|xxd -r -p;printf '\n'
```

Next, lets submit the following flag.

```
gatttool -b de:ad:be:ef:be:f1 -char-write-req -a 0x002c -n $(echo -n
"12345678901234567890"|xxd -ps)
```

Finally, check out your score again to see your flag got accepted:

```
gatttool -b de:ad:be:ef:be:f1 --char-read -a 0x002a|awk -F':' '{print $2}'|
tr -d ' '|xxd -r -p;printf '\n'
```

# Flag 1

## Bluetoothctl

```
bluetooth ctl
scan le
scan off
connect YO:UR:MA:CA:DD:SS
gatt.list-attributes
gatt.select-attribute 0000ff02-0000-1000-8000-00805f9b34fb
gatt.read
gatt.write "0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38 0x39 0x30 0x31 0x32 0x33 0x34
0x35 0x36 0x37 0x38 0x39 0x30"
gatt.select-attrubute 0000ff01-0000-1000-8000-00805f9b34fb
gatt.read
```

# Flag 2

## Bluetoothctl

# Extra Credit

## Sniffers

If you have an ubertooth or nordic sniffer, try sniffing your connections as you work the exercises

```
sudo ubertooth-btle -tDE:AD:BE:EF:12:34

sudo ubertooth-btle -f -r bt.pcap
```

Read your pcaps with tshark or wireshark

```
tshark -r bt.pcap -x -V
```

Look for read or write values that went clear text over the wire

```
tshark -r bt.pcap -x -V -Y 'btatt.opcode == 0x0b’
```