BLE Workshop

By: Ryan Holeman

Question

- How many of you here have done my BLE CTF?

Prep Work

- Make sure you have an ESP32 and micro USB cable
- Get your system setup while I talk
  - https://github.com/hackgnar/ble_ctf_infinity/blob/master/docs/workshop_setup.md
- Feel free to start the exercises while I talk
  - Beginners doing BLE_CTF v1
    - https://github.com/hackgnar/ble_ctf
  - Advanced users doing BLE_CTF_INFINITY
    - https://github.com/hackgnar/ble_ctf_infinity
- Please only connect to your MAC address
- I have some bluetooth USB dongles incase you have issues with yours

Honor System

- Please return your BLE_CTF chips and USB cables when you leave
- They cost money. I don’t get them for free =)
- If you want to take one home with you, I sell them for $20

Ryan Holeman

- Atlassian - Manager
  - Incident Response Team
  - Detection Team
  - Red Team
- Ziften - Advisor
- Likes skateboarding
- Master Degree from a past life
- Speak at various conferences
- AHA junkie
- Twitterz: @hackgnar

Agenda

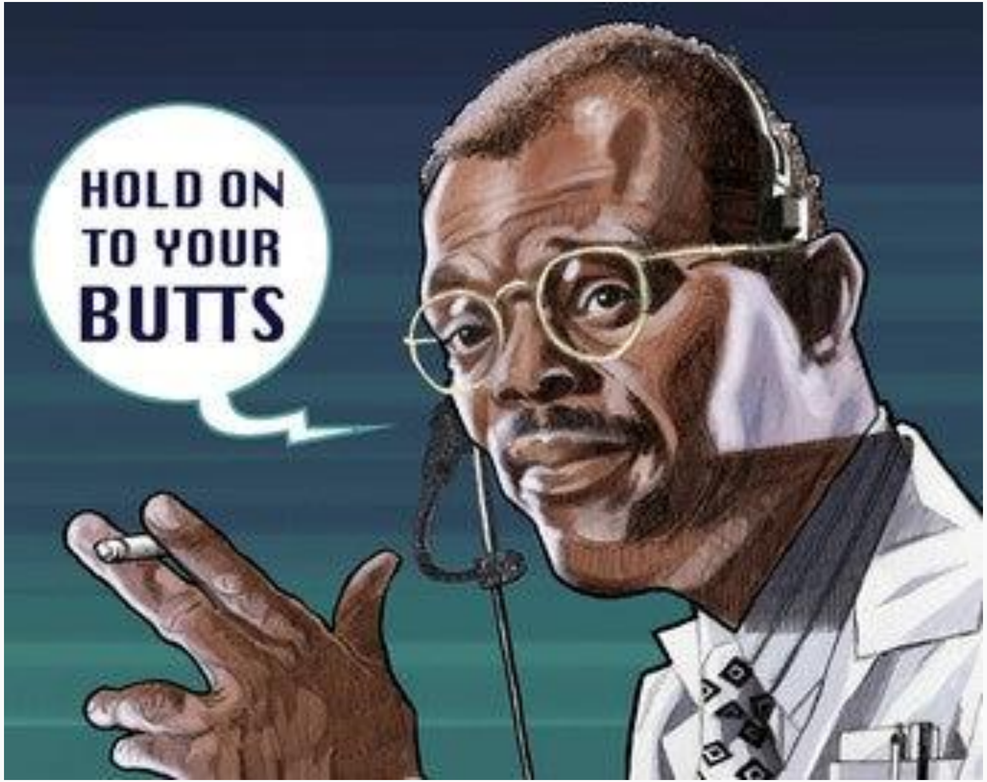
- BLE basics
  - Protocols
  - Stacks
  - Hardware
  - Software
- Workshop essentials
  - GATT
  - BLE CTF
  - Tools
- Training
  - +20 exercises

Warning!!!

- Many things I say during this class will not be 100% accurate
- I use a lot of analogies and comparisons to accelerate your understanding
- **Be responsible with what you learn!**

Hold On!

- 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

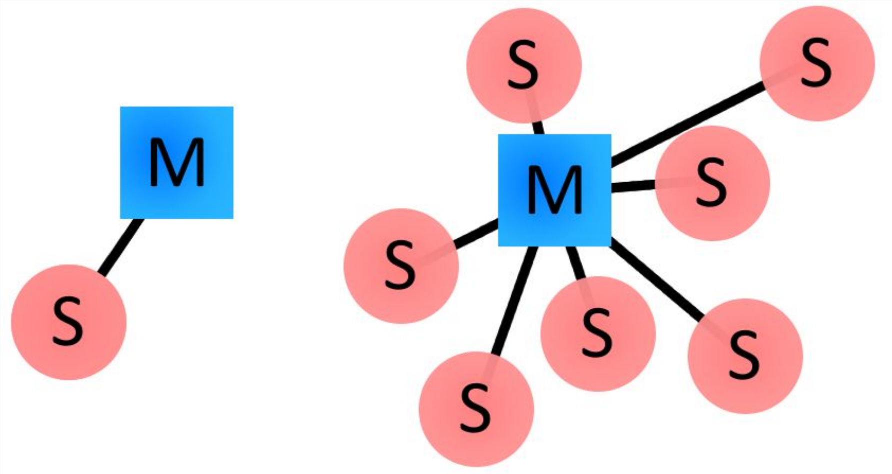


Bluetooth - BLE vs Basic Rate

- BLE (aka Smart, 4.0)
  - 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 3-5 years ago
  - Harder to sniff and discover
  - Still in use today
    - Devices with bigger batteries
    - Keyboards, cars, etc

Client Server Topology

- Master
    - i.e. your computer or phone
- Slave
    - i.e. your watch, earphones, mouse, keyboard, heart rate monitor, etc

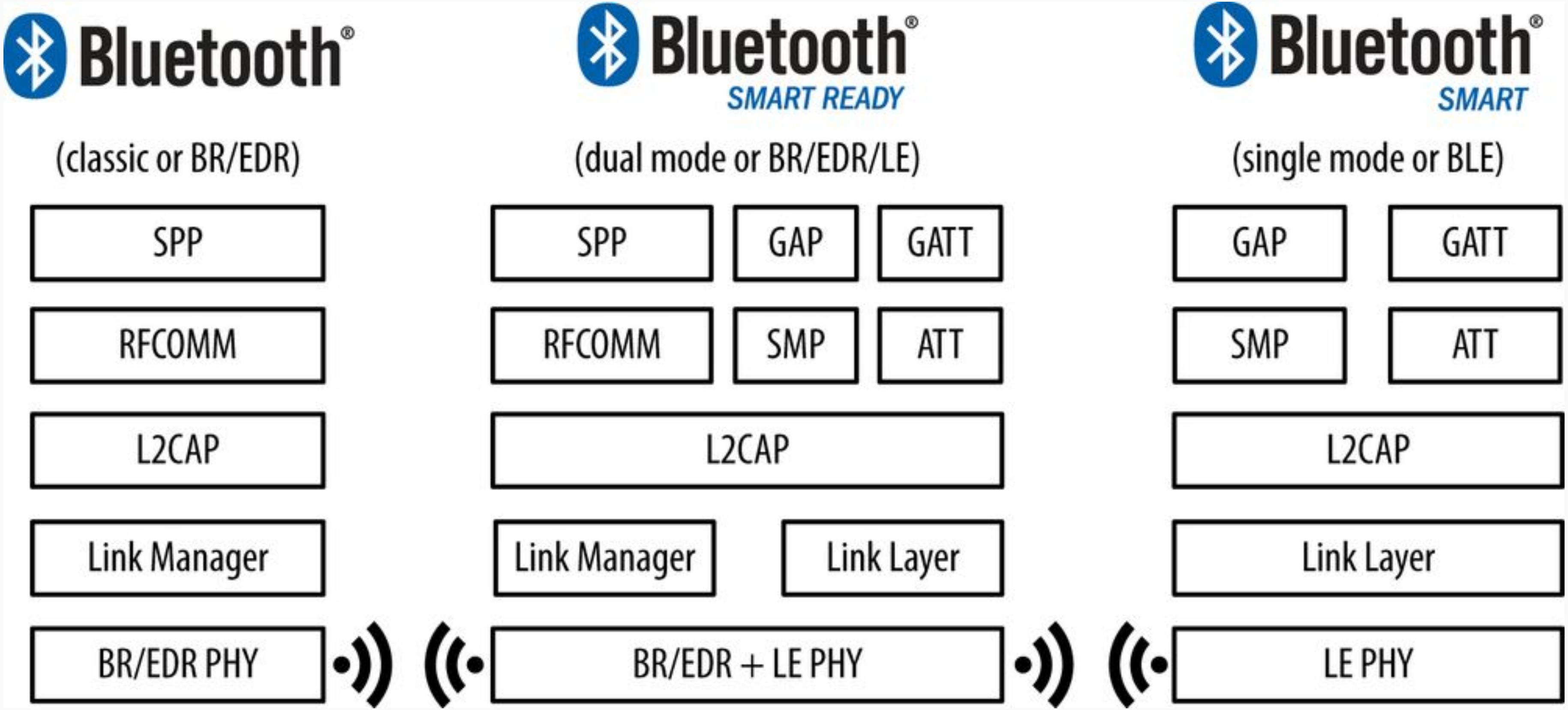


Connection Types

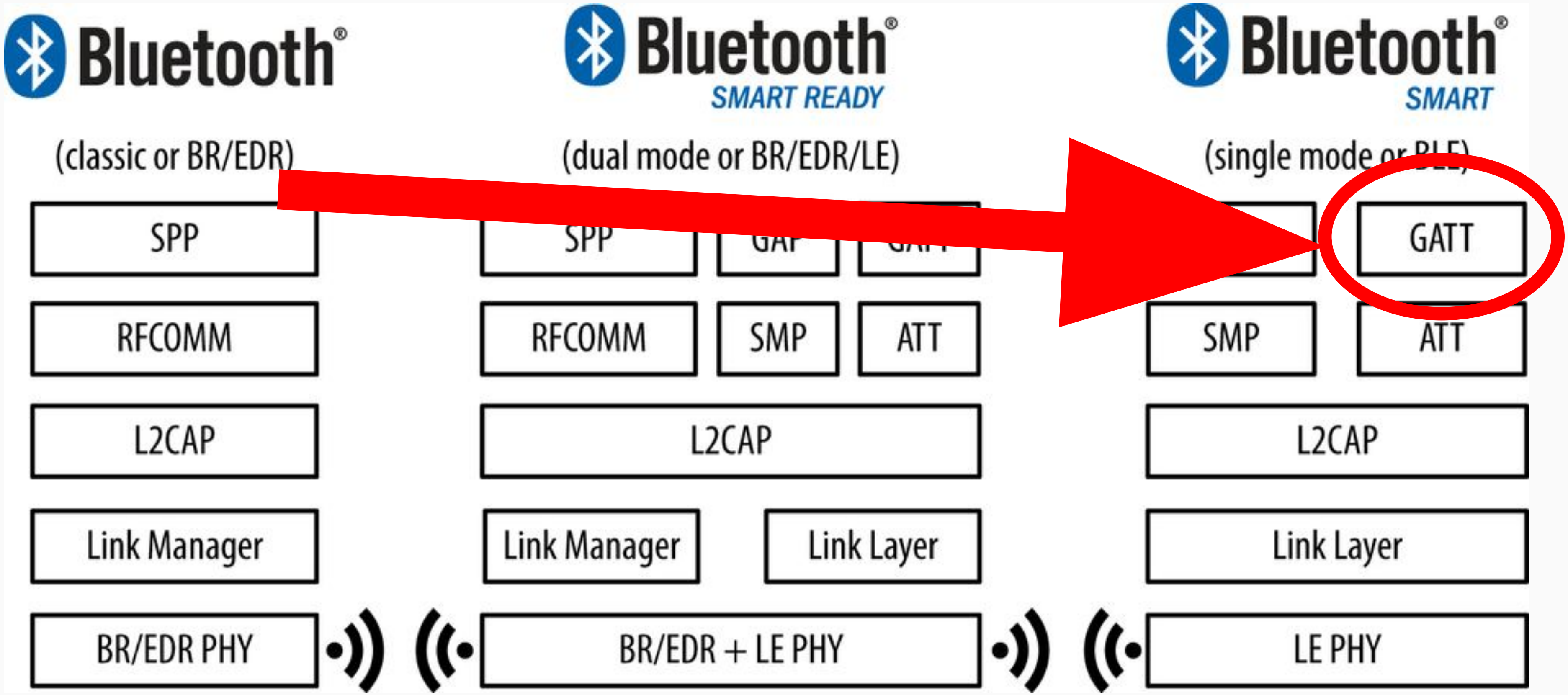
- Paired vs Unpaired
- Authentication
    - In band
    - Out of band
- Encryption

- 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

Hardware

- What's it all do???
- What do I need?

Hardware - Categories

- Standard Bluetooth modules
    - Your computer bluetooth chip
    - UD100
    - Bluetooth usb dongles
- IOT devices
    - Esp32
    - Microbit
    - Nordic based devices
- Sniffers
    - Ubertooth
    - Hackrf & other SDR devices
- Hybrids
    - Nordic chips (Microbit, Adafruit sniffer, etc)

Hardware - Standard Bluetooth Modules

- 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

Software - Standard Bluetooth Modules

- hciconfig
  - Basically ifconfig for bluetooth interfaces
  - `sudo hciconfig -a`
- hcitool
  - Kind of like iwlist for bluetooth interfaces
  - `sudo hcitool lescan`
- gatttool
  - Kind of like curl for bluetooth
  - `sudo gatttool -i hci0 -b DE:AD:BE:EF:12:34 --characteristics`
- bleah
  - A pretty printed display of GATT characteristics
  - `sudo bleah -b DE:AD:BE:EF:12:34 -e`

Hardware - Sniffers

- Allow you to passively sniff bluetooth traffic
- Can be used for various types of injection or BT protocol simulation
- Can not be used as typical Bluetooth host OS devices
- Operate at the PHY layer
- Require custom firmware & host software
- Come in various different flavors
- **Not needed for this workshop**

Software - Sniffers

- ubertooth-btle
    - Ubertooth host software for interacting with your ubertooth
    - sudo ubertooth-btle -tDE:AD:BE:EF:12:34
    - sudo ubertooth-btle -f -r bt.pcap
- Adafruit_BLESniffer_Python
    - Cool curses method for creating pcaps
    - sudo python sniffer.py /dev/ttyUSB0
- btlejack
    - Sniffer and injector tool and firmware for microbit
    - Cool stuff… haven’t played with it too much yet
- Various SDR tools & libs

Hardware - BT Devices, IOT Devices & Hybrids

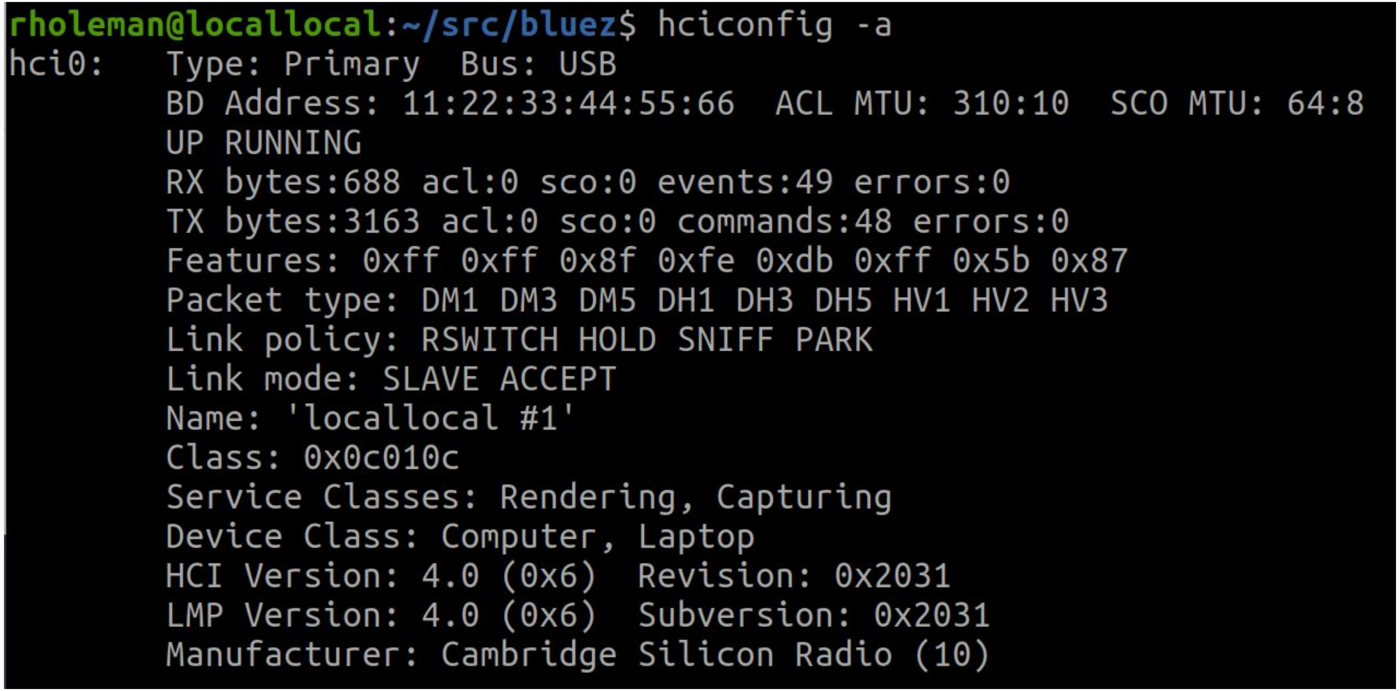
- Can host firmware to act as standard BT clients or servers
- Some can be used as sniffers
  - Nordic 4x & 5x based chipsets
- Firmware libraries depend on chipsets
- Capabilities vary based on firmware api support
- Mostly all C code based

Let’s Step Back

- Bluetooth is crazy!
- We will only be focusing on the least crazy today
  - Standard BT software
  - GATT
- In Bluetooth land you can think of GATT kind of like HTTP in network land
- Lets make some horribly untrue comparisons of GATT and HTTP

5 MINUTE BREAK

- Try running the following on your computer:



```
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)
```

Agenda - What’s Next?

- Dirty GATT overview
- BLE CTF overview
- Tools primer

GATT

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

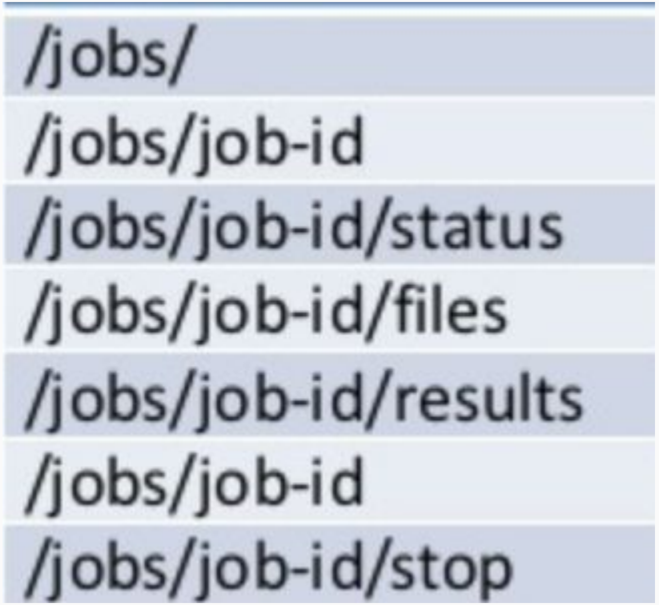
- Technicalz
  - The most typical type of service hosted in BLE
  - When you scan for BLE and connect you are most always connecting to a GATT server
  - Only one client can connect to a GATT server at a time
  - Most do not require authentication & encryption
  - Some will require auth and encryption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

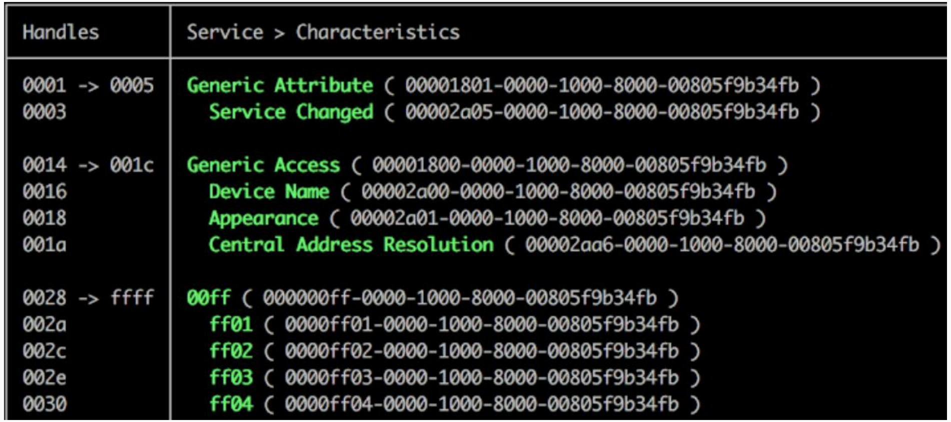
GATT Characteristics

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

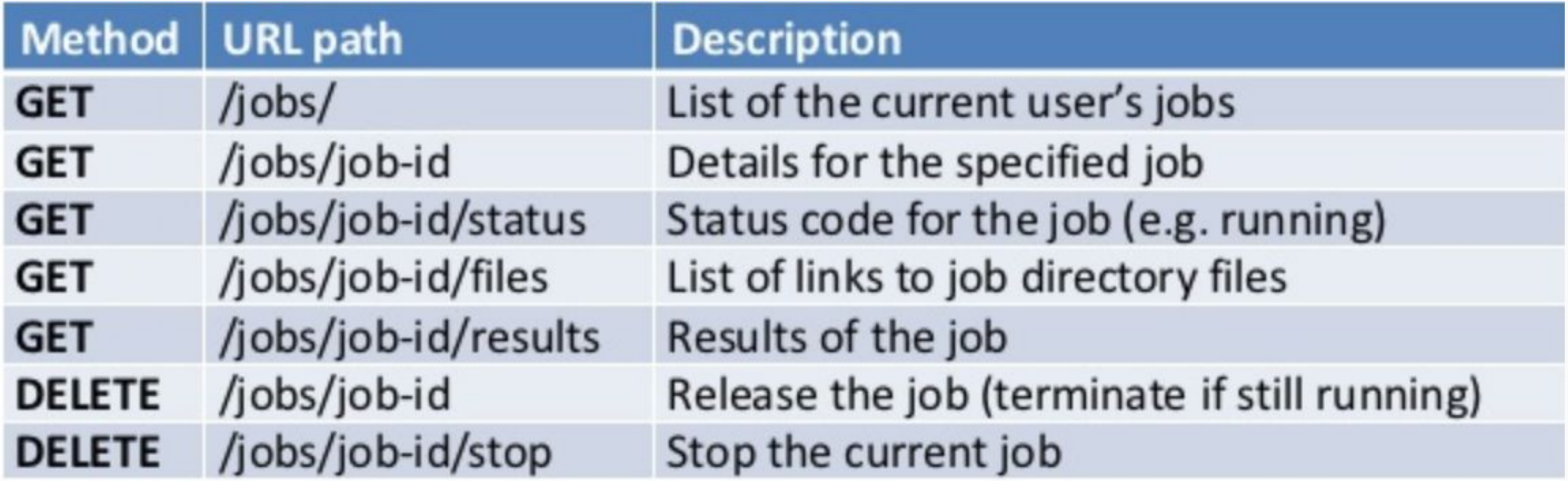


```
Handles        Service > Characteristics

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

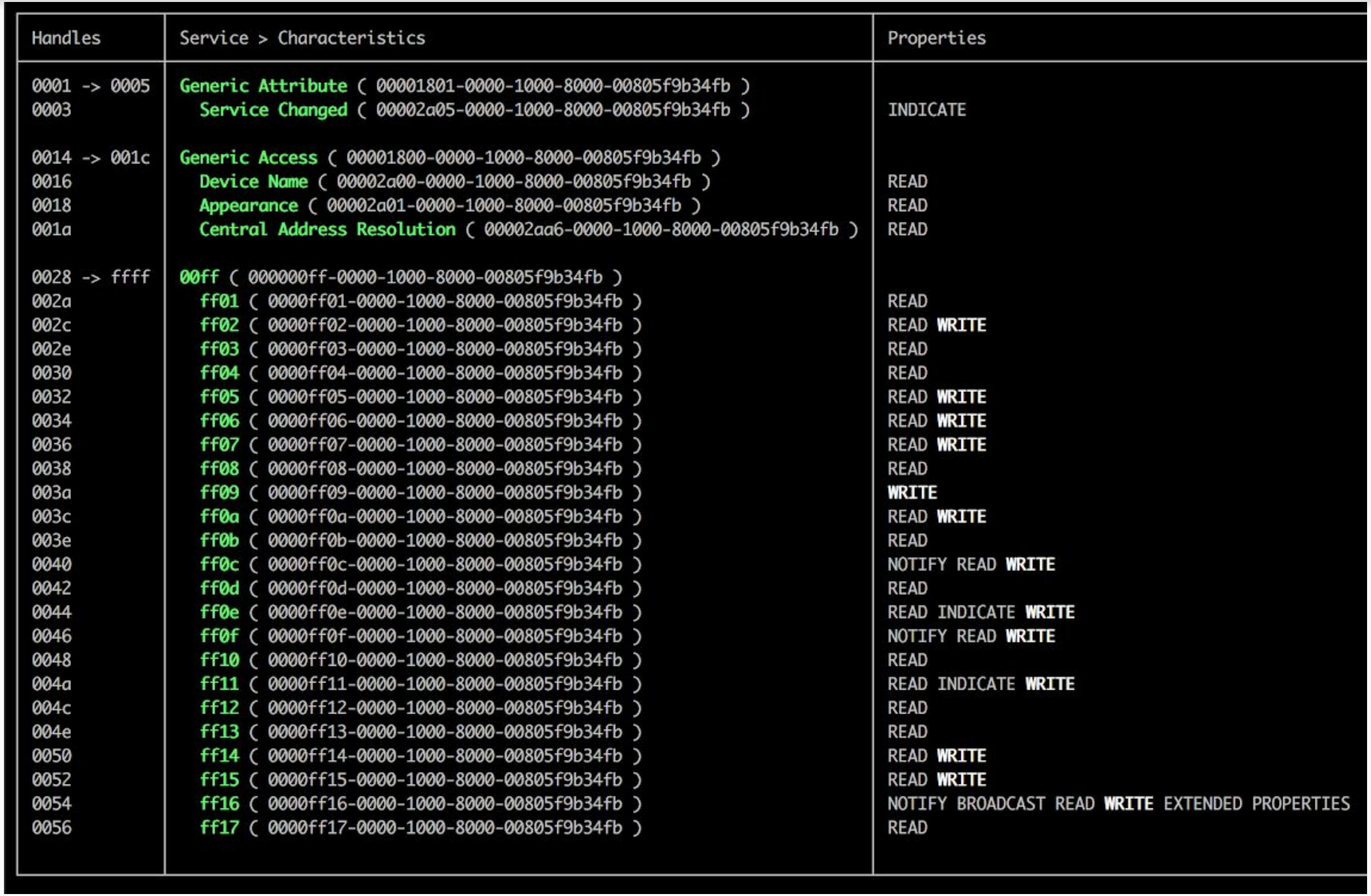
- Read
  - The HTTP GET method of Bluetooth
  - `curl http://google.com`
  - `gatttool -b 11:22:33:44:55:66 --char-read -a 0x0011`
- Write
  - The HTTP POST method of Bluetooth
  - `curl -d "param1=value1&param2=value2" -X POST http://localhost:3000/data`
  - `gatttool -b 11:22:33:44:55:66 --char-write -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



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



```
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
004a             ff11 ( 0000ff11-0000-1000-8000-00805f9b34fb )                         READ INDICATE WRITE
004c             ff12 ( 0000ff12-0000-1000-8000-00805f9b34fb )                         READ
004e             ff13 ( 0000ff13-0000-1000-8000-00805f9b34fb )                         READ
0050             ff14 ( 0000ff14-0000-1000-8000-00805f9b34fb )                         READ WRITE
0052             ff15 ( 0000ff15-0000-1000-8000-00805f9b34fb )                         READ WRITE
0054             ff16 ( 0000ff16-0000-1000-8000-00805f9b34fb )                         NOTIFY BROADCAST READ WRITE EXTENDED PROPERTIES
0056             ff17 ( 0000ff17-0000-1000-8000-00805f9b34fb )                         READ
```

BLE CTF

- 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

- Comes in 2 versions
- Version 1 - BLE_CTF
  - **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 to use
  - Very monolithic in nature
  - Has like 30ish characteristics
  - Firmware is not very modular
  - Does not allow for more advanced challenges
  - No persistence

BLE CTF

- Comes in 2 versions
- Version 2 - BLE_CTF_INFINITY
  - Extremely modular
    - People can contribute new flags
  - Hosts 20 stand alone GATT servers on one chip
    - WHAT! How you do that?
  - Authentication based challenges
  - Encryption based challenges
  - Client/server based challenges
  - Dirty dirty dirty GATT tricks
  - Dynamically include flags values and challenges
  - Persistent on reboot

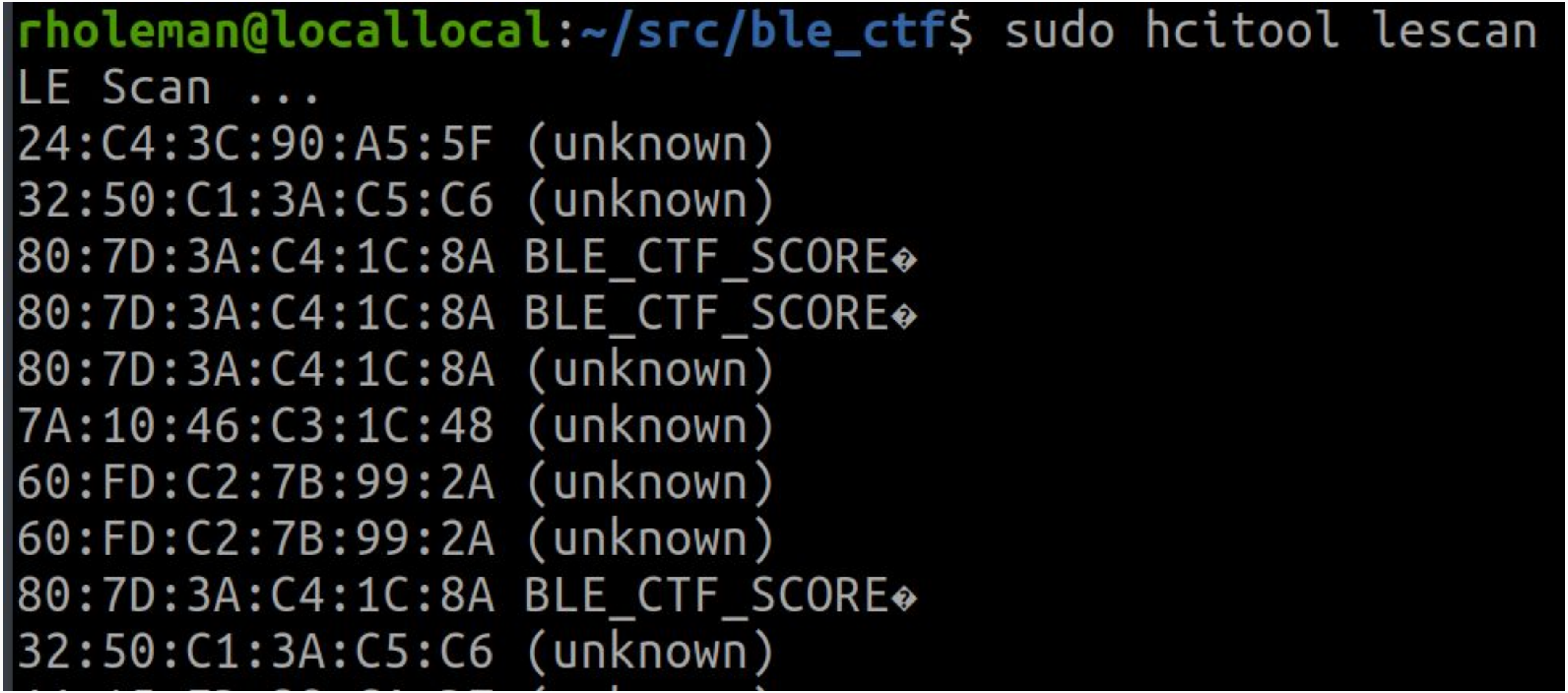
Tools

- Linux box
  - Or Vagrant on OSX/Windows
- Bluetooth module in your computer or a USB dongle
- Bluetooth software
  - Gatttool
  - Hcitool
  - Bleah - optional
  - bluetoothctl
- Bash Commands
  - Xdd
  - Echo
  - Md5sum
  - Tr
  - For loops

Tools - Software - 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`

Tools - Software - 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)
```

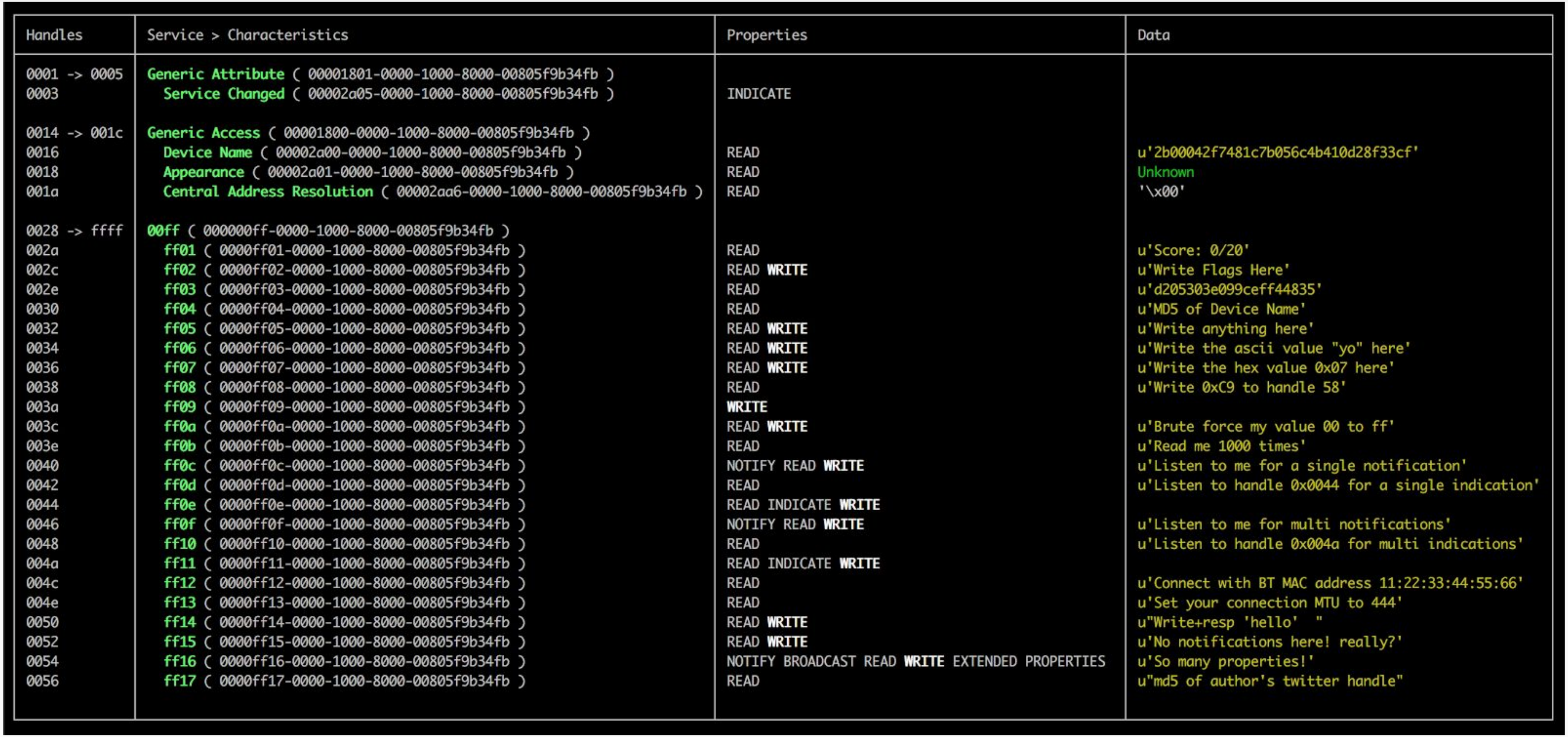
Tools - Software - gatttool

- 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-alll

Tools - Software - bleah

- Bleah is a great visualization tool for BLE
  - Created by @evilsocket
- Provides functionality to scan BLE devices like hcitool
- Provides functionality to do mass reads across all characteristics/handles
- It also provides functionality to read/write like gatttool with the addition of ascii support
- Recently deprecated but still available via forks
- Totally optional for this workshop
- Easier to install with python2
  - With python3 you will have to edit some code here and there

Tools - Software - bleah



```
Handles       Service > Characteristics                                 Properties                                      Data

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                                            u'2b00042f7481c7b056c4b410d28f33cf'
0018            Appearance ( 00002a01-0000-1000-8000-00805f9b34fb )     READ                                            Unknown
001a            Central Address Resolution ( 00002aa6-0000-1000-8000-00805f9b34fb )  READ                               '\x00'

0028 -> ffff  00ff ( 000000ff-0000-1000-8000-00805f9b34fb )
002a            ff01 ( 0000ff01-0000-1000-8000-00805f9b34fb )           READ                                            u'Score: 0/20'
002c            ff02 ( 0000ff02-0000-1000-8000-00805f9b34fb )           READ WRITE                                      u'Write Flags Here'
002e            ff03 ( 0000ff03-0000-1000-8000-00805f9b34fb )           READ                                            u'd205303e099ceff44835'
0030            ff04 ( 0000ff04-0000-1000-8000-00805f9b34fb )           READ                                            u'MD5 of Device Name'
0032            ff05 ( 0000ff05-0000-1000-8000-00805f9b34fb )           READ WRITE                                      u'Write anything here'
0034            ff06 ( 0000ff06-0000-1000-8000-00805f9b34fb )           READ WRITE                                      u'Write the ascii value "yo" here'
0036            ff07 ( 0000ff07-0000-1000-8000-00805f9b34fb )           READ WRITE                                      u'Write the hex value 0x07 here'
0038            ff08 ( 0000ff08-0000-1000-8000-00805f9b34fb )           READ                                            u'Write 0xC9 to handle 58'
003a            ff09 ( 0000ff09-0000-1000-8000-00805f9b34fb )           WRITE
003c            ff0a ( 0000ff0a-0000-1000-8000-00805f9b34fb )           READ WRITE                                      u'Brute force my value 00 to ff'
003e            ff0b ( 0000ff0b-0000-1000-8000-00805f9b34fb )           READ                                            u'Read me 1000 times'
0040            ff0c ( 0000ff0c-0000-1000-8000-00805f9b34fb )           NOTIFY READ WRITE                               u'Listen to me for a single notification'
0042            ff0d ( 0000ff0d-0000-1000-8000-00805f9b34fb )           READ                                            u'Listen to handle 0x0044 for a single indication'
0044            ff0e ( 0000ff0e-0000-1000-8000-00805f9b34fb )           READ INDICATE WRITE
0046            ff0f ( 0000ff0f-0000-1000-8000-00805f9b34fb )           NOTIFY READ WRITE                               u'Listen to me for multi notifications'
0048            ff10 ( 0000ff10-0000-1000-8000-00805f9b34fb )           READ                                            u'Listen to handle 0x004a for multi indications'
004a            ff11 ( 0000ff11-0000-1000-8000-00805f9b34fb )           READ INDICATE WRITE
004c            ff12 ( 0000ff12-0000-1000-8000-00805f9b34fb )           READ                                            u'Connect with BT MAC address 11:22:33:44:55:66'
004e            ff13 ( 0000ff13-0000-1000-8000-00805f9b34fb )           READ                                            u'Set your connection MTU to 444'
0050            ff14 ( 0000ff14-0000-1000-8000-00805f9b34fb )           READ WRITE                                      u"Write+resp 'hello'  "
0052            ff15 ( 0000ff15-0000-1000-8000-00805f9b34fb )           READ WRITE                                      u'No notifications here! really?'
0054            ff16 ( 0000ff16-0000-1000-8000-00805f9b34fb )           NOTIFY BROADCAST READ WRITE EXTENDED PROPERTIES u'So many properties!'
0056            ff17 ( 0000ff17-0000-1000-8000-00805f9b34fb )           READ                                            u"md5 of author's twitter handle"
```

Tools - Software - 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

15 MINUTE BREAK

- Make sure you have your computer setup

Flag 1

github.com/hackgnar/ble_ctf/doc

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'
```

Extra Credit

- 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’`

Contributions

- BLE CTF Infinity is very modular
- I wrote some docs on how to write your own flags:
  - https://github.com/hackgnar/ble_ctf_infinity/blob/master/docs/contributing.md
- Feel free to ask me to create specific challenges, but ultimately, it’s a lot more rad if you do it yourself =)

Fin

Ryan Holeman

- @hackgnar
- github/hackgnar