BLE Workshop

By: Ryan Holeman

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

ATLASSIAN

AHA

ziften

Agenda

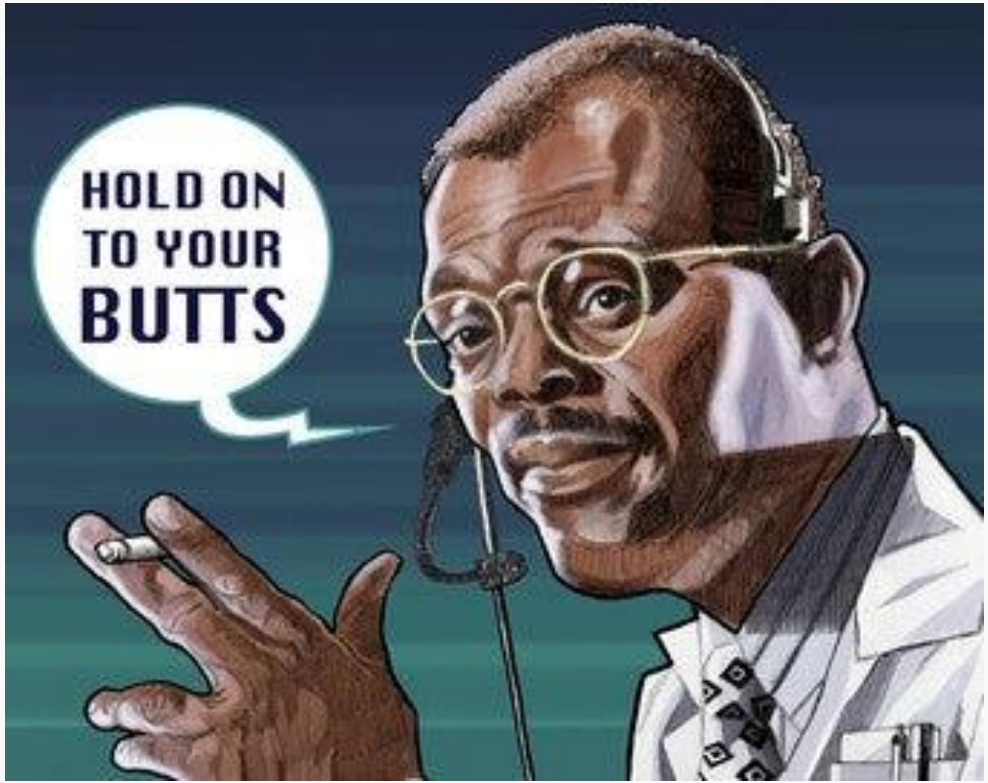
- BLE basics
  - Protocols
  - Stacks
  - Hardware
  - Software
- Workshop essentials
  - GATT
  - BLE CTF
  - Tools
- Training
  - +20 exercises

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!**

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

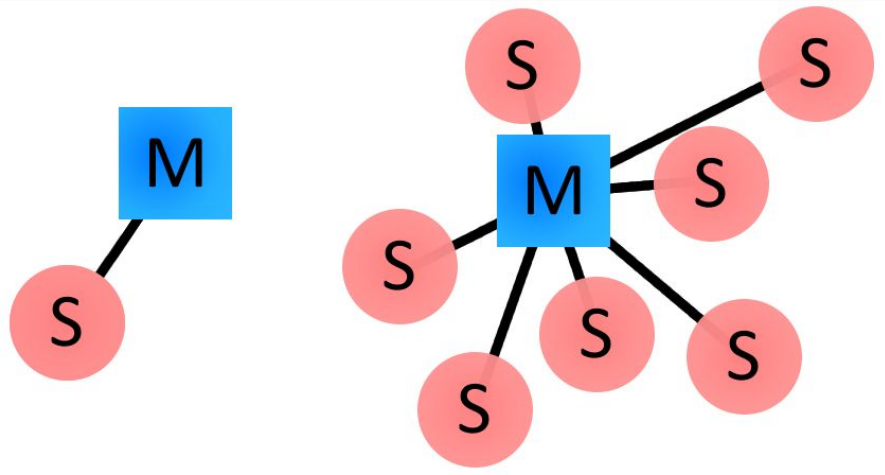


Bluetooth - BLE vs Basic Rate

- 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

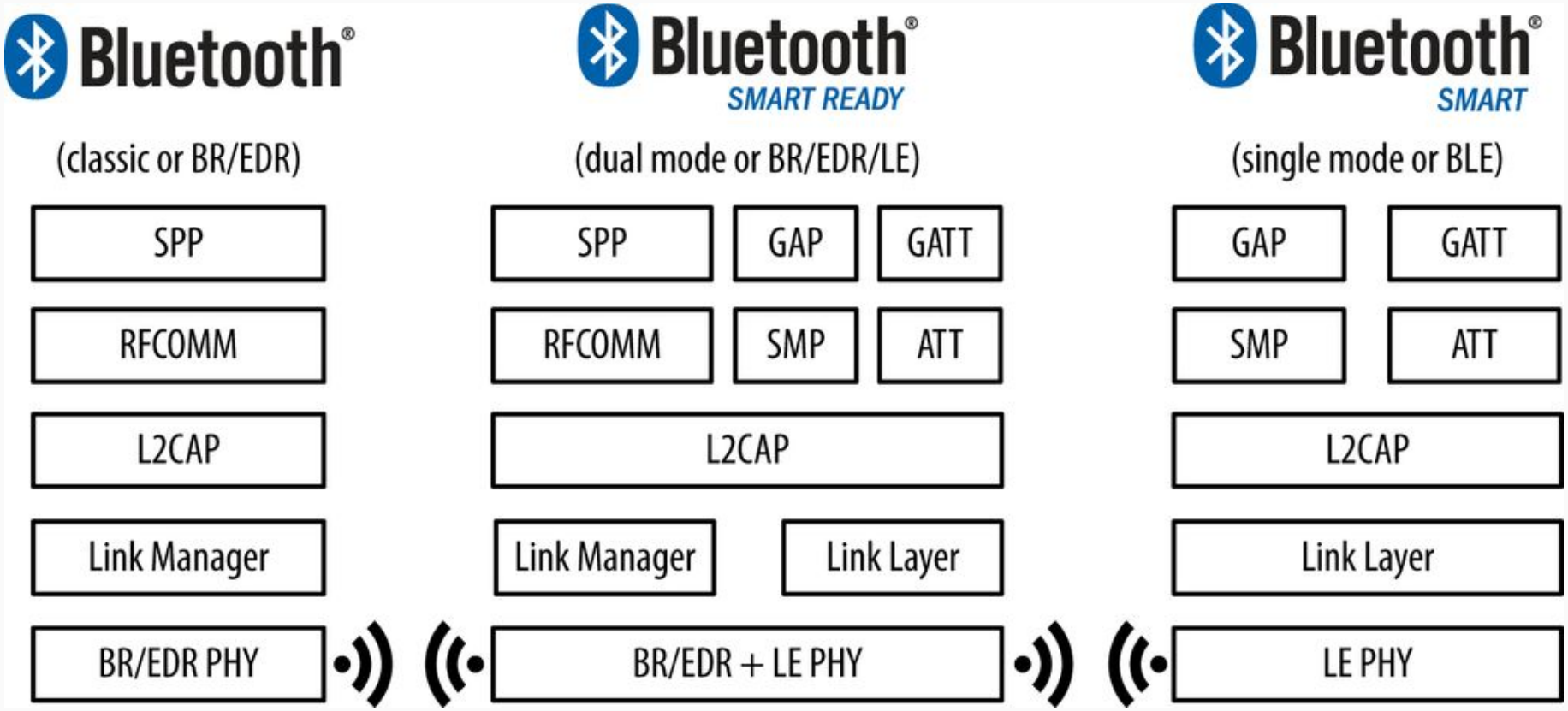


Connection Types

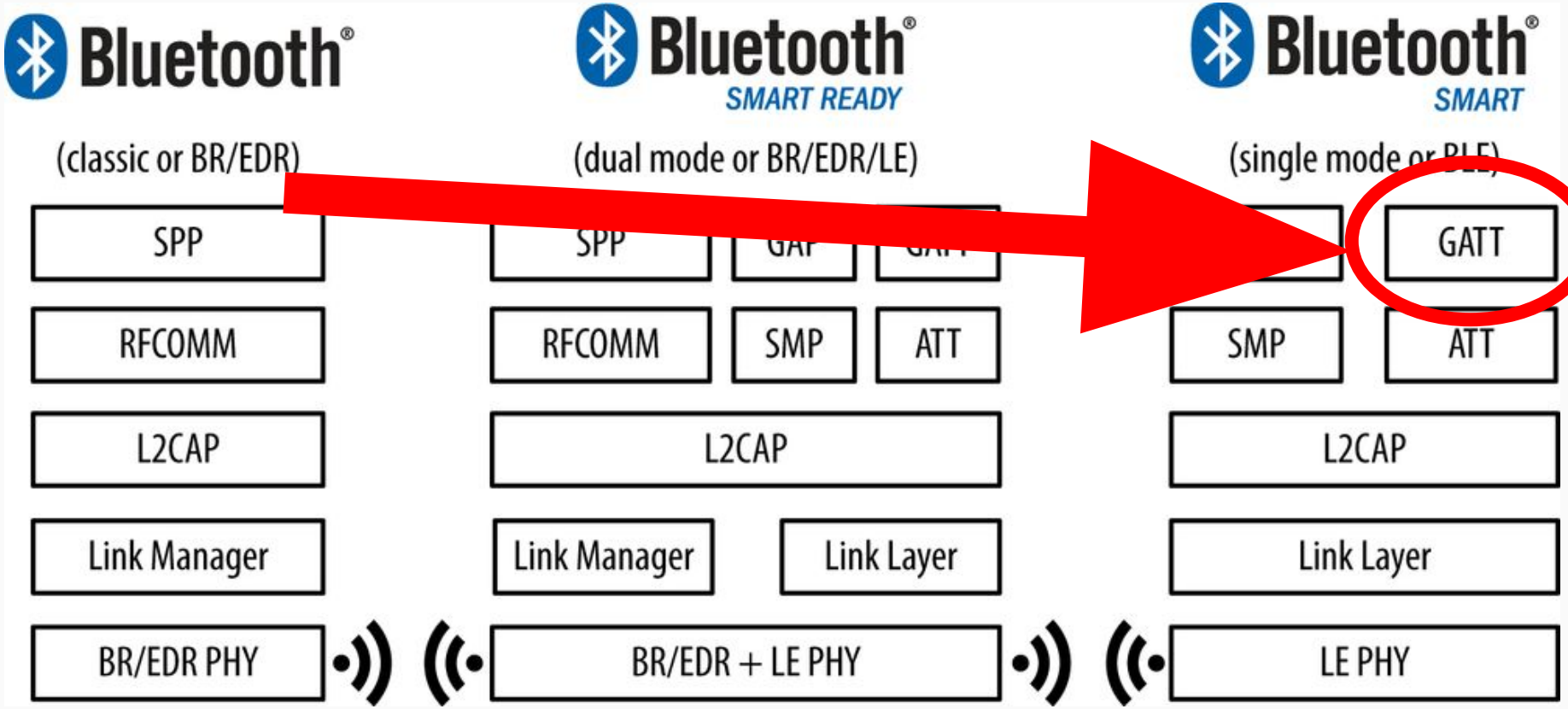
- Paired vs Unpaired
- Authentication
  - In band
  - Out of band
- Encryption

- 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
  - Adafruit/Nordic sniffer
- 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
- **<u>Not needed for this workshop</u>**

Software - Standard Bluetooth Modules

- 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
  - **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

BLE CTF

- Comes in 2 versions
- Version 2
  - Pre-release
  - 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

Tools

- Linux box
  - Or Vagrant on OSX/Windows
- Bluetooth module in your computer or a USB dongle
- Bluetooth software
  - Gatttool
  - Hcitool
  - Bleah - optional
- 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 virtualbox & vagrant
  - Install virtualbox usb extention pack
  - git clone https://github.com/hackgnar/ble_ctf
  - cd ble_ctf
  - add your bluetooth devices vendorid and productid
  - vagrant up
  - vagrant ssh

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 - install

```
git clone https://github.com/hackgnar/bleah.git

git clone https://github.com/IanHarvey/bluepy.git

cd bluepy

python setup.py build

python setup.py install

cd ../bleah

python setup.py build

python setup.py install
```

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

Come Get a MAC Address

- Come up to me and grab a MAC address to connect to your GATT server
- Once you have it, try and do a characteristics list list of your server
  - gatttool -b 11:22:33:44:55:66 --characteristics

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 sumbmit the following flag. `gatttool -b de:ad:be:ef:be:f1 --char-write-req -a 0x002c -n $(echo -n "1234567890123456789O"|xxd -ps)`

Finaly, 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 2

- Handle 0x002e
- Learn how to read handles

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’

Flag 2

Check out the ascii value of handle 0x002e and submit it to the flag submision handle 0x002c. If you are using gatttool, make sure you convert it to hex with xxd. If you are using bleah, you can send it as a string value.

Flag 2 - solution

Flag 3

- Flag 0x0030
- Read handle puzzle fun

Flag 3

Check out the ascii value of handle 0x0030. Do what it tells you and submit the flag you find to 0x002c.

Flag 3 - solution

Flag 4

- Flag 0x0016
- Learn about discoverable device attributes

Flag 4

Bluetooth GATT services provide some extra device attributes. Try finding the value of the Generic Access -> Device Name.

Flag 4 - solution

Flag 5

- Flag 0x0032
- Learn about reading and writing to handles

Flag 5

Read handle 0032 and do what it says. Notice that its not telling you to write to the flag handle as you have been. When you find the flag, go ahead and write it to the flag handle you have used in the past flags.

Flag 5 - solution

Flag 6

- Flag 0x0034
- Learn about reading and writing ascii to handles

Flag 6

Follow the instructions found from reading handle 0x0034. Keep in mind that some tools only write hex values while other provide methods for writing either hex or ascii

Flag 6 - solution

Flag 7

- Flag 0x0036
- Learn about reading and writing hex to handles

Flag 7

Follow the instructions found from reading handle 0x0036. Keep in mind that some tools only write hex values while other provide methods for writing either hex or ascii

Flag 7 - solution

Flag 8

- Flag 0x0038
- Learn about reading and writing to handles differently

Flag 8

Follow the instructions found from reading handle 0x0038. Pay attention to handles here. Keep in mind handles can be refrenced by integer or hex. Most tools such as gatttool and bleah allow you to specify handles both ways.

Flag 8 - solution

Flag 9

- Flag 0x003c
- Learn about write fuzzing

Flag 9

Take a look at handle 0x003c and do what it says. You should script up a solution for this one. Also keep in mind that some tools write faster than others.

Flag 9 - solution

Flag 10

- Flag 0x003e
- Learn about read and write speeds

Flag 10

Take a look at handle 0x003e and do what it says. Keep in mind that some tools have better connection speeds than other for doing reads and writes. This has to do with the functionality the tool provides or how it uses cached BT connections on the host OS. Try testing different tools for this flag. Once you find the fastest one, whip up a script or bash 1 liner to complete the task. FYI, once running, this task takes roughly 90 seconds to complete if done right.

Flag 10 - solution

Flag 11

- Flag 0x0040
- Learn about single response notifications

Flag 11

Check out handle 0x0040 and google search gatt notify. Some tools like gatttool have the ability to subsribe to gatt notifications

Flag 11 - solution

Flag 12

- Flag 0x0042
- Learn about single response indicate

Flag 12

Check out handle 0x0042 and google search gatt indicate. For single response indicate messages, like this chalange, tools such as gatttool will work just fine.

Flag 12 - solution

Flag 13

- Flag 0x0046
- Learn about multi response notifications

Flag 13

Check out handle 0x0046 and do what it says. Keep in mind that this notification clallange requires you to recieve multiple responses in order to complete.

Flag 13 - solution

Flag 14

- Flag 0x0048
- Learn about multi response indicate

Flag 14

Check out handle 0x0042 and google search gatt indicate. Keep in mind that this chalange will require you to parse multiple indicate responses in order to complete the chalange.

Flag 14 - solution

Flag 15

- Flag 0x004c
- Learn about BT client device attributes

Flag 15

Check out handle 0x004c and do what it says. Much like ethernet or wifi devices, you can also change your bluetooth devices mac address.

Flag 15 - solution

Flag 16

- Flag 0x004e
- Learn about message sizes MTU

Flag 16

Read handle 0x0048 and do what it says. Setting MTU can be a tricky thing. Some tools may provide mtu flags, but they dont seem to really trigger MTU negotiations on servers. Try using gatttool's interactive mode for this task. By default, the BLECTF server is set to force an MTU size of 20. The server will listen for MTU negotiations, and look at them, but we dont really change the MTU in the code. We just trigger the flag code if you trigger an MTU event with the value specified in handle 0x0048. GLHF!

Flag 16 - solution

Flag 17

- Flag 0x0050
- Learn about write responses

Flag 17

Check out handle 0x0050 and do what it says. This chalange differs from other write chalanges as your tool that does the write needs to have write response ack messages implemente correctly. This flag is also tricky as the flag will come back as notification response data even though there is no "NOTIFY" property.

Flag 17 - solution

Flag 18

- Flag 0x0052
- Hidden notify property

Flag 18

Take a look at handle 0x0052. Notice it does not have a notify property. Do a write here and listen for notifications anyways! Things are not always what they seem!

Flag 18 - solution

Flag 19

- Flag 0x0054
- Use multiple handle properties

Flag 19

Check out all of the handle properties on 0x0054! Poke around with all of them and find pieces to your flag.

Flag 19 - solution

Flag 20

- Flag 0x0056
- OSINT the author!

Flag 20

Figure out the authors twitter handle and do what 0x0056 tells you to do!

Flag 20 - solution

Fin

Ryan Holeman

- @hackgnar
- github/hackgnar

BLE CTF V1 is currently available. V2 will be released later this year