//=====< 2019-11-30T23:15:45+00:00 >==========\\

Hi! :3

== INTRO ==

I'm not going to lie, I don't find poking at Bluetooth or Bluetooth Low Energy (BLE) very fun. Not that's hard to figure out but there's little to no mystery about it any more. Everyone has a tool for it, most of it's vulnerabilities are known, and is the 2nd most popular wireless tech to hack besides WiFi. However! There's some fun to be had playing with this tech and you'll have good time messing around with it. The techniques I used to explore this device where learned from hackgnar's ble_ctf.
I recommend checking it out if you never used BLE before! :D

== TARGET ==

Kestrel is in the business of building weather and wind meters. The one I'm poking at is a 5500 Fire Weather Meter Pro and was designed for wildland fighters in mind. More or less it's the same as the other meters and can do some wize bang math to output the Probability of Ignition (POI) and Fine Dead Fuel Moisture… Also you're paying an extra $200 but you know lol, you're paying the price for something built for "Emergency Response". To help keep track of this data Kestrel built a phone app that connects to these devices using BLE. It stores weather mesurements and maps out trends in the weather. Because there's no real consequences (As far as I can tell) for open communication, the developers didn't bother implementing crypto / authentication for the BLE link.

IMG: Kestrel 5500

== BLE ==

BLE shares many of the same qualities as it's older brother Bluetooth but both are different beasts. The similarities include using the same frequency hopping tech, frequency range, and can be used on the same hardware. BLE was built to be cheaper and use less resources then Bluetooth it's capabilities have been reduced to support that. Below is a list of capabilities and characteristics for BLE...

- → Max Power: ~10mW (10dBm)
- → Latency: 3ms
- → Modulation: GFSK
- → Frequency: 2400 - 2483.5 MHz
- → Channels: 40 | 2MHz spacing | 3 advertising | 37 Data
- → Data Rate: 125 kbit/s | 1 Mbit/s | 2 Mbit/s
- → Application throughput: 0.27-1.37 Mbit/s
- → Security: 128-bit AES-CCM
- → Modes: Broadcast | Connection | Event Data Models | Reads | Writes

== DISCOVERY ==

→ Hardware: SENA UD100 Bluetooth dongle
→ Software: blue_hydra

blue_hydra is a cool tool maintained by the folks at Pwnie Express for

scanning the area for Bluetooth and or BLE devices. Below are the commands I used to search for the target device...

```
> hcitool dev            # List HCI devices
> hciconfig hci0 up      # Turn on on hci0
> ./blue_hydra           # Scan
```

The MAC address for this device is 88:6B:0F:9F:D1:15.

== ENUMERATION ==

→ Software: Bleah

I used evilsocket's 'bleah' tool to enumerate the GATT service of the target device. Unfortunately bleah is now depreciated and his work has been ported to the bettercap project. The GATT service can be can be described as a server client relationship between devices. Things get muddled fairly quickly when we also think about BLE's Master Slave relationship. Using the Attribute Protocol (ATT), a Master/Client(The Phone with the app for this application) can read and write data from the Slave/Server(Kestrel). The Slave/Server can also notify a Master/Client and push updates. The Kestrel pushes weather updates threw Notify services. Information about BLE's GATT service can be found here. Below are the commands used and the results I found enumerating the Kestrel.

```
> bleah -b "88:6B:0F:9F:D1:15" -e
```

88:6b:0f:9f:d1:15 (-41 dBm)

Vendor	Bluegiga Technologies OY
Allows Connections	✓
Address Type	public
Manufacturer	u'ea001100'
Incomplete 128b Services	'03290000-eab4-dea1-b24e-44ec023874db'
Complete Local Name	FIRE - 2334359
Flags	LE General Discoverable, BR/EDR Not Supported

```
@ Connecting to 88:6b:0f:9f:d1:15 ... connected.
@ Enumerating all the things .....
```

Handles	Service > Characteristics	Properties	Data
0001 -> 0008	85920000-0338-4b83-ae4a-ac1d217adb03		
0003	8592ffff-0338-4b83-ae4a-ac1d217adb03	READ	'\x05\x00\x08\x00\x0b\x00\x10\x00\x1
			\x00"\x00%\x00\'\x00)\x00+\x00.\x00
			\xff\xff\xff\xff\xff\xff\xff\xff\xf
			\xff\xff\xff\xff\xff\xff\xff\xff\xf
			\xff\xff\xff\xff\xff\xff\xff\xff\xf
			\xff\xff\xff\xff\xff\xff\xff\xff\xf
			\xff\xff\xff\xff\xff\xff\xff\xff\xf
			\xff\xff\xff\xff\xff\xff\xff\xff\xf
			\xff\xff\xff\xff\xff\xff\xff\xff\xf
			\xff\xff\xff\xff\xff\xff\xff\xff\xf
0005	85920100-0338-4b83-ae4a-ac1d217adb03	READ INDICATE	
0008	85920200-0338-4b83-ae4a-ac1d217adb03	WRITE	
000b	Device Name (00002a00-0000-1000-8000-00805f9b34fb)	READ	'FIRE - 2334359\x00\x00\x00\x00\x00\
000d	Appearance (00002a01-0000-1000-8000-00805f9b34fb)	READ	Unknown
000e -> 0010	Battery Service (0000180f-0000-1000-8000-00805f9b34fb)		
0010	Battery Level (00002a19-0000-1000-8000-00805f9b34fb)	READ	'\x0f'
0011 -> 001d	Device Information (0000180a-0000-1000-8000-00805f9b34fb)		
0013	Manufacturer Name String (00002a29-0000-1000-8000-00805f9b34fb)	READ	u'Kestrel by NK'
0015	Model Number String (00002a24-0000-1000-8000-00805f9b34fb)	READ WRITE	'5500FWL\x00\x00\x00\x00\x00\x00\x00
0017	Serial Number String (00002a25-0000-1000-8000-00805f9b34fb)	READ	'2334359\x00\x00\x00\x00\x00\x00\x00
0019	Hardware Revision String (00002a27-0000-1000-8000-00805f9b34fb)	READ	'Rev 11B\x00\x00\x00\x00\x00\x00\x00
001b	Firmware Revision String (00002a26-0000-1000-8000-00805f9b34fb)	READ	'1.21\x00\x00\x00\x00\x00\x00\x00\x0
001d	Software Revision String (00002a28-0000-1000-8000-00805f9b34fb)	READ	'\x00\x00\x00\x00\x00\x00\x00\x00\x0
001e -> ffff	03290000-eab4-dea1-b24e-44ec023874db		
0020	03290200-eab4-dea1-b24e-44ec023874db	READ	'0.06\x00\x00\x00\x00\x00\x00'
0022	03290101-eab4-dea1-b24e-44ec023874db	NOTIFY READ WRITE	'\x10\x0e\x00\x01\x05\x00\x00\x00\x0
0025	03290102-eab4-dea1-b24e-44ec023874db	READ WRITE	'\x00\x00\x00\x00\x00\x00\x00\x00\x0
0027	03290103-eab4-dea1-b24e-44ec023874db	READ WRITE	'\x00\x00\x00\x00\x00\x00\x00\x00\x0
0029	03290104-eab4-dea1-b24e-44ec023874db	READ WRITE	'\x00\x00\x00\x00\x00\x00\x00'

```
| 002b | 03290300-eab4-dea1-b24e-44ec023874db | NOTIFY READ       | '{\x7f` \x00\x00\x00\x00'
| 002e | 03290310-eab4-dea1-b24e-44ec023874db | NOTIFY READ       | '\x00\x00~\t\x01\x80/\x11\x80&\xff\x
| 0031 | 03290320-eab4-dea1-b24e-44ec023874db | NOTIFY READ       | '\xff\xff\xff\xff\x02\t\x00\x7f&\xff
| 0034 | 03290330-eab4-dea1-b24e-44ec023874db | NOTIFY READ       | 'g\x04\x1a\t\x00\xff\xff\xff\xff\xff
| 0037 | 03290340-eab4-dea1-b24e-44ec023874db | NOTIFY READ       | '\xff\xff\xff\x01\x80\xff\xff\xff\x0
| 003a | 03290105-eab4-dea1-b24e-44ec023874db | NOTIFY READ WRITE | '\x00\x00\x00\x00\x00\x00\x00\x00\x0
|      |                                      |                   |
```

\\ //

→ Headers: Headers in in the GATT are kinda like Ports in the TCP/IP world. They act as an address for each function that can be read or written too.

→ Service > Characteristics: Description for each service

→ Properties: What each service will do. For the Kestrel, there are services that will READ | WRITE | NOTIFY | INDICATE.
More info can be found here

→ Data: HEX, Little Endian, strings can be encoded in bash using...
> $(echo -n "STRING"|xxd -ps)
And decoded when you pipe the data into...
> awk -F':' '{print $2}'|tr -d ' '|xxd -r -p;printf '\n'

== REVERSING BLE ==

Android has a feature under (Settings → Developer Options → Bluetooth HCI snoop log) where you can enable packet capturing for Bluetooth and BLE. Earlier versions of android saved the Bluetooth pcap under '/sdcard/btsnoop_hci.log' however on my device, (Samsung S7) opened up a port via "USB debugging". Using wireshark I was able to capture BLE packets on port 5037. Mileage may vary and figuring this out took me the most time so don't feel bad if this slows you down. :3

Learning how these devices are controlled is very simple. All you do is send a message via the phone app then look at the raw data in the pcap. You should be looking for 'Sent Write Request' packet with a BLE handle address (ex. 0x0022) found in the details. The raw data is Little Endian so it may appear backwards and stupid at first but that's how the data gets transmitted. After the address, (which will look like 2200 because Little Endian) will be the value of the command. Bellow are values that I reordered and some of the commands I used to change settings on the Kestrel using the gatttool. These are only the commands that where transmitted from the phone app and not the received messages from the Kestrel.

== Header 0x0022 (NOTIFY | READ | WRITE) ==
→ Manage live readings
TIME SEC
100e0001 02 000100000f

Value: 100e000102000100000f #2 sec
Value: 100e000105000100000f #5 sec
Value: 100e00010a000100000f #10 sec
Value: 100e00013c000100000f #1 min

→ Manage Data Logs
Value: 100e00013c000100000f #Warp Log ON
Value: 100e00013c000100000f #Warp Log Off

→ Data Logging Rate
Value: 020000013c000100000f # 2sec
Value: 3c0000013c000100000f # 1min
Value: 100e00013c000100000f # 1hr
Value: 201c00013c000100000f # 2hr

== Header 0x0025 (READ | WRITE) ==
Value: 000001 #RESET
> gatttool -b 88:6B:0F:9F:D1:15 --char-write-req -a 0x0025 -n 000001

Value: 010000 #Clear Device Log
> gatttool -b 88:6B:0F:9F:D1:15 --char-write-req -a 0x0025 -n 010000

== Header 0x0027 (READ | WRITE) ==
Set name
> gatttool -b 88:6B:0F:9F:D1:15 --char-write-req -a 0x0027 -n $(echo -n "new_name"|xxd -ps)
> gatttool -b 88:6B:0F:9F:D1:15 --char-read -a 0x000b|

```
            awk -F':' '{print $2}'|tr -d ' '|xxd -r -p;printf '\n'
```

IMG: Name_Change
IMG: Name_Change_Cont.

== CONCLUSION ==

There's still a lot to be figure out with this Kestrel including how TXed data from the Kestrel is read by the phone app. Every Header that has the property of NOTIFY is most likely where the weather measurements are comming from. Below are the pcaps I recorded while experimenting with this device so feel free and contact me if you have any questions. :3

== PCAP ==
Kestrel_pcap.zip

-- NotPike