177246259 Bluetooth crash and corruption

2 people have starred this issue.

Android Public Tracker   Device-Specific App Issues

Reporter	oliver@motionmetrics.co
Type	Bug
Priority	P2
Severity	S2
Status	Won't Fix (Infeasible)
Assignee	ad...@google.com
Verifier	--
CC	ad...@google.com oliver@motionmetrics.co
AOSP ID	--
OEM Name	--
ReportedBy	Developer
Found In	--
Targeted To	--
Verified In	--
In Prod	

Oliver de Lange <oliver@motionmetrics.co> #1   Jan 12, 2021 02:30PM

Created issue.

I have shared a folder on drive containing supporting materials for this bugreport.

Context: Our company makes a digital ski coach which uses two bluetooth peripherals to collect force and motion data from sensors in a user's ski boot. getcarv.com for more info.

Summary:

Two main issues. I'm reporting both at the same time as it feels like they're linked.

1. Bluetooth on an android device turns off and on in response to a crash in the bluetooth stack
2. Bluetooth characteristic data arrives in our app corrupt. It appears like some race condition where an internal buffer is being flushed before it has finished being written to, causing the packet data to contain a portion of a previous packets data.

1. Bluetooth turns off and on in response to a crash in the Bluetooth stack.

Summary

The crash in the android bluetooth stack causes our app users to be notified "Error, bluetooth is turned off" and our bluetooth peripheral devices to re-connect causing lost data. This happens frequently during app usage.

Devices affected:

- OnePlus: 8 Pro, 8T
- Samsung: Galaxy S20 FE,
- LG: V60 ThinQ,

Maybe interesting: All of these devices have the SD 865 SoC.

Frequency of crash

Not consistent - some days it'll crash many times per day, other times it'll not crash for up to 8 hours (max duration tested without crash occurring on OnePlus 8T)

Supporting documents

Two bug reports taken after the bluetooth crash - sadly no hci snoop logs - not sure why as it was enabled.

1. https://drive.google.com/file/d/1LO4SRzZyfOqGgJQgYIk58Z90cWvavNkR/view?usp=sharing
2. https://drive.google.com/file/d/1UKsXvJoyz70znETxdWgsYmBFK6aKMJWL/view?usp=sharing

OnePlus_8T_Bluetooth Crash Logcat output.txt logcat logs capturing the bluetooth crash:

This seems to show the cause of the crash being `invalid HCI cmd type received` Interesting bits:

```
12-29 14:33:54.461 11912 11985 E vendor.qti.bluetooth@1.0-uart_controller: OnDataReady: Invalid hci packet type byte received 0x0, invalid_bytes_counter_ = 0
12-29 14:33:54.461 11912 11985 E vendor.qti.bluetooth@1.0-uart_controller: OnDataReady: Invalid hci packet type byte received 0x0, invalid_bytes_counter_ = 1
12-29 14:33:54.461 11912 11985 D vendor.qti.bluetooth@1.0-uart_controller: SsrCleanup: SSR triggered due to 18 skip sending special buffer
12-29 14:33:54.461 11912 11985 D vendor.qti.bluetooth@1.0-uart_controller: ReportSocFailure: reason 18
12-29 14:33:54.463 11912 11985 I vendor.qti.bluetooth@1.0-logger: BtPrimaryCrashReason:Invalid HCI cmd type received
```

OnePlus_8T_bluetoothcrash.txt logcat logs capturing the bluetooth crash

Interesting bits:

```
12-29 14:23:31.378  1666  4652 W ActivityManager: Scheduling restart of crashed service com.android.bluetooth/.gatt.GattService in 1000ms for connection
12-29 14:23:31.379  1666  4652 W ActivityManager: Scheduling restart of crashed service com.android.bluetooth/.btservice.AdapterService in 11000ms for connection
```

2. Corrupt / half written BLE characteristic data

Summary

BLE Characteristic data sent over the air (OTA) becomes corrupt by the time the receiving android application reads the bytes.

Devices affected

- Moto: G8 (Very reproducible)
- Huawei: P8 lite
- OnePlus: 8T (Only when Crash (first issue) happens), 8 Pro
- Samsung: A5 (Crash doesn't happen), Galaxy S20 FE
- LG: V60 ThinQ

All captures below are from the Moto G8 power, as this device exhibits the issue in a far more reproducible manner.

Points of interest:

- Turning wifi OFF on the Moto G8 reduces error rate significantly
- Increasing the data rate (data packets sent per second from our peripheral to the device) increases the error rate

Simplified example

1. Ble Peripheral sends abc123 OTA
2. Android app receives abc123 - OK!
3. Ble peripheral sends def456 OTA
4. Android app receives def123 - BAD!

The main point here is that at step 4, the android app receives the last half of the packet at step 1 (123).

Initial investigations

First, we added a monotonically increasing int as byte number 2 and as the last byte in the data array and checked they were equal in the app. This mostly confirmed that the peripheral firmware was not sending bad data and that the corruption was somewhere between the android device receiving the packet and the app reading the packet payload.

In addition to this, we confirmed that the packet data being sent OTA by our peripherals firmware was correct by sniffing traffic using an nRF52840 dongle - wireshark captures of both OTA and on device are available in supporting documents if you care to check these, but the important bits are below.

Detailed examples

Looking in more detail at the packet bytes from logcat & wireshark

In logcat.txt 3 corrupt packets were captured and their bytes were logged at three stages

1. OTA captured via nRF52840 BLE sniffer
2. On device, captured via wireshark androiddump extension (i think this is equivalent to looking at bluetooth hci snoop logs in wireshark)
3. In App, captured by logging the raw characteristic payload bytes

The OTA wireshark capture, the Android HCI snoop log capture and logcat.txt show the packets at the different stages.

In the follow examples, the packet numbers are the packet numbers as seen in wireshark when the relevant capture is opened.

There's also a bug report from the Moto G8 power which may be of use. HCI logs are in this one at location `.\FS\data\misc\bluetooth\logs`

ps. Apologies i did color code this stuff in a google doc - which might be easier to follow. I can give access if needs be.

Corruption 1

Bytes comparison:

Packet which the corrupt packet takes its last portion of bytes from (highlighted yellow):

Packet 1775 (ota-sniff) 03:0f:f0:14:f6:fb:fd:f3:fd:fb:fb:fc:f7:ff:fe:fe:ee:eb:ea:ed:f0:f5:e9:df:e6:ec:f1:f4:f3:ec: ea:eb:f0:f3:ec:de:db:e7:f1:f5:f6:f2:f2:ec:ec:f3:f6:f2:ea:e3:eb:f3:31:23:8c:fd:65:fd:fc:84:66:00:3b:00:e3:07:bb:da:0a:00:0f

Packet 335 (device-ble-sniff)
03:0f:f0:14:f6:fb:fd:f3:fd:fb:fb:fc:f7:ff:fe:fe:ee:eb:ea:ed:f0:f5:e9:df:e6:ec:f1:f4:f3:ec: ea:eb:f0:f3:ec:de:db:e7:f1:f5:f6:f2:f2:ec:ec:f3:f6:f2:ea:e3:eb:f3:31:23:8c:fd:65:fd:fc:84:66:00:3b:00:e3:07:bb:da:0a:00:0f

Packet 1775/335 (app-logs)
03:0f:f0:14:f6:fb:fd:f3:fd:fb:fb:fc:f7:ff:fe:fe:ee:eb:ea:ed:f0:f5:e9:df:e6:ec:f1:f4:f3:ec:-> | split | <- ea:eb:f0:f3:ec:de:db:e7:f1:f5:f6:f2:f2:ec:ec:f3:f6:f2:ea:e3:eb:f3:31:23:8c:fd:65:fd:fc:84:66:00:3b:00:e3:07:bb:da:0a:00:0f

Corrupt packet:

Packet 1783 (ota-sniff) 03:11:f0:14:f6:fa:fd:f3:fd:fa:fb:fc:f7:fe:ff:fe:ee:ea:ea:ed:f0:f5:e9:e0:e6:ec:f1:f4:f3:ec: ea:eb:f0:f3:ec:de:db:e7:f1:f5:f6:f2:f2:ec:ec:f4:f6:f2:ea:e3:eb:f3:25:23:58:fd:7a:fd:f9:84:68:00:38:00:e3:07:20:db:0a:00:11

Packet 337 (device-ble-sniff) 03:75:00:42:04:01:80:66:f4:fe:fb:37:62:77:f6:fe:fb:37:62:76:01:00:00:00:00:00:00:00:52:56:89: ea:eb:f0:f3:ec:de:db:e7:f1:f5:f6:f2:f2:ec:ec:f3:f6:f2:ea:e3:eb:f3:31:23:8c:fd:65:fd:fc:84:66:00:3b:00:e3:07:bb:da:0a:00:0f

Packet 1783/337 (app-logs) 03:75:00:42:04:01:80:66:f4:fe:fb:37:62:77:f6:fe:fb:37:62:76:01:00:00:00:00:00:00:00:52:56:89:-> | split | <- Everything right of here is from packet 1775/335... ea:eb:f0:f3:ec:de:db:e7:f1:f5:f6:f2:f2:ec:ec:f3:f6:f2:ea:e3:eb:f3:31:23:8c:fd:65:fd:fc:84:66:00:3b:00:e3:07:bb:da:0a:00:0f

Points of interest:

1. Here, in the corrupt packet only the first byte (03) is that same OTA as on device (highlighted red)
2. Where does "03:75:00:42:04:01:80:66:f4:fe:fb:37:62:77:f6:fe:fb:37:62:76:01:00:00:00:00:00:00:00:52:56:89" come from - its not from another packet. (highlighted blue)
3. The monotonically increasing byte at position 2 should be 11 as per OTA sniff, but its 75, which indicates the data is from a very old packet (highlighted green), or a packet type which doesn't have the monotonically increasing integer as the first byte (non data packet, like status etc)

Corruption 2

Packet which corrupt packet gets last bytes from:

Packet 2249 (ota-sniff) 03:a3:f0:14:f7:fa:fd:f3:fd:fb:fc:fc:f7:fe:ff:fe:ed:ea:ea:ed:f0:f5:e9:df:e7:eb:f1:f4:f3 ec:ea:eb:f0:f3:ec:de:db:e7:f1:f5:f6:f2:f2:ec:ed:f4:f6:f2:eb:e3:eb:f3:40:23:68:fd:8b:fd:00:85:68:00:38:00:e7:07:a3:f7:0a:00:a3

Packet 494 (device-ble-sniff) 03:a3:f0:14:f7:fa:fd:f3:fd:fb:fc:fc:f7:fe:ff:fe:ed:ea:ea:ed:f0:f5:e9:df:e7:eb:f1:f4:f3 ec:ea:eb:f0:f3:ec:de:db:e7:f1:f5:f6:f2:f2:ec:ed:f4:f6:f2:eb:e3:eb:f3:40:23:68:fd:8b:fd:00:85:68:00:38:00:e7:07:a3:f7:0a:00:a3

Packet 2249/494 (app-logs) 03:a3:f0:14:f7:fa:fd:f3:fd:fb:fc:fc:f7:fe:ff:fe:ed:ea:ea:ed:f0:f5:e9:df:e7:eb:f1:f4:f3: SPLIT ec:ea:eb:f0:f3:ec:de:db:e7:f1:f5:f6:f2:f2:ec:ed:f4:f6:f2:eb:e3:eb:f3:40:23:68:fd:8b:fd:00:85:68:00:38:00:e7:07:a3:f7:0a:00:a3

Previous good packet:

Packet 2273 (ota-sniff) 03:aa:f3:14:f6:fa:fd:f3:fd:fa:fb:fc:f7:fe:ff:fe:ed:ea:e9:ed:f0:f5:e9:df:e6:eb:f0:f4:f3:ec:e9:eb:f0:f3:ec:dd:db:e7:f1:f5:f6:f2:f2:ed:ec:f3:f6:f2:eb:e3:eb:f3:36:23:7a:fd:5c:fd:fe:84:65:00:3b:00:e6:07:02:f9:0a:00:aa

Packet 505 (device-ble-sniff) 03:aa:f3:14:f6:fa:fd:f3:fd:fa:fb:fc:f7:fe:ff:fe:ed:ea:e9:ed:f0:f5:e9:df:e6:eb:f0:f4:f3:ec:e9:eb:f0:f3:ec:dd:db:e7:f1:f5:f6:f2:f2:ed:ec:f3:f6:f2:eb:e3:eb:f3:36:23:7a:fd:5c:fd:fe:84:65:00:3b:00:e6:07:02:f9:0a:00:aa

Packet 2273/505 (app-logs) 03:aa:f3:14:f6:fa:fd:f3:fd:fa:fb:fc:f7:fe:ff:fe:ed:ea:e9:ed:f0:f5:e9:df:e6:eb:f0:f4:f3:ec:e9:eb:f0:f3:ec:dd:db:e7:f1:f5:f6:f2:f2:ed:ec:f3:f6:f2:eb:e3:eb:f3:36:23:7a:fd:5c:fd:fe:84:65:00:3b:00:e6:07:02:f9:0a:00:aa

Corrupt packet:

Packet 2275 (ota-sniff) 03:ab:f4:14:f6:fa:fd:f3:fc:fb:fb:fc:f7:fe:ff:fe:ed:ea:e9:ed:f0:f5:e9:df:e6:eb:f0:f4:f2:eb:ea:eb:f0:f3:ec:de:db:e7:f0:f4:f6:f2:f2:ec:ec:f3:f6:f2:ea:e3:eb:f3:31:23:73:fd:06:fd:ff:84:67:00:3a:00:e4:07:33:f9:0a:00:ab

Packet 506 (device-ble-sniff) 03:1a:0b:ff:4c:00:09:06:03:0b:c0:a8:01:67:ff:fb:2d:5b:34:01:08:09:e2:9b:b7:43:41:52:56:ec:ea:eb:f0:f3:ec:de:db:e7:f1:f5:f6:f2:f2:ec:ed:f4:f6:f2:eb:e3:eb:f3:40:23:68:fd:8b:fd:00:85:68:00:38:00:e7:07:a3:f7:0a:00:a3

Packet 2275/506 (app-logs) 03:1a:0b:ff:4c:00:09:06:03:0b:c0:a8:01:67:ff:fb:2d:5b:34:01:08:09:e2:9b:b7:43:41:52:56:ec:ea:eb:f0:f3:ec:de:db:e7:f1:f5:f6:f2:f2:ec:ed:f4:f6:f2:eb:e3:eb:f3:40:23:68:fd:8b:fd:00:85:68:00:38:00:e7:07:a3:f7:0a:00:a3

Points of interest:

1. Once again, the second portion of the corrupt packet is from a previous packet (yellow)
2. Again the first portion of the corrupt packet doesn't seem to come from any previous packet, or the OTA bytes (red)
3. Same story with second byte

Corruption 3

Packet which corrupt packet gets last bytes from:

Packet 4961 (ota-sniff) 03:e4:f7:14:f6:fa:fd:f3:fd:fa:fc:fc:f7:fe:ff:fe:ed:ea:e9:ed:f0:f5:e9:df:e6:ec:f0:f4:f2:ec:ea:eb:f0:f3:ec:de:dc:e7:f1:f4:f6:f2:f2:ec:ec:f3:f6:f2:eb:e3:eb:f3:25:23:be:fd:4e:fd:f8:84:66:00:38:00:e7:07:51:9a:0b:00:e4

Packet 1375 (device-ble-sniff) 03:e4:f7:14:f6:fa:fd:f3:fd:fa:fc:fc:f7:fe:ff:fe:ed:ea:e9:ed:f0:f5:e9:df:e6:ec:f0:f4:f2:ec:ea:eb:f0:f3:ec:de:dc:e7:f1:f4:f6:f2:f2:ec:ec:f3:f6:f2:eb:e3:eb:f3:25:23:be:fd:4e:fd:f8:84:66:00:38:00:e7:07:51:9a:0b:00:e4

Packet 4961/1375 (app-logs) 03:e4:f7:14:f6:fa:fd:f3:fd:fa:fc:fc:f7:fe:ff:fe:ed:ea:e9:ed:f0:f5:e9:df:e6:ec:f0:f4:f2:ec:ea:eb:f0:f3:ec:de:dc:e7:f1:f4:f6:f2:f2:ec:ec:f3:f6:f2:eb:e3:eb:f3:25:23:be:fd:4e:fd:f8:84:66:00:38:00:e7:07:51:9a:0b:00:e4

Previous good packet:

Packet 4965 (ota-sniff) 03:e5:f6:14:f6:fa:fd:f3:fd:fb:fc:fc:f7:ff:ff:fe:ed:eb:ea:ed:f0:f5:ea:df:e7:ec:f0:f4:f3:ec:ea:ec:f0:f3:ec:de:dc:e7:f1:f5: f6:f2:f2:ec:ed:f3:f6:f2:eb:e3:eb:f3:22:23:a2:fd:3f:fd:f8:84:64:00:3a:00:e5:07:81:9a:0b:00:e5

Packet 1376 (device-ble-sniff) 03:e5:f6:14:f6:fa:fd:f3:fd:fb:fc:fc:f7:ff:ff:fe:ed:eb:ea:ed:f0:f5:ea:df:e7:ec:f0:f4:f3:ec:ea:ec:f0:f3:ec:de:dc:e7:f1:f5: f6:f2:f2:ec:ed:f3:f6:f2:eb:e3:eb:f3:22:23:a2:fd:3f:fd:f8:84:64:00:3a:00:e5:07:81:9a:0b:00:e5

Packet 4965/1376 (app-logs) 03:e5:f6:14:f6:fa:fd:f3:fd:fb:fc:fc:f7:ff:ff:fe:ed:eb:ea:ed:f0:f5:ea:df:e7:ec:f0:f4:f3:ec:ea:ec:f0:f3:ec:de:dc:e7:f1:f5: f6:f2:f2:ec:ed:f3:f6:f2:eb:e3:eb:f3:22:23:a2:fd:3f:fd:f8:84:64:00:3a:00:e5:07:81:9a:0b:00:e5

Corrupt:

Packet 4969 (ota-sniff) 03:e6:f7:14:f6:fa:fd:f3:fc:fa:fc:fc:f7:ff:fe:fe:ee:ea:e9:ed:f0:f5:ea:df:e7:ec:f0:f4:f2:ec:ea:eb:f0:f3:ec:de:db:e7:f1:f5:f6:f2:f2:ed:ec:f3:f6:f2:ea:e3:eb:f3:1b:23:78:fd:39:fd:f7:84:62:00:3a:00:e2:07:b9:9a:0b:00:e6

Packet 1377 (device-ble-sniff) 03:e6:f7:14:f6:fa:fd:f3:fc:fa:fc:fc:f7:ff:fe:fe:ee:ea:e9:ed:f0:f5:ea:df:e7:ec:f0:f4:f2:ec:ea:eb:f0:f3:ec:de:db:e7:f1:f5:f6:f2:f2:ec:ec:f3:f6:f2:eb:e3:eb:f3:25:23:be:fd:4e:fd:f8:84:66:00:38:00:e7:07:51:9a:0b:00:e4

Packet 4969/1377 (app-logs) 03:e6:f7:14:f6:fa:fd:f3:fc:fa:fc:fc:f7:ff:fe:fe:ee:ea:e9:ed:f0:f5:ea:df:e7:ec:f0:f4:f2:ec:ea:eb:f0:f3:ec:de:db:e7:f1:f5:f6:f2:f2:ec:ec:f3:f6:f2:eb:e3:eb:f3:25:23:be:fd:4e:fd:f8:84:66:00:38:00:e7:07:51:9a:0b:00:e4

Points of interest:

1. Again yellow shows second portion from old packet data
2. This time secondbyte is as expected from OTA bytes
3. In fact, a lot of the first portion of the data is correct as per OTA. This is as per the simple example given at the start

Extra info

1. I'm finding it had to pinpoint where the issue is, from the above it seems like its an issue with the bluetooth firmware on the device, possibly related to the SD 865 SoC. The processing from OTA to the Android bluetooth APIs is a bit of a black box to me at the moment. Apologies if i've missed something obvious.
2. Customers and myself included have no issues with other bluetooth operations on the devices with these issues. For example, bluetooth music works fine.
3. We've tested over 50 phones with the app at the end of last year, and non of those had these issues. This seems to only be occuring on very new SD 865 based phones, and then randomly, a very old Samsung Galaxy A5 which only has the corruption issue - not the crash.
4. I wasn't sure if this is the right place to raise this bug, since it may well be in the domain of the manufacturer to investigate. So I have also reached out to Motorola (no response) and OnePlus (in progress) as they're the devices I have with me now. But its good to have a central point of reference for this issue.

Thanks in advance, and sorry for the long bug report! Olly

ad...@google.com <ad...@google.com> #2   Jan 12, 2021 02:36PM

Assigned to ad...@google.com.

Thank you for reporting this issue. For us to further investigate this issue, please provide the following additional information:

Are you able to reproduce the issue on Pixel device as well?

Oliver de Lange <oliver@motionmetrics.co> #3   Jan 12, 2021 02:43PM

No, no pixel devices have been seen to exhibit this behavior

br...@gmail.com <br...@gmail.com> #4 Jan 13, 2021 03:59AM

My phone poped up that my network connection was interup. My service keeps going in and out.

Oliver de Lange <oliver@motionmetrics.co> #5 Jan 13, 2021 08:51AM

Sorry, I'm not quite sure I understand the meaning of the above comment? Can you elaborate please?

lk...@gmail.com <lk...@gmail.com> #6 Jan 14, 2021 01:54PM

We have also seen the data corruption issue, when the peripheral sends multiple packages one after another and sends the next package as soon as it receives the ACK for the previous package.

Our work-arounds have been to:
* In all callback methods in BluetoothGattCallback: switch to a new thread as fast as possible to let the BT thread return.
* In the onCharacteristicChanged() and onCharacteristicRead() methods: clone the data as the first thing.

ad...@google.com <ad...@google.com> #7 Jan 14, 2021 04:48PM

Please ignore comment #4. It's not from @google.

ad...@google.com <ad...@google.com> #8 Jan 14, 2021 04:50PM

Status: Won't Fix (Infeasible)

Unfortunately, this issue is not reproducible with our version of the AOSP source code for Pixel devices. We will internally investigate this and discuss it with respective OEM.

Oliver de Lange <oliver@motionmetrics.co> #9 Jan 14, 2021 04:58PM

Can you share any additional details you might have discovered when investigating the bug reports, and may I ask how you attempted to reproduce the issues? It feels quite specific to our scenario, ie two peripherals transmitting at high frequency, triggering a race condition type error.

What are the time frames for OEM discussions? There's a time pressure from my side to fix this as its affecting new flagship devices.

Oliver de Lange <oliver@motionmetrics.co> #10 Jan 21, 2021 01:14PM

Quick update, we bought a Samsung Galaxy S20 FE to reproduce this issue and so far I've been able to reproduce issue 1 - The BLE crash due to 'Invalid HCI cmd type'. It happened only after 5 and a half hours of streaming data @ 100hz on both peripherals - so not very reproducible, but this matches what we're seeing with out customers.

I'm fairly happy this isn't an Android bug, and leaning more on the SD 865 theory. I wonder, if google made a pixel with an SD 865 in, would it exhibit this issue? :)

Oliver de Lange <oliver@motionmetrics.co> #11 Jan 21, 2021 03:18PM

Some more supporting docs have been added to the drive:

samsung_s20_fe_logcat_reduced.txt - captured logcat output from the time of the BLE crash.

Note how the last two data packets are identicle before the crash. OTA BLE capture shows the packet only being sent once from the peripheral. Open samsung_s20_fe_blecrash_OTA_sniff.pcapng in wireshark and find packet 3,870,822.

```
01-20 18:07:18.171  6621  7191 V :::CARV:::CarvDeviceController$1listenForData: (data) received: 1611166038.171, side: left - 03:90:cb:19:de:e0:e1:e5:e3:e5:e4:e3:f1:e3:e8:f2:f5:fc:fd:f6:ff:f6:f7:f5:f2:f2:f5:ea:f4:f7:f1:f6:f8:f6:fa:f8:f7:f6:f9:f7:f4:f1:e6:ef:f5:f8:f3:ef:e9:e6:f6:f7:70:02:06:0d:b1:80:df:00:60:00:e8:ff:4c:f7:2e:15:81:01:90
01-20 18:07:18.172  6621  7191 W Rust     : deserializer/src/matcher.rs.deserializer::matcher:51 - 25236.78200000(time), 1611140803.75049996(zero), 1611166038.17100000(received_at))
01-20 18:07:18.173  6621  7191 V :::CARV:::CarvDeviceController$1listenForData: (data) received: 1611166038.173, side: left - 03:90:cb:19:de:e0:e1:e5:e3:e5:e4:e3:f1:e3:e8:f2:f5:fc:fd:f6:ff:f6:f7:f5:f2:f2:f5:ea:f4:f7:f1:f6:f8:f6:fa:f8:f7:f6:f9:f7:f4:f1:e6:ef:f5:f8:f3:ef:e9:e6:f6:f7:70:02:06:0d:b1:80:df:00:60:00:e8:ff:4c:f7:2e:15:81:01:90
01-20 18:07:18.174  6621  7191 W Rust     : deserializer/src/matcher.rs.deserializer::matcher:51 - 25236.78200000(time), 1611140803.75049996(zero), 1611166038.17300010(received_at))
01-20 18:07:18.177   846  3317 E vendor.qti.bluetooth@1.0-uart_controller: OnDataReady: Invalid hci packet type byte received 0x0, invalid_bytes_counter_ = 0
01-20 18:07:18.178   846  3317 E vendor.qti.bluetooth@1.0-uart_controller: OnDataReady: Invalid hci packet type byte received 0x0, invalid_bytes_counter_ = 1
01-20 18:07:18.178   846  3317 D vendor.qti.bluetooth@1.0-uart_controller: SsrCleanup: SSR triggered due to 18 skip sending special buffer
01-20 18:07:18.178   846  3317 D vendor.qti.bluetooth@1.0-uart_controller: ReportSocFailure: reason 18
01-20 18:07:18.190   846  3317 I vendor.qti.bluetooth@1.0-logger: Primary Reason for SoC Crash:Invalid HCI cmd type received
01-20 18:07:18.191   846  3317 I vendor.qti.bluetooth@1.0-logger: Secondary Reason for SoC Crash:Default at time :Wed Jan 20 18:07:18 2021
01-20 18:07:18.200   846  3317 D vendor.qti.bluetooth@1.0-uart_controller: ReportSocFailure send H/W error event to clients
01-20 18:07:18.200   846  3317 E vendor.qti.bluetooth@1.0-logger: Rx HW error event::Crash reason not found
01-20 18:07:18.201  2197  3322 E bt_hci  : [BTCORE] Ctlr H/w error event - code:0xf
01-20 18:07:18.201   846  3317 I vendor.qti.bluetooth@1.0-logger: FrameCrashEvent: for crash reason :18-0-Wed Jan 20 18:07:18 2021
01-20 18:07:18.201   846  3317 D vendor.qti.bluetooth@1.0-uart_controller: SendCrashPacket send crash reasons to the client
01-20 18:07:18.201  2197  3322 E bt_btif : btif_dm_error_reporter_cback : 3
```

Also added dumpstate-2021-01-20-18-10-09.zip which contains `./FS/data/log/bt/btsnoop_hci.log.last` - Which appears to be the last BLE comms before the crash.

The last data packet: Packet number 9,378 - btatt.value == 03:7e:d3:19:df:e0:e1:e5:e3:e4:e4:e1:f1:e3:e7:f1:f3:ff:fc:f7:fd:f5:f8:f3:f4:f1:f5:e8:f4:f7:f1:f6:f7:f5:f8:f7:f9:f6:f8:f6:f6:ef:e5:f0:f3:f8:f0:f1:ea:e9:f8:f6:a1:02:07:0d:b2:80:0e:01:5e:00:e3:ff:4d:f7:ac:14:81:01:7e - Correlates to packet 3,870,796 in samsung_s20_fe_blecrash_OTA_sniff.pcap, which is 13 packets before the last packet the app received before the crash.

After this packet are two HCI_EVT and one HCI_CMD (reset) I've added screenshots of the two HCI_EVT's which show a vendor (qualcomm?) specific hardware error.

Thu Jan 21 2021 15:13:12 GMT+0000 (Greenwich Mean Time).png Restricted
72 KB View Download

Thu Jan 21 2021 15:16:14 GMT+0000 (Greenwich Mean Time).png Restricted
88 KB View Download