Advanced Hexagon DIAG

and getting started with baseband vulnerability research

Alisa Esage
Zero Day Engineering

Chaos Communications Congress 2020

https://t.me/learningnotes

About me

- Reverse engineer, low-level hacker, advanced vulnerability researcher & exploit dev
  - Targets: previously Browsers, JS, Windows Kernel & userland; now mostly Hypervisors
  - Reluctant speaker: RECON 2009, ZeroNights 2011, PHDays 2014, POC x Zer0Con 2020
  - Hall of fame: Microsoft, Firefox, Oracle, Google, ...
  - Phrack 2015: “Exploitation of Microsoft XML”
- Passion for hard research targets, sprawling technological stacks, ultra narrow edge memory corruptions and non-trivial exploit engineering
  - it's my e-sport of choice
- My project: Zero Day Engineering {0days.engineer}
  - training and (soon) deep vulnerability research intelligence subscriptions

0days.engineer

About this talk

- Primary focus of this talk is on the modern state and system internals of **Qualcomm DIAG (QCDM)**, a proprietary baseband management and diagnostics protocol which is included in Qualcomm's baseband OS on all Snapdragon SoCs (SDxxx) and MDM/MSM/SDM cellular modem chips
- Modern Qualcomm cellular modems run on a custom silicone (QDSP6) with Qualcomm-proprietary ISA named **Hexagon**, in which all the Qurt RTOS code is written, including DIAG handlers and OTA vectors
- With a bit of generalized **overview of baseband vulnerability research** for those who actually read the slides

It started around 2 years ago...

During past 3 years I was working on **virtualization and hypervisor vulnerability research** and exploit dev. In early 2019 I just completed a little research project with Microsoft Hyper-V [HYPERVISORS], was getting bored with hypervisors and looking for something new and challenging to put my brain to for a short-term distraction.

Basebands are challenging for the same reasons as hypervisors, though to a larger extent: the technological stack is enormously extensive and varied; data flows traverse multiple privilege boundaries; lowest level operations stand on the brink of pure Physics; a combination of ultra-low-level access requirements with remote wireless attack vectors.

Basebands let **no trivial debugging introspection**, which an instantly fun challenge for low-level hacking lovers like myself. On most modern OTS implementations JTAG is fused and the baseband OS doesn't export any kernel debugging facilities (similar to iOS)

Also, I had an old **USRP** B100 (sic!) from my hackerspace foundation period, that needed some good usage

Let’s get started

Agenda

1️⃣, 2️⃣: research directors, C-level and everyone else

2️⃣, 3️⃣: security researchers, software engineers, hackers

All materials in this presentation are based on my own independent work, views and analysis (no affiliations)

https://t.me/learningnets

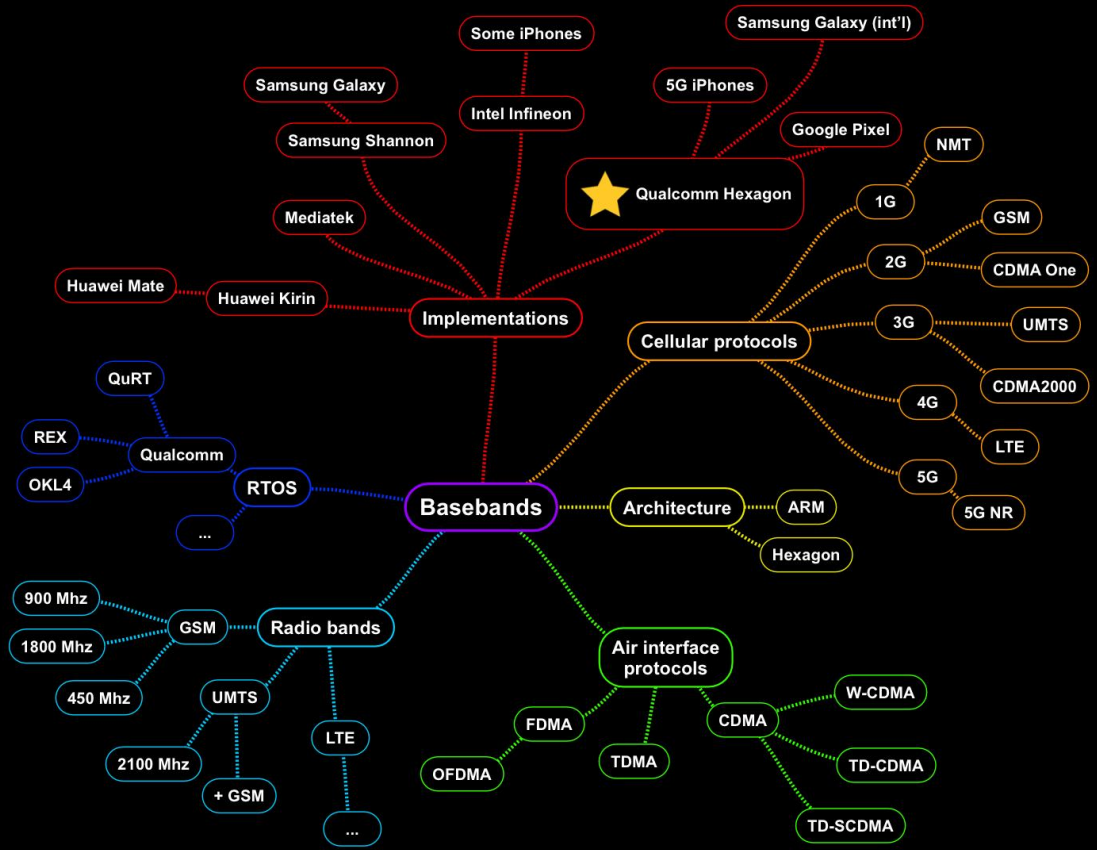
- The Big Picture 1️⃣
  - Basebands technological landscape
  - Generalized architecture & threat models
  - Security research overview
- Hexagon baseband 2️⃣
  - Architecture overview
  - Hardening observations
  - Reverse-engineering
- DIAG protocol 3️⃣
  - Architecture & system internals
  - Diagchar driver & Qualcomm SMD/SMEM
  - New commands and capabilities
  - diagtalk

Part 1

The Big Picture

Cellular technologies 101

Cellular technology landscape (UE side)



Cellular protocols

Generation	Technology name	Air interface	Specifications	Comments
1G	NMT	FDMA		analog
2G	GSM	TDMA, FDMA	GSM 04.07, 04.08	
	CDMA One / IS-95	**CDMA**		Qualcomm
3G	UMTS	W-CDMA, TD-CDMA, TD-SCDMA	TS 24.007, TS 24.008, TS 44.018	
	CDMA2000 / IS-2000	CDMA		Qualcomm
4G	LTE	OFDMA	**Partially same as GSM+UMTS**	
5G	5G NR	OFDMA		

Cellular protocols

Generation	Technology name	Air interface	Specifications	Comments
1G	NMT	FDMA		analog
2G	GSM	TDMA, FDMA	GSM 04.07,	
3G				
4G	LTE	OFDMA	Partially same as GSM+UMTS	
5G	5G NR	OFDMA		

V·T·E **Channel access methods and media access control**

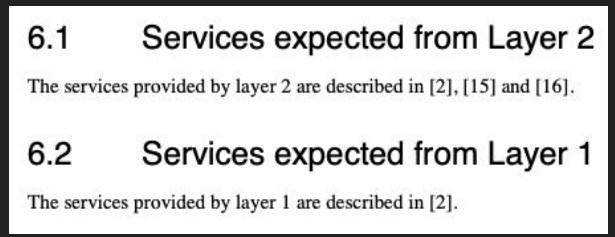
Channel-based	FDMA	FDM (OFDMA · SC-FDMA) · WDM (WDMA)
	TDMA	MF-TDMA · STDMA
	CDMA	W-CDMA · TD-CDMA · TD-SCDMA · DS-CDMA · FH-CDMA · MC-CDMA
	SDMA	HC-SDMA
	PDMA	
	PAMA	

https://t.me/learningnas

0days.engineer

GSM+ layers and 3gpp / ETSI standards

- L0 (unofficial): RF driver and hardware boundary
- L1: Physical layer
- L2: Data Link layer
- L3: Network layer



6.1 Services expected from Layer 2

The services provided by layer 2 are described in [2], [15] and [16].

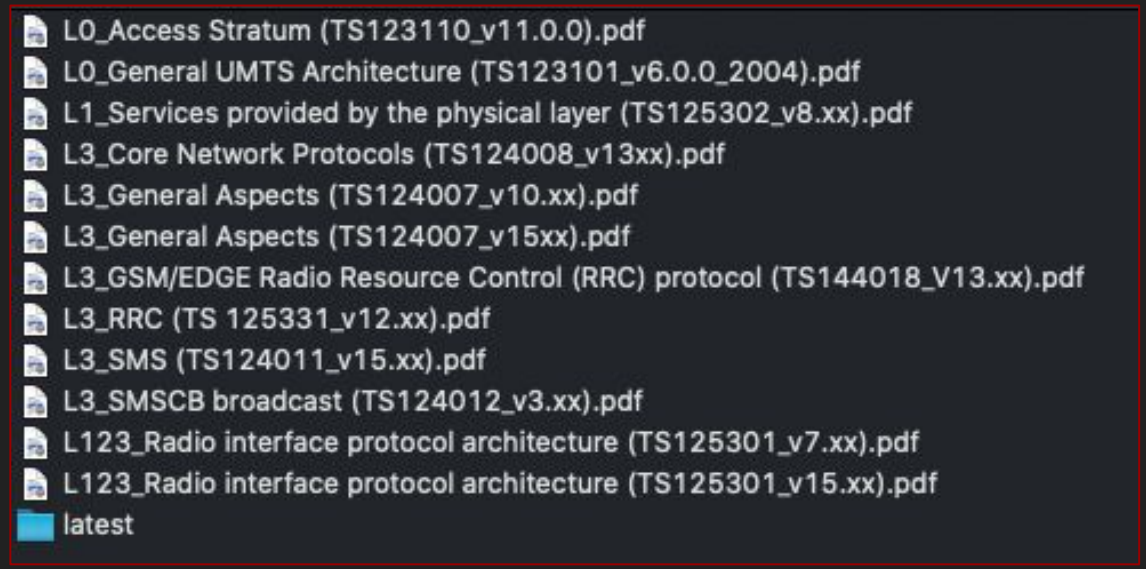
6.2 Services expected from Layer 1

The services provided by layer 1 are described in [2].

[2] 3GPP TS 25.301: "Radio Interface Protocol Architecture".

[15] 3GPP TS 25.321: "Medium Access Control (MAC) protocol specification".

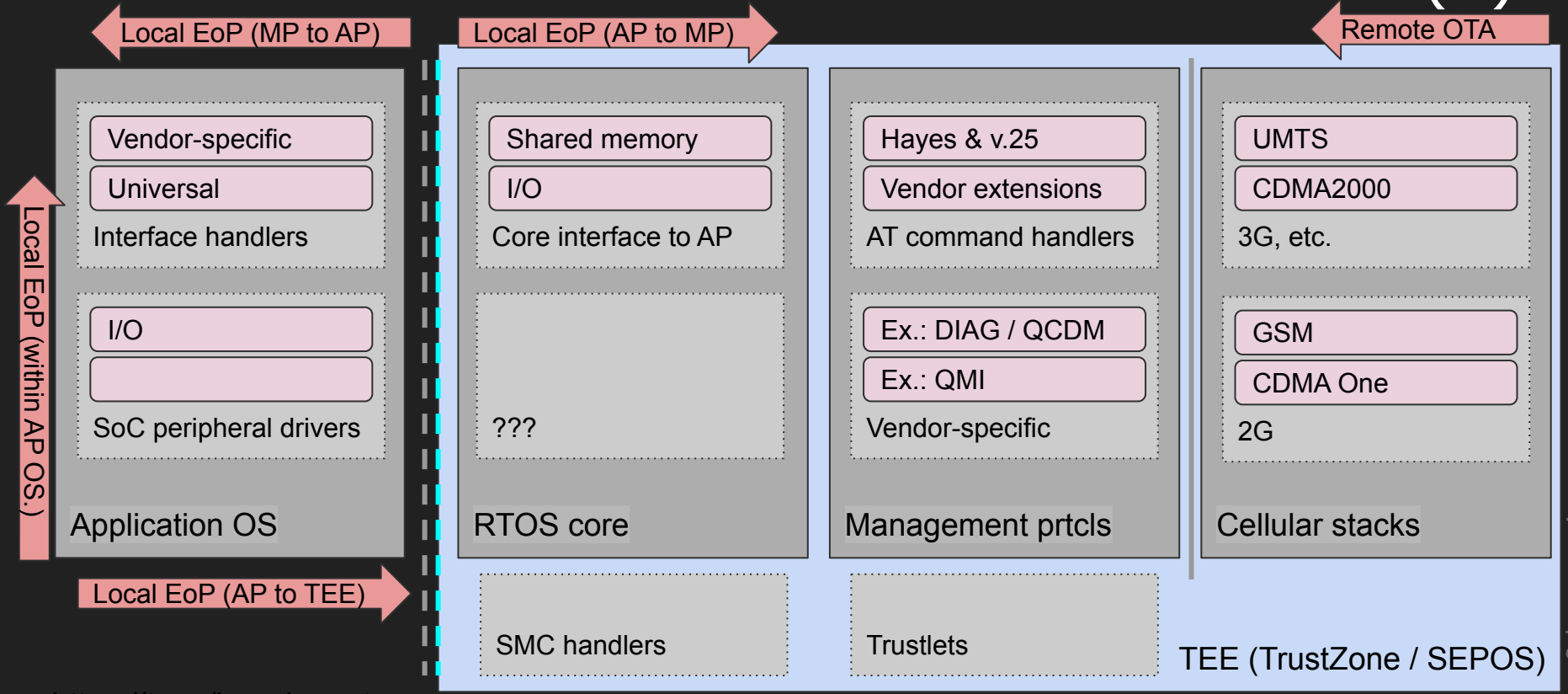
[16] 3GPP TS 25.322: "Radio Link Control (RLC) protocol specification".



Generalized baseband architecture + threat model

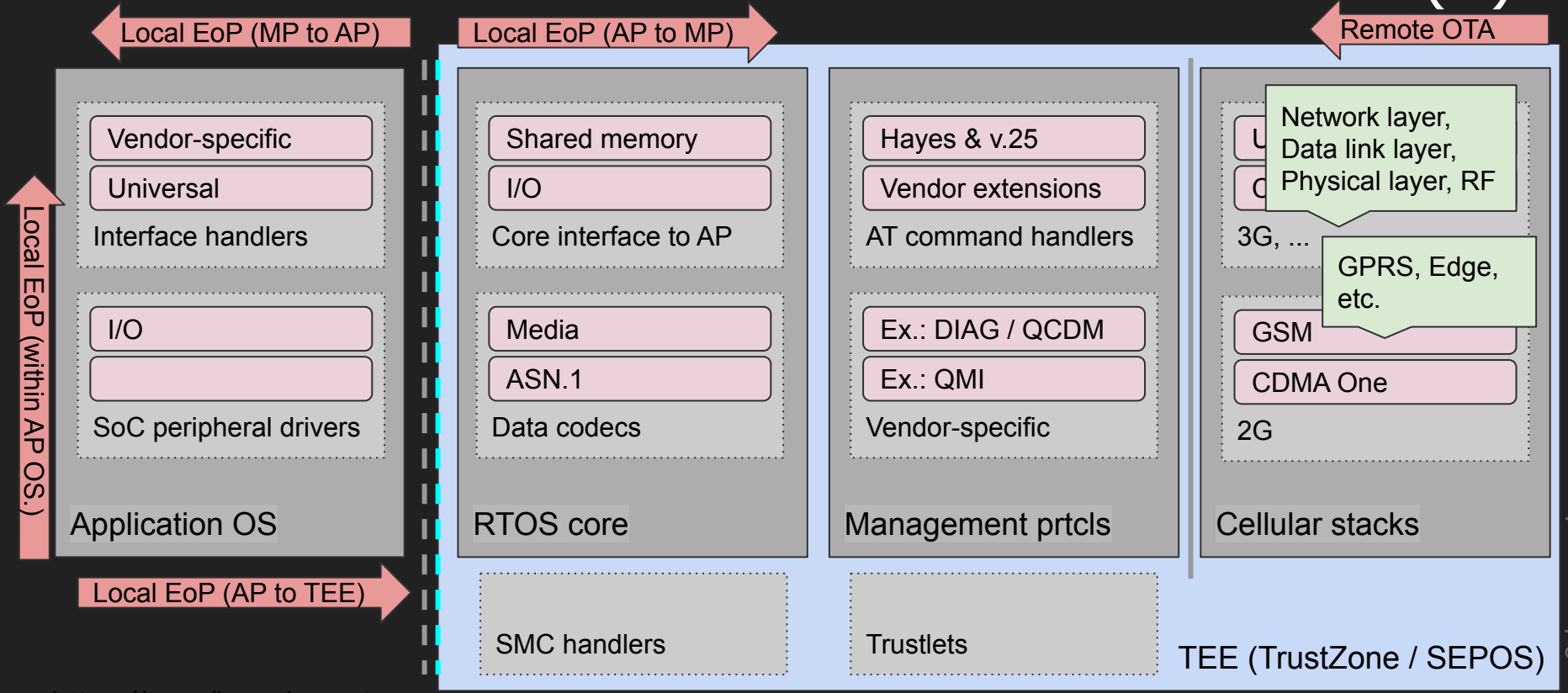
Note: INCOMPLETE!

Basebands: architecture + threat models (1)



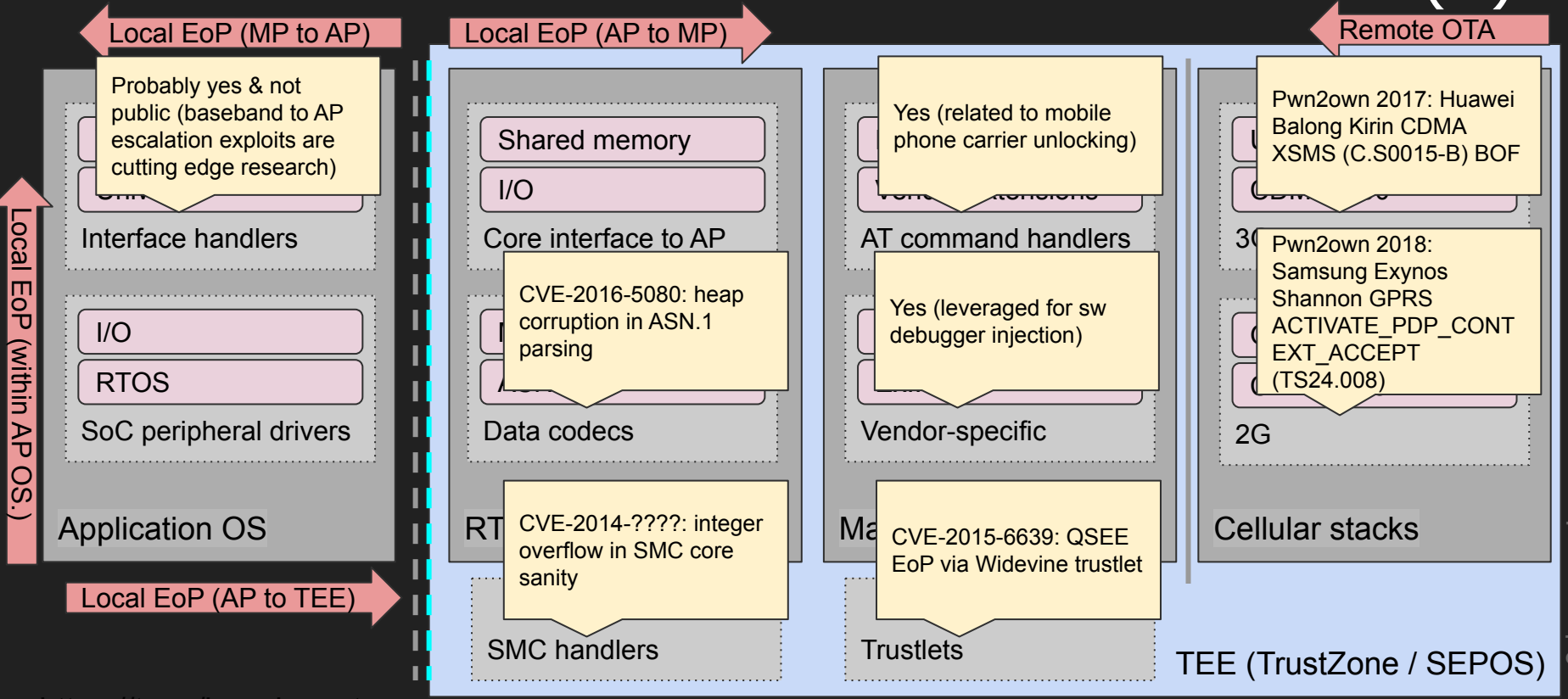
Note: INCOMPLETE!

Basebands: architecture + threat models (2)



Note: INCOMPLETE!

Basebands: architecture + threat models (3)



Baseband offensive research

Baseband offensive research landscape: OTA

How to

- Fake base station based on SDR
  - Related: “IMSI catcher”
- Reverse-engineering modem fw
  - Medium to hard complexity
  - + static analysis
- Fuzzing
  - Live (in-memory or open device)
  - Emulated

Targets

- Implementations
  - Shannon, Kirin, Hexagon, Infineon
- Protocols, layers, specific functions

Hardware

- SDR: Ettus Research USRP, Blade RF, etc.
- Advanced: Agilent 8960, CMU200, etc.
- Dev boards
- JTAG tools

Software

- GSM: OpenBTS, YateBTS
- UMTS: OpenBTS-UMTS
- LTE: srsLTE
- CDMA: **none**

Why Hexagon?

Initially Hexagon intrigued me due to the **esoteric architecture**. While majority of basebands are based on ARM, Qualcomm took the less-easy path of developing a novel MP architecture; unlike some other mobile vendors who take ARM specs, burn it to a slightly customized silicone and brand it as a novel chip (yes, we see it, and eagerly scoff at your marketing bullshit). So they developed a custom ISA, and then they built a custom DSP silicone from scratch for it.

This is a ***major* business investment** move that surely must be for good reasons. Qualcomm chips dominate the mobile market by a wide margin. This vendor cannot be expected to take such major risks based on a fancy whim.

Hexagon is a DSP, not a CPU. It's a different world vs x86/ARM/MIPS, and that world is the future. (More on this later)

Cursory reconnaissance indicated that Hexagon basebands are so **closed and hardened** [MODKIT] that it would require an advanced exploit to even begin building your own custom debugger for it

Should be fun enough!

Part 2

QDSP6 Hexagon

One month

As soon as I decided to focus on QDSP6 Hexagon baseband, I set a rigid time box of **one month** to the reconnaissance project, and started research. It was my first exposure to basebands and cellular protocols.

As usual, I started my research with a **systematic review** of all available security publications (1.5 count in this case), analysis of vendor's security advisories, studying all available official documentation, SDKs and potentially related open source code bases. I then performed **deep technical analysis** of all publicly documented security bugs in basebands, set up a research platform with OpenBTS and USRP, and skimmed through 3gpp specifications.

Concurrently, I took out the modem binary from the firmware of my test device (Nexus 6P with angler Android and MSM kernel) and started **reverse-engineering** it.

Typically my goals in preliminary reconnaissance projects: 1) gear up and build a research platform, 2) map out attack surfaces, and 3) find at least one good zero-day bug. I quickly completed 1 and 2 and stumbled at 3, and realized that it was even harder than I expected.

Qualcomm, why so hard?

QDSP6 / Hexagon

- Unfamiliar arch
  - VLIW, closer to GPU than CPU
  - very different from x86/ARM/MIPS
- No decompiler
  - Only disasm
- No QEMU full system emulation
  - Want!
- No binary patch diffing

Vs.

- Samsung Shannon (for example)
  - On demand memdumps, downloader mode, familiar architecture, plenty of log strings, decompiler, binary diffing

Hardenede

- Live introspection
  - Off-the-shelf devices are fused and hardened
  - Baseband RTOS runs on a separate chip, protected by QSEE TrustZone
  - No debugging, no crashdumps, can’t read mDSP memory
- Reverse engineering
  - Huge binary
  - No debug symbols
  - Obscure RTOS
  - Parts of code are compressed / relocated
  - NO LOG STRINGS!!!

Hexagon 101

[SNAPDRAGON] Qualcomm® Snapdragon™ 820E Processor (APQ8096SGE) https://developer.qualcomm.com/download/sd820e/qualcomm-snapdragon-820e-processor-apq8096sge-device-specification.pdf

Snapdragon 820E

- HTC 10, Galaxy S7
- 1st gen AI engine based on Hexagon

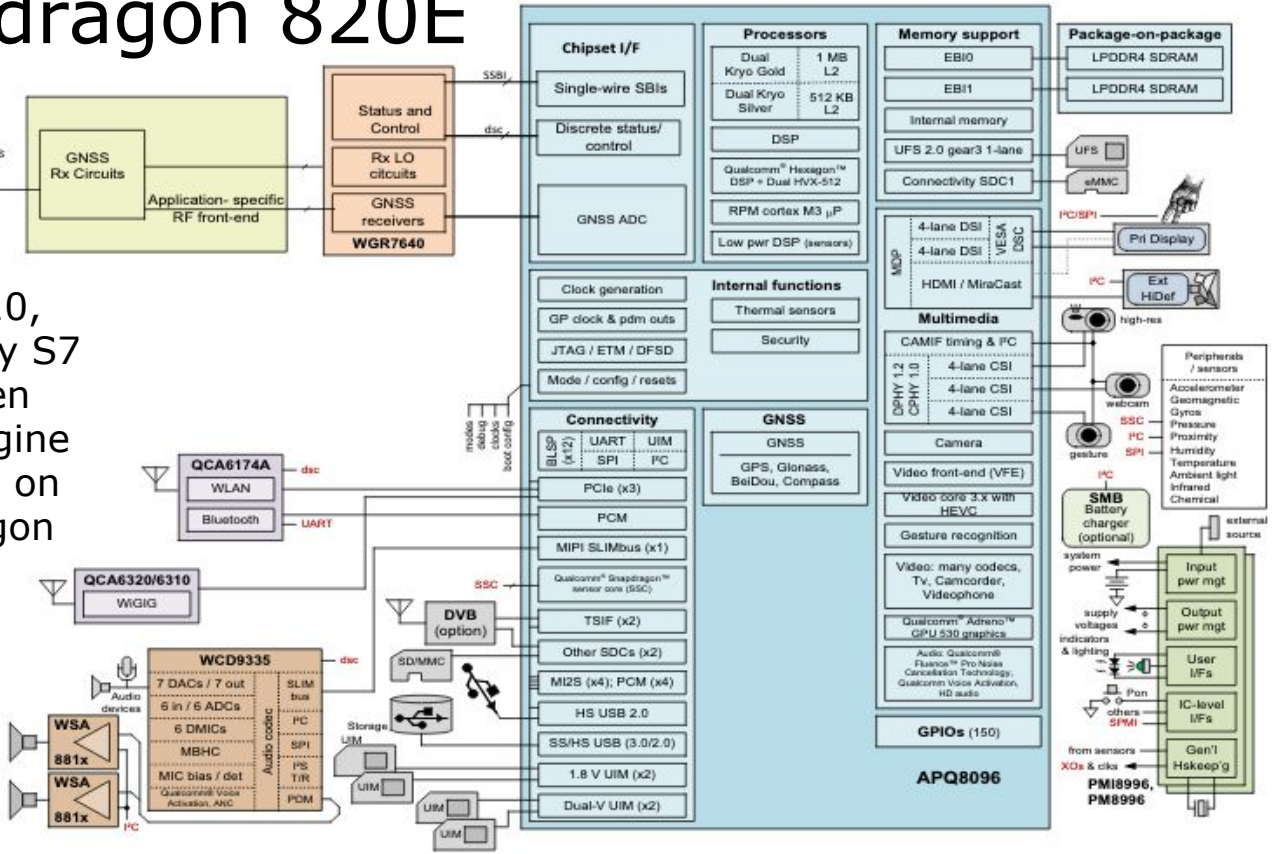
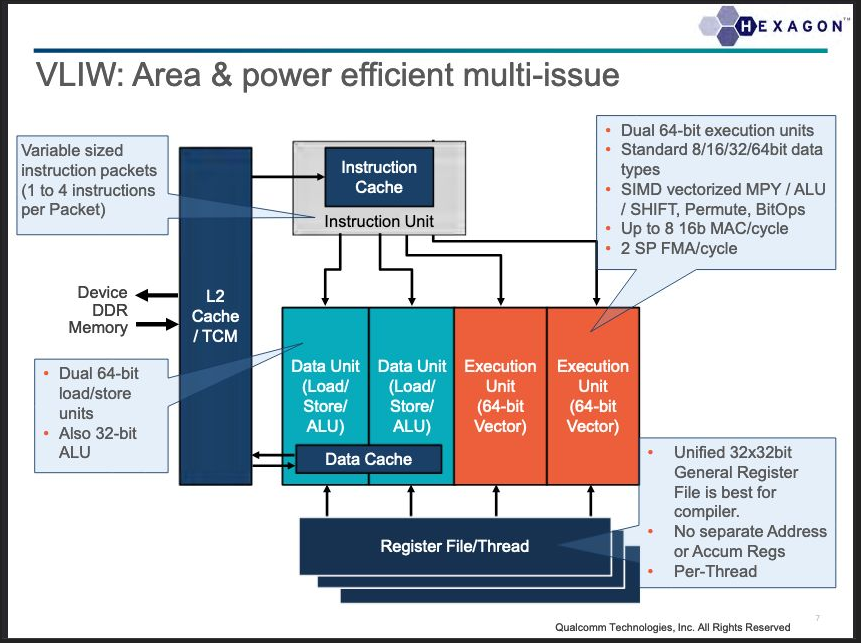
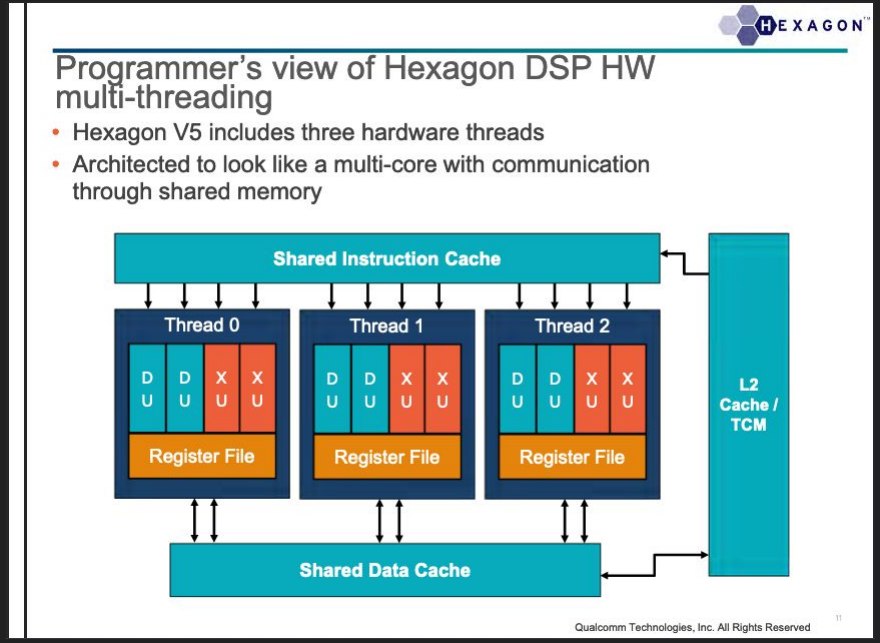


Figure 1-1 APQ8096SGE functional block diagram and example application

Hexagon: architecture properties





https://developer.qualcomm.com/software/hexagon-dsp-sdk/dsp-processor

[HEXAGONDSP] https://developer.qualcomm.com/download/hexagon/hexagon-dsp-architecture.pdf

Hexagon: programmer's view

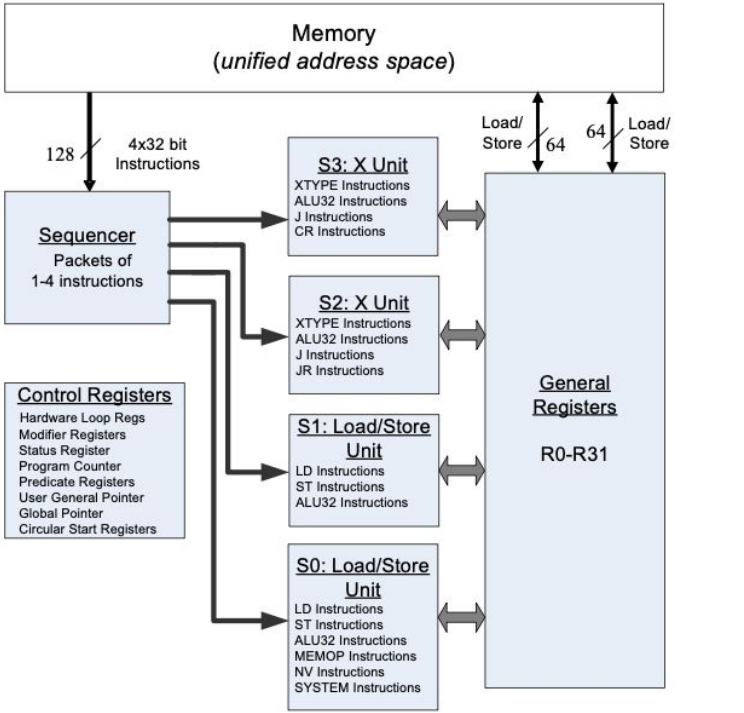


Figure 1-1 Hexagon V62 processor architecture

1.3.6 Instruction packets

Sequences of instructions can be explicitly grouped into packets for parallel execution. For example:

```
{
    R8 = memh(R3++#2)
    R12 = memw(R1++#4)
    R = mpy(R10,R6):<<1:sat
    R7 = add(R9,#2)
}
```

1.3.7 Dot-new instructions

In many cases, a predicate or general register can be both generated and used in the same instruction packet. This feature is expressed in assembly language by appending the suffix ".new" to the specified register. For example:

```
{
P0 = cmp.eq(R2,#4)
if (P0.new) R3 = memw(R4)
if (!P0.new) R5 = #5
}

{
R2 = memh(R4+#8)
memw(R5) = R2.new
}
```


[HEXAGONISA] https://developer.qualcomm.com/download/hexagon/hexagon-v62-programmers-reference-manual.pdf

Why did they roll their own architecture???

The future of technology is all about **optimized digital signal processing**. Growing requirements for graphics due to VR, ever increasing demands of media codecs due to online streaming, AI as in artificial neural network processing and deep learning.

Especially AI. It's the technology which is evolving at break-neck speed, with multiple trends developing concurrently. One of current trends is **offloading ANN processing to end user devices** as opposed to running it in the cloud. While the need for specialized AI hardware has been around for a while, and is high on the agenda of every major chips vendor on the planet; but with the offloading trend, it's further narrowed down to the **requirement of compact and cost-effective specialized chips** that be plugged into mobile devices.

Deep learning operates on huge matrices with rational numbers, at which **common CPU architectures are very bad**. SIMD couldn’t solve this. Chipset vendors started working on specialized NPU architectures, meanwhile software vendors under the pressure of market demands were forced to run ANNs on GPU. But GPUs were not made for this. The need for specialized hardware for cost-effective ANN operations remains.

Result: Hexagon has virtually no competition as an all-in-one cost-effective and optimized DSP for cellular signal processing, hardware-accelerated audio and video, sensor processing, and AI

https://en.wikipedia.org/wiki/Machine_learning

Hardening

"Production-fused" ?

2.5 Commercial release configurations

Some features are intended for use only during the development and debugging stages. These features must be disabled prior to commercial release to improve performance and user experience.

Disabling kernel debug feature to optimize HLOS performance

By default, the kernel configuration file includes many settings that are useful during the development and debugging phase but may affect performance.

To disable the default kernel debugging settings, use LINUX/android/kernel/arch/arm/configs/msm8226-perf_defconfig instead of msm8226_defconfig.

Enabling Subsystem Restart (SSR)

When a subsystem (mode, wifi, video core, etc.) crashes, the subsystem can be configured to automatically notify the AP. The AP then restarts the applicable subsystem. This provides a better user experience.

1. To enable subsystem restart prior to commercial release, open the android/device/qcom/msmXXXX/system.prop file.

2. Modify the persist.sys.ssr.restart_level property as follows:

persist.sys.ssr.restart_level=modem,wcnss,adsp,venus or 1

This enables subsystem restart for modem, wifi, adsp, and video subsystems. This is the recommended setting for commercial release.

During engineering development, this property should be set to 1 as shown below. This enables subsystem panic which is useful for engineering releases.

persist.sys.ssr.restart_level=1

Subsystem panic means that when each subsystem encounters a problem which triggers a reset, the AP will catch this event and will call kernel panic. As a result, the device can enter download mode to capture ramdump.

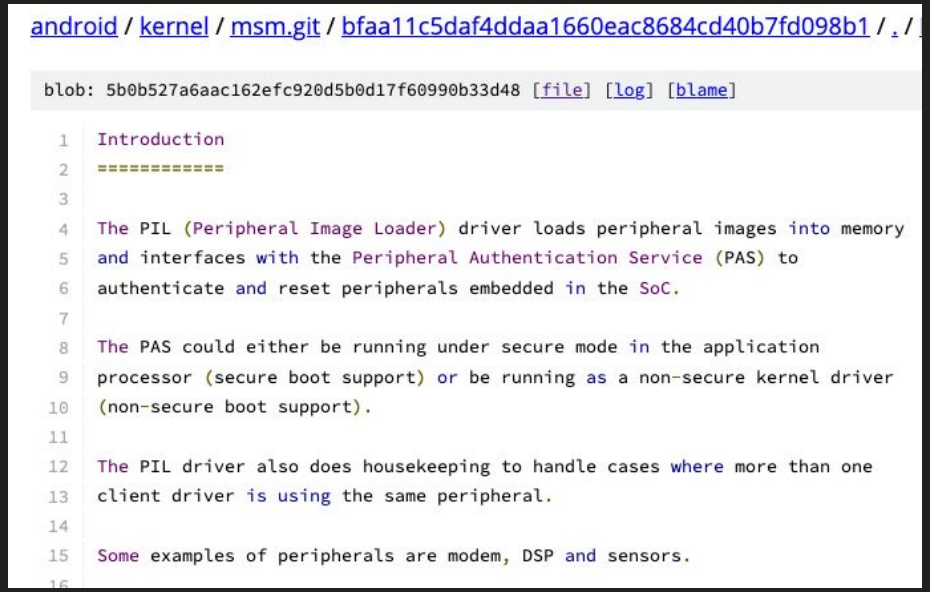
Disabling Download Mode

During development and debugging, Download mode is used to get memory dump for crash analysis. For a production device, Download mode is not needed since the device should just reboot.

80-NP616-2 A

[PRODFUSING]
https://pastebin.com/DUEbnuTf

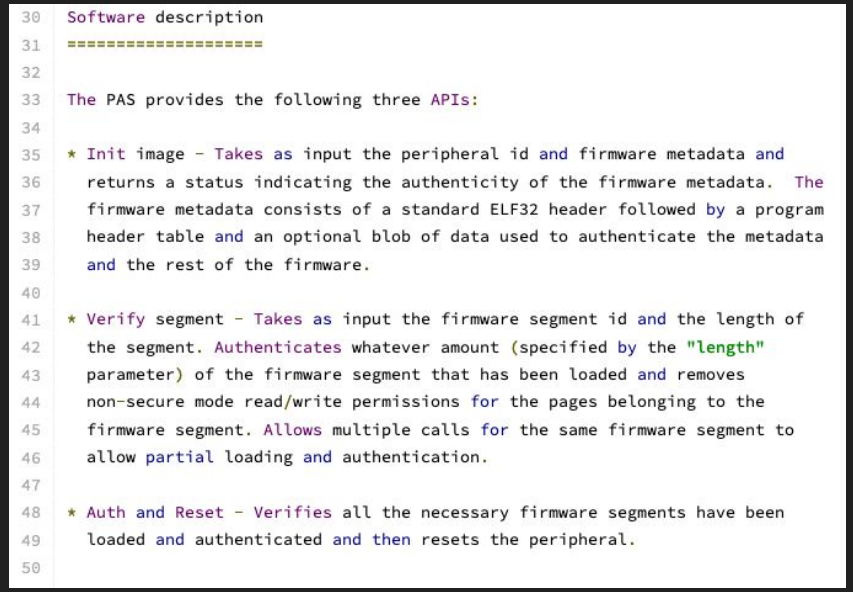
Android kernel PIL, TrustZone, and the MBA



android / kernel / msm.git / bfaa11c5daf4ddaa1660eac8684cd40b7fd098b1 / . /

blob: 5b0b527a6aac162efc920d5b0d17f60990b33d48 [file] [log] [blame]

```
Introduction
============

The PIL (Peripheral Image Loader) driver loads peripheral images into memory
and interfaces with the Peripheral Authentication Service (PAS) to
authenticate and reset peripherals embedded in the SoC.

The PAS could either be running under secure mode in the application
processor (secure boot support) or be running as a non-secure kernel driver
(non-secure boot support).

The PIL driver also does housekeeping to handle cases where more than one
client driver is using the same peripheral.

Some examples of peripherals are modem, DSP and sensors.
```



```
Software description
====================

The PAS provides the following three APIs:

* Init image - Takes as input the peripheral id and firmware metadata and
  returns a status indicating the authenticity of the firmware metadata.  The
  firmware metadata consists of a standard ELF32 header followed by a program
  header table and an optional blob of data used to authenticate the metadata
  and the rest of the firmware.

* Verify segment - Takes as input the firmware segment id and the length of
  the segment. Authenticates whatever amount (specified by the "length"
  parameter) of the firmware segment that has been loaded and removes
  non-secure mode read/write permissions for the pages belonging to the
  firmware segment. Allows multiple calls for the same firmware segment to
  allow partial loading and authentication.

* Auth and Reset - Verifies all the necessary firmware segments have been
  loaded and authenticated and then resets the peripheral.
```

Analysis

Qualcomm SoC software & dev ecosystem

Android MSM

- Android kernel with Qualcomm SoC -specific drivers
- CodeAurora and android/kernel/msm

Shared Memory Device

- Core interface to communicate Android with modem (maybe more)

QSEE

- Qualcomm’s TrustZone implementation
- Modem runs as a trustlet

Hexagon SDK

- Tools and headers for developers on Hexagon architecture

Dragonboard

- Open development board for OEM prototyping

Hexagon runtime libs

- AP-side support for code that runs on Hexagon processors

Reverse-engineering the Hexagon firmware

Firmware extraction

- unify_trustlet
- pymdt

Disassembly

- Several IDA Pro plugins available
  - gsmk
- Primitive disassembly via Hexagon SDK
  - objdump
- mDSP: missing sections with critical OTA vectors code (decompressed & relocated in runtime)
  - q6zip

Reverse-engineering analysis (mDSP)

- Start from some root points
  - RTOS task structs
  - Allocation primitives (a lot)
- IDA script to add the Qshrink'ed debug log strings
  - After that you can locate interesting code by grepping text

Debugging introspection

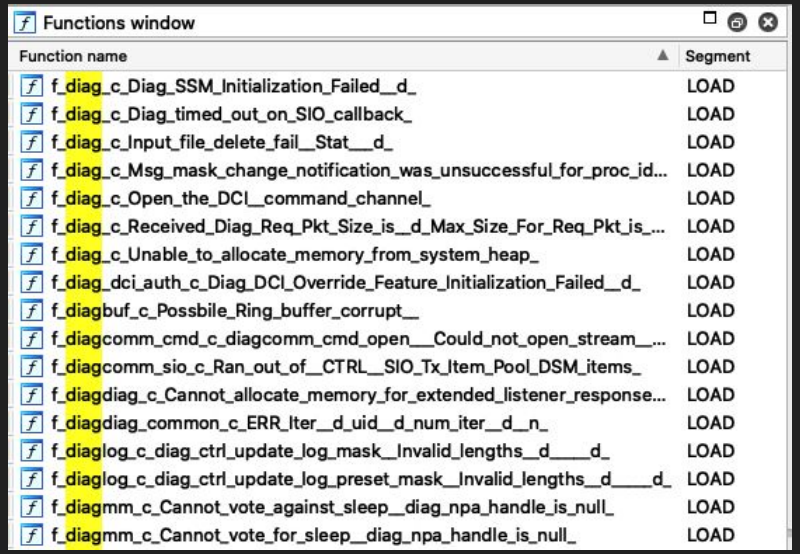
- None out of the box
  - JTAG – fused on production devices
  - Qcombbdbg – for obsolete ARM impls with Diag R/W commands available
  - Possible via exploit

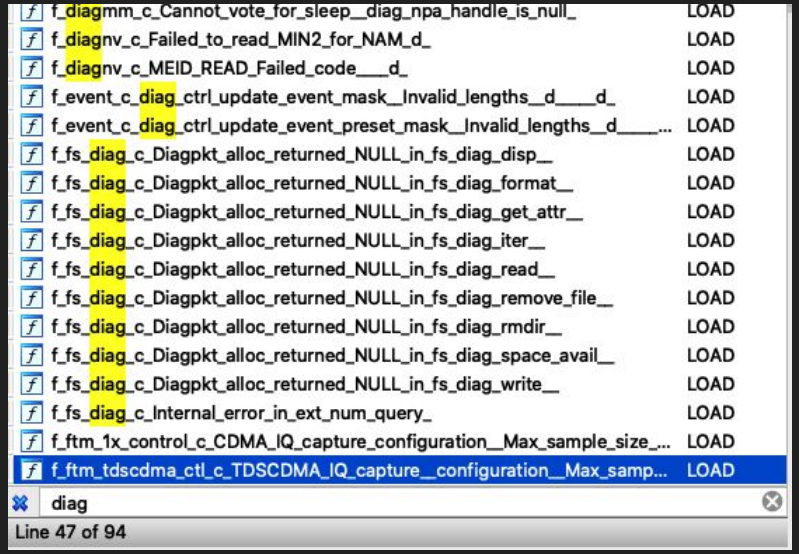
Where is my log strings???

```
LOAD:C15C4498 32 01 85 19                 .long 0x19850132
LOAD:C15C449C 0D 00 88 16 diag_c_diag_set_current_preset_mask_id___Preset__d_not_supported__:.long 0x1688000D
LOAD:C15C44A0 81 93 A4 71                 .long 0x71A49381        @ diag.c:diag_set_current_preset_mask_id - Preset %d not supported.
LOAD:C15C44A0                                                     @
LOAD:C15C44A4 0D 00 32 17 diag_c_Event_mask_change_notification_was_unsuccessful_for_proc_id__d_:.long 0x1732000D
LOAD:C15C44A4                                                     @ DATA XREF: f_diag_c_Msg_mask_change_notification_was_unsuccessful_for_proc_id__d_+24↑o
LOAD:C15C44A8 7D 6C E8 A2                 .long 0xA2E86C7D        @ diag.c:Event mask change notification was unsuccessful for proc id %d
LOAD:C15C44A8                                                     @
LOAD:C15C44AC 0D 00 40 17 diag_c_Log_mask_change_notification_was_unsuccessful_for_proc_id__d_:.long 0x1740000D
LOAD:C15C44AC                                                     @ DATA XREF: f_diag_c_Msg_mask_change_notification_was_unsuccessful_for_proc_id__d_+5C↑o
LOAD:C15C44B0 5D A0 5F 88                 .long 0x885FA05D        @ diag.c:Log mask change notification was unsuccessful for proc id %d
LOAD:C15C44B0                                                     @
LOAD:C15C44B4 0D 00 4D 17 diag_c_Msg_mask_change_notification_was_unsuccessful_for_proc_id__d_:.long 0x174D000D
LOAD:C15C44B4                                                     @ DATA XREF: f_diag_c_Msg_mask_change_notification_was_unsuccessful_for_proc_id__d_+A8↑o
LOAD:C15C44B8 01 53 EB 2F                 .long 0x2FEB5301        @ diag.c:Msg mask change notification was unsuccessful for proc id %d
LOAD:C15C44B8                                                     @
LOAD:C15C44BC 0D 00 2B 13 diag_c_efs_close___failed_:.long 0x132B000D
LOAD:C15C44BC                                                     @ DATA XREF: f_diag_c_Input_file_delete_fail__Stat___d_+24↑o
LOAD:C15C44C0 D5 0B A6 7A                 .long 0x7AA60BD5        @ diag.c:efs_close() failed
LOAD:C15C44C0                                                     @
LOAD:C15C44C4 0B 00 6C 0C diag_c_Received_Diag_Req_Pkt_Size_is__d_Max_Size_For_Req_Pkt_is__d__n_:.long 0xC6C000B
LOAD:C15C44C4                                                     @ DATA XREF: f_diag_c_Received_Diag_Req_Pkt_Size_is__d_Max_Size_For_Req_Pkt_is__d__n_+40↑o
LOAD:C15C44C8 2D 66 6D 3B                 .long 0x3B6D662D        @ diag.c:Received Diag Req Pkt Size is %d Max Size For Req Pkt is %d \n
LOAD:C15C44C8                                                     @
LOAD:C15C44CC 0B 00 62 07 diag_c_Diag_event_mask_SSM_Initialization_Failed__d_:.long 0x762000B
LOAD:C15C44CC                                                     @ DATA XREF: f_diag_c_Diag_SSM_Initialization_Failed__d_+138↑o
LOAD:C15C44D0 2F CE 73 22                 .long 0x2273CE2F        @ diag.c:Diag event mask SSM Initialization Failed %d
LOAD:C15C44D0                                                     @
LOAD:C15C44D4 0B 00 35 07 diag_c_Diag_log_mask_SSM_Initialization_Failed__d_:.long 0x735000B
LOAD:C15C44D4                                                     @ DATA XREF: f_diag_c_Diag_SSM_Initialization_Failed__d_+C8↑o
LOAD:C15C44D8 42 9D 61 DE                 .long 0xDE619D42        @ diag.c:Diag log mask SSM Initialization Failed %d
LOAD:C15C44D8                                                     @
LOAD:C15C44DC 0B 00 09 07 diag_c_Diag_SSM_Initialization_Failed__d_:.long 0x709000B
LOAD:C15C44DC                                                     @ DATA XREF: f_diag_c_Diag_SSM_Initialization_Failed__d_+5C↑o
LOAD:C15C44E0 25 83 81 F6                 .long 0xF6818325        @ diag.c:Diag SSM Initialization Failed %d
LOAD:C15C44E0                                                     @
LOAD:C15C44E4 0D 00 91 04 dword_C15C44E4: .long 0x491000D         @ DATA XREF: sub_C0C17510+138↑o
LOAD:C15C44E8 71 B3 D3 B2                 .long 0xB2D3B371
LOAD:C15C44EC 0D 00 31 13 diag_c_Input_file_delete_fail__Stat___d_:.long 0x1331000D
LOAD:C15C44EC                                                     @ DATA XREF: f_diag_c_Input_file_delete_fail__Stat___d_+40↑o
LOAD:C15C44F0 67 F3 9E 0B                 .long 0xB9EF367         @ diag.c:Input file delete fail! Stat: %d
LOAD:C15C44F0                                                     @
LOAD:C15C44F4 85 53 B6 C1                 .long aDiagC            @ "diag.c"
LOAD:C15C44F8 0B 00 BC 03 diagbuf_c_Attempt_to_alloc_too_much___d_:.long 0x3BC000B
LOAD:C15C44F8                                                     @ DATA XREF: f_diagbuf_c_Possbile_Ring_buffer_corrupt__+358↑o
LOAD:C15C44FC 79 BF 0B E1                 .long 0xE10BBF79        @ diagbuf.c:Attempt to alloc too much: %d
LOAD:C15C44FC                                                     @
LOAD:C15C4500 0B 00 51 04 diagbuf_c_Possbile_Ring_buffer_corrupt__:.long 0x451000B
LOAD:C15C4500                                                     @ DATA XREF: f_diagbuf_c_Possbile_Ring_buffer_corrupt__+264↑o
LOAD:C15C4504 61 A2 20 DD                 .long 0xDD20A261        @ diagbuf.c:Possbile Ring buffer corrupt!
LOAD:C15C4504                                                     @
LOAD:C15C4508 0B 00 88 02 diagdiag_common_c_Out_of_memory__Cannot_send_the_response_:.long 0x288000B
LOAD:C15C4508                                                     @ DATA XREF: LOAD:C0C1C818↑o
LOAD:C15C450C 05 A3 B6 69                 .long 0x69B6A305        @ diagdiag_common.c:Out of memory- Cannot send the response
LOAD:C15C450C                                                     @
LOAD:C15C4510 0B 00 97 02 diagdiag_common_c_Out_of_memory__Cannot_send_the_delayed_response_:.long 0x297000B
LOAD:C15C4510                                                     @ DATA XREF: LOAD:C0C1C840↑o
```

- Majority of log strings removed by linker-stage tool Qshrink4, replaced with an MD5 hash msg_hash.txt

Diag subsystem funcs





Mid-project reflections

Too hard for a 1-month project; serious vulndev is out of question

Can I still make something fun and useful within the time box allocated?

Maybe enable debug logging? With textual strings, not just protocol dumps. It should be fun to see what exactly the baseband is doing, and it's not trivial, since the log strings are stripped away

Also: ramdumps!

Trials & failures

Build & flash Android MSM kernel ✔️

- Nothing special, just a regular cross-build

Firmware downgrades ✔️

- Trivial

Build qcom fw ⚠️

- Ran out of time, probably wrong sources
- Someone just confirmed in a private talk that it's possible, will try again

Ramdumps 🚫

- AP kernel callbacks are not there + prod-fused?

Shady Q* diagnostic tools ❌

- QPST, QXDM, etc.
- Skipped this part

Dragonboard ❌

- Saved for later (maybe)
- For exploit dev you need debugging introspection on exactly the attacked device, not some abstract Qualcomm modem chip

Lauterbach debugger ⚠️

- Useless for prod devices, skipped
- Will check again if I can bypass JTAG fusing

But still…

Debug logging is here in the binary code, surely it’s used somehow?

Can it be enabled opt-in? Where will it log to? Is it exported to Android kernel?

DIAG protocol seems powerful for configuration, could it be the answer?

Part 3

QCDM DIAG

Qualcomm Diag / QCDM

Overview

- Qualcomm proprietary protocol for cellular modem RTOS management
- Alongside QMI and other Q protocols
- ~200 commands in theory

History

- Libqcdm / ModemManager
  - ~2010, partially reversed
- CCC2011, Guillaume Delugré
  - Diag message format + HDLC
  - Some interesting commands
  - Mostly irrelevant for modern production devices

Applications

- High-level debugging for OEM devs
- Baseband firmware reconfiguration
- (Obsolete) powerful diagnostics tools such as downloader mode, live dumps, memory R/W

Offensive perspective

- Local EoP attack vector from AP kernel to baseband RTOS
- Common scenario: mobile carrier unlocking exploits
- Scenario #2: exploit to enable a custom software debugger injection

Diag, modern status (first order view)

Public info

- ARM-based
- RTOS REX
- Downloader mode*
- Memory R/W commands*
- Live snapshots*
- Directly accessible channel over USB*

--

* may be still relevant for obscure OEM devices on Qualcomm MDM/MSM chips

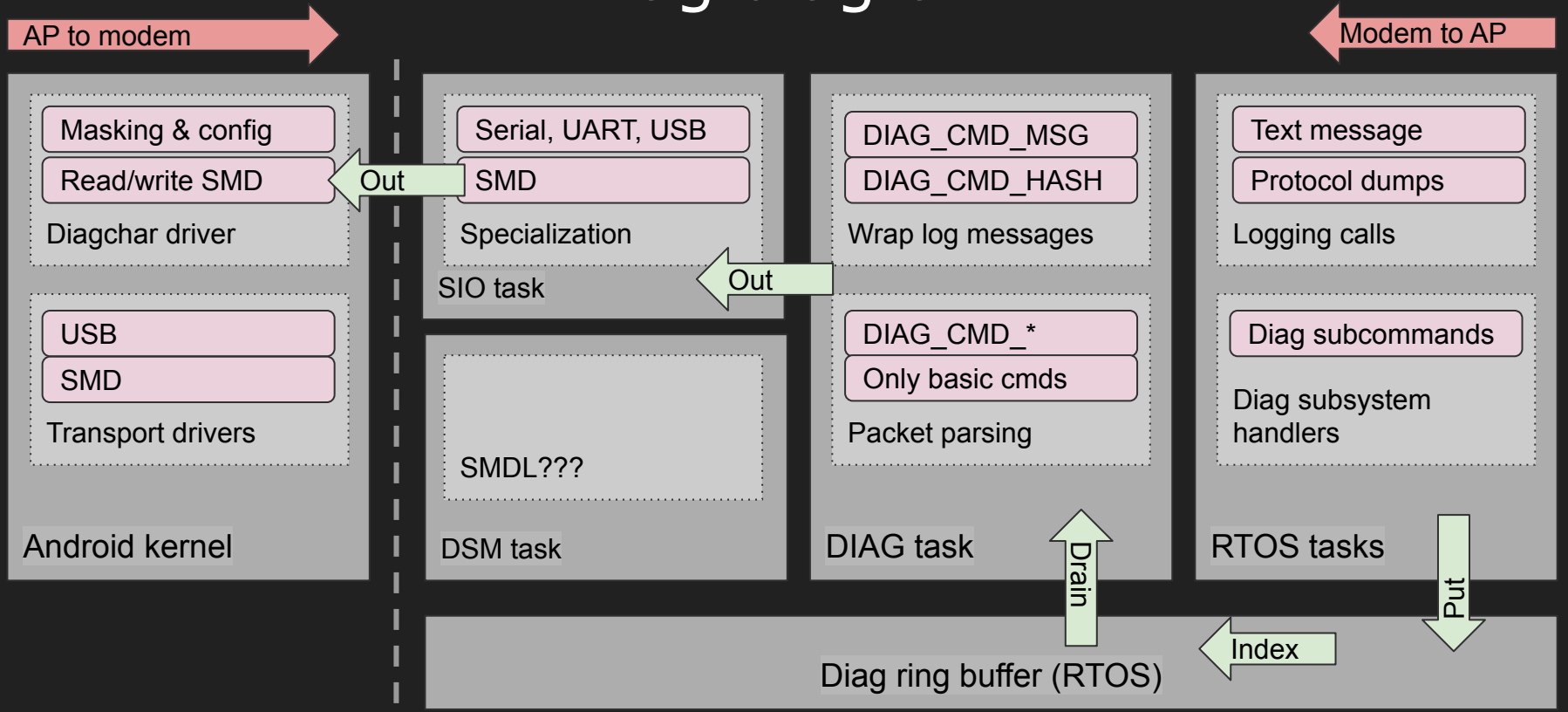
Current status

- QDSP6 / Hexagon -based
- RTOS QuRT
- No downloader mode*
- No memory RW*
- No live snapshots*
- No USB channel*
  - Possible to enable on some(?) devices via boot settings

--

* relevant to modern production devices (tested on Nexus 6P, expected on Google Pixels and everything else)

Diag diagram



/dev/diag

/dev/diag

Overview

- diagchar + diagfwd kernel drivers on Qualcomm MSM Android kernel

Functions

- Support the Diag interface
- Multiplex Diag channel to USB or memory device
- IOCTL interface to userland
- Masking of unnecessary Diag commands

Implementation

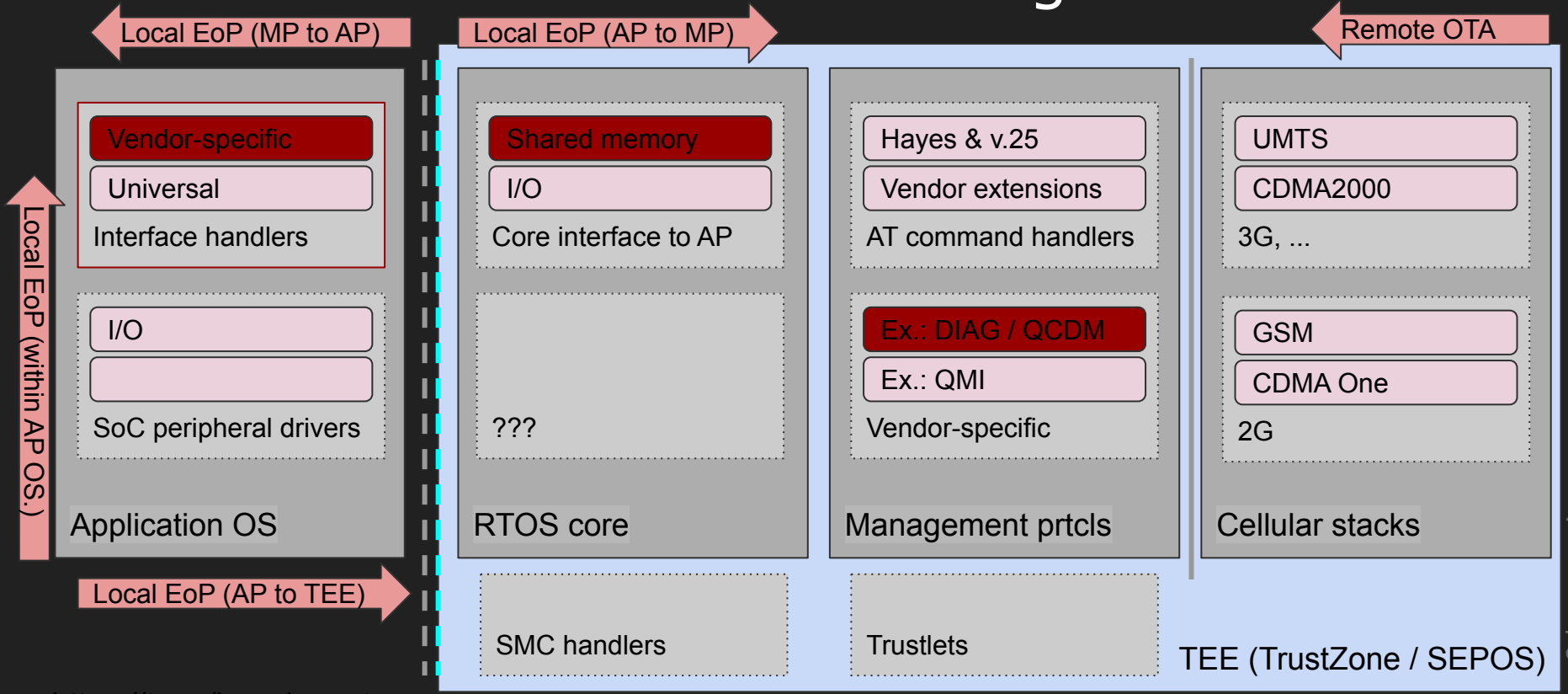
- Based on SMD/SMEM shared memory device (MSM specific)

drivers > char > diag > Kconfig

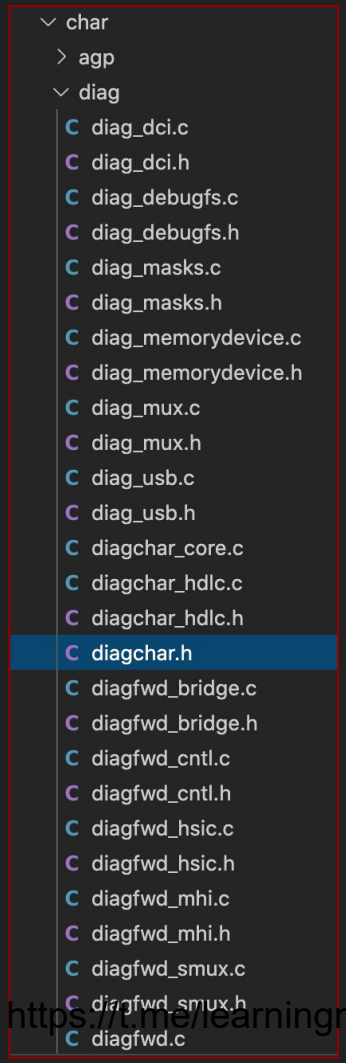
```
menu "Diag Support"

config DIAG_CHAR
    tristate "char driver interface and diag forwarding to/from modem"
    default m
    depends on USB_G_ANDROID || USB_FUNCTION_DIAG || USB_QCOM_MAEMO
    depends on ARCH_MSM
    select CRC_CCITT
    help
     Char driver interface for diag user space and diag-forwarding to modem ARM and back.
     This enables diagchar for maemo usb gadget or android usb gadget based on config selected.
endmenu

menu "DIAG traffic over USB"

config DIAG_OVER_USB
    bool "Enable DIAG traffic to go over USB"
        depends on ARCH_MSM
    default y
    help
     This feature helps segregate code required for DIAG traffic to go over USB.
endmenu

menu "HSIC/SMUX support for DIAG"

config DIAGFWD_BRIDGE_CODE
    depends on USB_QCOM_DIAG_BRIDGE
    default y
    bool "Enable QSC/9K DIAG traffic over SMUX/HSIC"
    help
     SMUX/HSIC Transport Layer for DIAG Router
endmenu

```

Note: INCOMPLETE!

Where is DIAG + diagchar?



diagchar key points




```
diag_ws_init();
ret = diag_real_time_info_ir
if (ret)
    goto fail;
ret = diag_debugfs_init();
if (ret)
    goto fail;
ret = diag_masks_init();
if (ret)
    goto fail;
ret = diag_mux_init();
if (ret)
    goto fail;
ret = diagfwd_init();
if (ret)
    goto fail;
ret = diag_remote_init();
if (ret)
    goto fail;
ret = diagfwd_bridge_init();
if (ret)
    goto fail;
ret = diagfwd_cntl_init();
if (ret)
    goto fail;
```

```
const struct file_operations diag_dbgfs_status_ops = {
    .read = diag_dbgfs_read_status,
};

const struct file_operations diag_dbgfs_table_ops = {
    .read = diag_dbgfs_read_table,
};

const struct file_operations diag_dbgfs_workpending_ops = {
    .read = diag_dbgfs_read_workpending,
};

const struct file_operations diag_dbgfs_mempool_ops = {
    .read = diag_dbgfs_read_mempool,
};

const struct file_operations diag_dbgfs_usbinfo_ops = {
    .read = diag_dbgfs_read_usbinfo,
};

const struct file
    .read = diag_
};

const struct file
    .read = diag_
};

int diag_debugfs_
{
```

```
/* Different IOCTL values */
#define DIAG_IOCTL_COMMAND_REG      0
#define DIAG_IOCTL_SWITCH_LOGGING   7
#define DIAG_IOCTL_GET_DELAYED_RSP_ID   8
#define DIAG_IOCTL_LSM_DEINIT       9
#define DIAG_IOCTL_DCI_INIT     20
#define DIAG_IOCTL_DCI_DEINIT       21
#define DIAG_IOCTL_DCI_SUPPORT      22
#define DIAG_IOCTL_DCI_REG      23
#define DIAG_IOCTL_DCI_STREAM_INIT  24
#define DIAG_IOCTL_DCI_HEALTH_STATS 25
#define DIAG_IOCTL_DCI_LOG_STATUS   26
#define DIAG_IOCTL_DCI_EVENT_STATUS 27
#define DIAG_IOCTL_DCI_CLEAR_LOGS   28
#define DIAG_IOCTL_DCI_CLEAR_EVENTS 29
#define DIAG_IOCTL_REMOTE_DEV       32
#define DIAG_IOCTL_VOTE_REAL_TIME   33
#define DIAG_IOCTL_GET_REAL_TIME    34
#define DIAG_IOCTL_PERIPHERAL_BUF_CONFIG    35
#define DIAG_IOCTL_PERIPHERAL_BUF_DRAIN     36
```

```
static const struct file_operations diagcharfops = {
    .owner = THIS_MODULE,
    .read = diagchar_read,
    .write = diagchar_write,
#ifdef CONFIG_COMPAT
    .compat_ioctl = diagchar_compat_ioctl,
#endif
    .unlocked_ioctl = diagchar_ioctl,
    .open = diagchar_open,
    .release = diagchar_close
};
```

diagchar and diag protocol

```
#define DIAG_CMD_VERSION    0
#define DIAG_CMD_DOWNLOAD   0x3A
#define DIAG_CMD_DIAG_SUBSYS    0x4B
#define DIAG_CMD_LOG_CONFIG 0x73
#define DIAG_CMD_LOG_ON_DMND    0x78
#define DIAG_CMD_EXT_BUILD  0x7c
#define DIAG_CMD_MSG_CONFIG 0x7D
#define DIAG_CMD_GET_EVENT_MASK 0x81
#define DIAG_CMD_S
#define DIAG_CMD_E
```

```
static int diag_switch_logging(int requested_mode)
{
    int success = -EINVAL;
    int temp = 0, status = 0;
    int new_mode = DIAG_USB_MODE; /* set the mode from diag_mux.h */
    int old_logging_id;

    switch (requested_mode) {
    case USB_MODE:
    case MEMORY_DEVICE_MODE:
    case NO_LOGGING_MODE:
    case UART_MODE:
    case SOCKET_MODE:
    case CALLBACK_MODE:
        break;
    default:
        pr_err("diag: In %s, request to switch to invalid mode: %d\n",
            __func__, requested_mode);
        return -EINVAL;
    }
```

```
int mask_request_validate(unsigned char mask_buf[])
{
    uint8_t packet_id;
    uint8_t subsys_id;
    uint16_t ss_cmd;

    packet_id = mask_buf[0];

    if (packet_id == 0x4B) {
        subsys_id = mask_buf[1];
        ss_cmd = *(uint16_t *)(mask_buf + 2);
        /* Packets with SSID which are allowed */
        switch (subsys_id) {
        case 0x04: /* DIAG_SUBSYS_WCDMA */
            if ((ss_cmd == 0) || (ss_cmd == 0xF))
                return 1;
            break;
        case 0x08: /* DIAG_SUBSYS_GSM */
            if ((ss_cmd == 0) || (ss_cmd == 0x1))
                return 1;
            break;
        case 0x09: /* DIAG_SUBSYS_UMTS */
        case 0x0F: /* DIAG_SUBSYS_CM */
            if (ss_cmd == 0)
                return 1;
            break;
        case 0x0C: /* DIAG_SUBSYS_OS */
            if ((ss_cmd == 2) || (ss_cmd == 0x100))
                return 1; /* MPU and APU */
            break;
        case 0x12: /* DIAG_SUBSYS_DIAG_SERV */
            if ((ss_cmd == 0) || (ss_cmd == 0x6) || (ss_cmd == 0x7))
                return 1;
            break;
        case 0x13: /* DIAG_SUBSYS_FS */
            if ((ss_cmd == 0) || (ss_cmd == 0x1))
                return 1;
            break;
        default:
            return 0;
            break;
        }
    } else {
```

Diagchar and shared memory (SMD)

```c
int diag_smd_constructor(struct diag_smd_info *smd_info, int peripheral,
                         int type)
{
    if (!smd_info)
        return -EIO;

    smd_info->peripheral = peripheral;
    smd_info->type = type;
    smd_info->encode_hdlc = 0;
    smd_info->inited = 0;
    mutex_init(&smd_info->smd_ch_mutex);
    spin_lock_init(&smd_info->in_busy_lock);

    switch (peripheral) {
    case MODEM_DATA:
        smd_info->peripheral_mask = DIAG_CON_MPSS;
        break;
    case LPASS_DATA:
        smd_info->peripheral_mask = DIAG_CON_LPASS;
        break;
    case WCNSS_DATA:
        smd_info->peripheral_mask = DIAG_CON_WCNSS;
        break;
    case SENSORS_DATA:
        smd_info->peripheral_mask = DIAG_CON_SENSORS;
        break;
    default:
        pr_er
        goto
    }
```

```c
int smd_read(smd_channel_t *ch, void *data, int len)
{
    if (!ch) {
        pr_err("%s: Invalid channel specified\n", __func__);
        return -ENODEV;
    }

    return ch->read(ch, data, len);
}
EXPORT_SYMBOL(smd_read);
```

```c
struct diag_smd_info {
    int peripheral; /* The peripheral this smd channel communicates wi
    int type;   /* The type of smd channel (data, control, dci) */
    uint16_t peripheral_mask;
    int encode_hdlc; /* Whether data is raw and needs to be hdlc encod

    smd_channel_t *ch;
    smd_channel_t *ch_save;

    struct mutex smd_ch_mutex;

    int in_busy_1;
    int in_busy_2;
    spinlock_t in_busy_lock;

    unsigned char *buf_in_1;
    unsigned char *buf_in_2;

    unsigned char *buf_in_1_raw;
    unsigned char *buf_in_2_raw;

    unsigned int buf_in_1_size;
    unsigned int buf_in_2_size;

    unsigned int buf_in_1_raw_size;
    unsigned int buf_in_2_raw_size;

    int buf_in_1_ctxt;
    int buf_in_2_ctxt;

    ruct workqueue_struct *wq;

    ruct work_struct diag_read_smd_work;
    ruct work_struct diag_notify_update_smd_work;
    t notify_context;
    ruct work_struct diag_general_smd_work;
    t general_context;
    nt8_t inited;


     Function ptr for function to call to process the data that
     was just read from the smd channel
    /
    t (*process_smd_read_data)(struct diag_smd_info *smd_info,
                               void *buf, int num_bytes);
};
```

25 results in 19 files - Open in editor

- C adsprpc.c drivers/char 1
  - VERIFY(err, 0 == smd_named_open_on_edge(
- C msm_smd_pkt.c drivers/char 1
  - r = smd_named_open_on_edge(smd_pkt_devp->ch_name,
- C diag_dci.c drivers/char/diag 2
  - err = smd_named_open_on_edge("DIAG_2",
  - err = smd_named_open_on_edge("DIAG_2_CMD",
- C diagfwd_cntl.c drivers/char/diag 1
  - r = smd_named_open_on_edge(channel_name,
- C diagfwd.c drivers/char/diag 2
  - r = smd_named_open_on_edge(channel_name,
  - r = smd_named_open_on_edge(channel_name,
- C radio-iris-transport.c drivers/media/radio 1
  - rc = smd_named_open_on_edge("APPS_FM", SMD_APPS_WCNSS,
- C wcnss_wlan.c drivers/net/wireless/wcnss 1
  - ret = smd_named_open_on_edge(WCNSS_CTRL_CHANNEL, SMD_AP...
- C ssm.c drivers/platform/msm 1
  - rc = smd_named_open_on_edge(ssm_drv->channel_name,
- C glink_smd_xprt.c drivers/soc/qcom 2
  - ret = smd_named_open_on_edge("GLINK_CTRL", einfo->smd_edge,
  - ret = smd_named_open_on_edge(ch->name, einfo->smd_edge, &ch-...

SMD/SMEM

Qualcomm Shared Memory Device

drivers > soc > qcom > C smem.c > msm_smem_probe(platform_device *)

```
1279
1280    smem_targ_info_legacy:
1281        SMEM_INFO("%s: reading dt-specified SMEM address\n", __func__);
1282        r = platform_get_resource_byname(pdev, IORESOURCE_MEM, "smem");
1283        if (r) {
1284            smem_ram_size = resource_size(r);
1285            smem_ram_phys = r->start;
1286        }
1287
1288    smem_targ_info_done:
1289        if (!smem_ram_phys || !smem_ram_size) {
1290            LOG_ERR("%s: Missing SMEM TARGET INFO\n", __func__);
1291            return -ENODEV;
1292        }
1293
1294        smem_ram_base = ioremap_nocache(smem_ram_phys, smem_ram_size);
1295
1296        if (!smem_ram_base) {
1297            LOG_ERR("%s: ioremap_nocache() of addr:%pa size: %pa\n",
1298                    __func__,
1299                    &smem_ram_phys, &smem_ram_size);
1300            return -ENODEV;
1301        }
1302
1303        if (!smem_initialized_check())
1304            return -ENODEV;
```

```
struct smd_channel {
    volatile void __iomem *send; /* some variant of smd_half_channel */
    volatile void __iomem *recv; /* some variant of smd_half_channel */
    unsigned char *send_data;
    unsigned char *recv_data;
    unsigned fifo_size;
    struct list_head ch_list;

    unsigned current_packet;
    unsigned n;
    void *priv;
    void (*notify)(void *priv, unsigned flags);

    int (*read)(smd_channel_t *ch, void *data, int len);
    int (*write)(smd_channel_t *ch, const void *data, int len,
            bool int_ntfy);
    int (*read_avail)(smd_channel_t *ch);
    int (*write_avail)(smd_channel_t *ch);
    int (*read_from_cb)(smd_channel_t *ch, void *data, int len);

    void (*update_state)(smd_channel_t *ch);
    unsigned last_state;
    void (*notify_other_cpu)(smd_channel_t *ch);
    void * (*read_from_fifo)(void *dest, const void *src, size_t num_bytes);
    void * (*write_to_fifo)(void *dest, const void *src, size_t num_bytes);

    char name[20];
    struct platform_device pdev;
    unsigned type;

    int pending_pkt_sz;

    char is_pkt_ch;

    /*
     * private internal functions to access *send and *recv.
     * never to be exported outside of smd
     */
    struct smd_half_channel_access *half_ch;
};
```

SMEM: entries and channels

46 results in 12 files - Open in editor

- hfi_response_handler.c drivers/media/platform/msm/vidc 1
  - smem_table_ptr = smem_get_entry(SMEM_IMAGE_VERSION_TABLE,
- venus_hfi.c drivers/media/platform/msm/vidc 1
  - smem_table_ptr = smem_get_entry(SMEM_IMAGE_VERSION_TABLE,
- msm_qpic_nand.c drivers/mtd/devices 2
  - (smem_get_entry(SMEM_AARM_PARTITION_TABLE, &len,
  - (smem_get_entry(SMEM_AARM_PARTITION_TABLE, &len,
- glink_smem_native_xprt.c drivers/soc/qcom 1
  - einfo->rx_fifo = smem_get_entry(SMEM_GLINK_NATIVE_XPRT_FIFO_1,
- pil-q6v5-mss.c drivers/soc/qcom 2
  - smem_reason = smem_get_entry_no_rlock(SMEM_SSR_REAS...
  - failure reason: (unknown, smem_get_entry_no_rlock failed).\n");
- smd_debug.c drivers/soc/qcom 5
  - buffer = smem_get_entry(fifo_base_id + n, &buffer_size,
  - tbl = smem_get_entry(ID_CH_ALLOC_TBL, &tbl_size, 0, SMEM_ANY_H...
  - tbl = smem_get_entry(SMEM_CHANNEL_ALLOC_TBL_2, &tbl_size, 0,
  - tbl = smem_get_entry(ID_CH_ALLOC_TBL, &tbl_size, i, 0);
  - tbl = smem_get_entry(SMEM_CHANNEL_ALLOC_TBL_2, &tbl_size, i, 0);
- smd.c drivers/soc/qcom 6
  - shared = smem_get_entry(ID_CH_ALLOC_TBL, &tbl_size,
  - shared = smem_get_entry(SMEM_CHANNEL_ALLOC_TBL_2, &tbl_size,
  - shared_pri = smem_get_entry(ID_CH_ALLOC_TBL, &pri_size, restar...
  - shared_sec = smem_get_entry(SMEM_CHANNEL_ALLOC_TBL_2, &sec...
  - buffer = smem_get_entry(fifo_id + ch->n, &buffer_sz,
  - SMD_INFO("smem_get_entry failed\n");
- smem_debug.c drivers/soc/qcom 1
  - data = smem_get_entry(SMEM_HW_SW_BUILD_ID, &size, 0,
- smem.c drivers/soc/qcom 17
  - identify the early callers of smem_get_entry()
  - * __smem_get_entry_nonsecure - Get pointer and size of existing SME...
  - static void *__smem_get_entry_nonsecure(unsigned id, unsigned *size,
  - * __smem_get_entry_secure - Get pointer and size of existing SMEM it...
  - static void *__smem_get_entry_secure(unsigned id,
  - return __smem_get_entry_nonsecure(id, size, skip_init_check,
  - ptr = __smem_get_entry_nonsecure(id, &size, skip_init_check, true);
  - ptr = smem_get_entry(id, &size, to_proc, flags);
  - ret = __smem_get_entry_secure(id, &size_out, to_proc, flags, true,
  - * smem_get_entry - Get existing item with security support
  - void *smem_get_entry(unsigned id, unsigned *size, unsigned to_proc,
  - return __smem_get_entry_secure(id, size, to_proc, flags, false, true);
  - EXPORT_SYMBOL(smem_get_entry);

https://t.me/learningnets

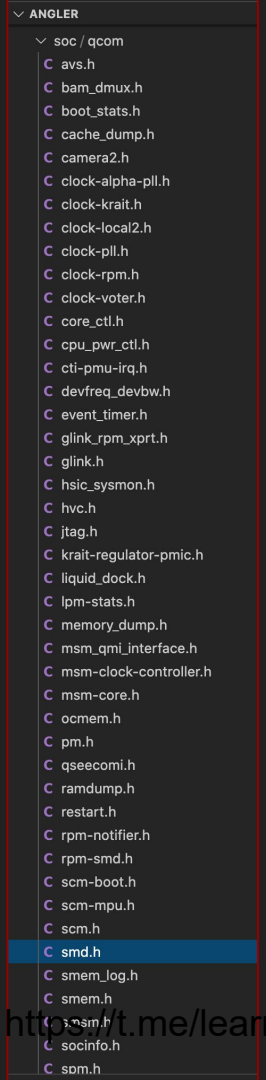
```c
/**
 * smd_alloc_channel() - Create and init local structures for a newly allocated
 *          SMD channel
 *
 * @alloc_elm: the allocation element stored in SMEM for this channel
 * @table_id: the id of the table this channel resides in. 1 = first table, 2 =
 *      seconds table, etc
 * @r_info: pointer to the info structure of the remote proc for this channel
 * @returns: error code for failure; 0 for success
 */
static int smd_alloc_channel(struct smd_alloc_elm *alloc_elm, int table_id,
                             struct remote_proc_info *r_info)
{
    struct smd_channel *ch;

    ch = kzalloc(sizeof(struct smd_channel), GFP_KERNEL);
    if (ch == 0) {
        pr_err("smd_alloc_channel() out of memory\n");
        return -ENOMEM;
    }
    ch->n = alloc_elm->cid;
    ch->type = SMD_CHANNEL_TYPE(alloc_elm->type);

    if (smd_alloc(ch, table_id, r_info)) {
        kfree(ch);
        return -ENODEV;
    }

    /* probe_worker guarentees ch->type will be a valid type */
    if (ch->type == SMD_APPS_MODEM)
        ch->notify_other_cpu = notify_modem_smd;
    else if (ch->type == SMD_APPS_QDSP)
        ch->notify_other_cpu = notify_dsp_smd;
    else if (ch->type == SMD_APPS_DSPS)
        ch->notify_other_cpu = notify_dsps_smd;
    else if (ch->type == SMD_APPS_WCNSS)
        ch->notify_other_cpu = notify_wcnss_smd;
    else if (ch->type == SMD_APPS_Q6FW)
        ch->notify_other_cpu = notify_modemfw_smd;
    else if (ch->type == SMD_APPS_RPM)
        ch->notify_other_cpu = notify_rpm_smd;
```

```c
if (smd_is_packet(alloc_elm)) {
    ch->read = smd_packet_read;
    ch->write = smd_packet_write;
    ch->read_avail = smd_packet_read_avail;
    ch->write_avail = smd_packet_write_avail;
    ch->update_state = update_packet_state;
    ch->read_from_cb = smd_packet_read_from_cb;
    ch->is_pkt_ch = 1;
} else {
    ch->read = smd_stream_read;
    ch->write = smd_stream_write;
    ch->read_avail = smd_stream_read_avail;
    ch->write_avail = smd_stream_write_avail;
    ch->update_state = update_stream_state;
    ch->read_from_cb = smd_stream_read;
}

if (is_word_access_ch(ch->type)) {
    ch->read_from_fifo = smd_memcpy32_from_fifo;
    ch->write_to_fifo = smd_memcpy32_to_fifo;
} else {
    ch->read_from_fifo = smd_memcpy_from_fifo;
    ch->write_to_fifo = smd_memcpy_to_fifo;
}

smd_memcpy_from_fifo(ch->name, alloc_elm->name, SMD_MAX_CH_NAME_LEN);
ch->name[SMD_MAX_CH_NAME_LEN-1] = 0;

ch->pdev.name = ch->name;
ch->pdev.id = ch->type;

SMD_INFO("smd_alloc_channel() '%s' cid=%d\n",
    ch->name, ch->n);

mutex_lock(&smd_creation_mutex);
list_add(&ch->ch_list, &smd_ch_closed_list);
mutex_unlock(&smd_creation_mutex);
```

0days.engineer

Other stuff

Qualcomm SoC drivers




```c
static int jtag_fuse_probe(struct platform_device *pdev)
{
    struct device *dev = &pdev->dev;
    struct fuse_drvdata *drvdata;
    struct resource *res;
    const struct of_device_id *match;

    drvdata = devm_kzalloc(dev, sizeof(*drvdata), GFP_KERNEL);
    if (!drvdata)
        return -ENOMEM;
    /* Store the driver data pointer for use in exported functions */
    fusedrvdata = drvdata;
    drvdata->dev = &pdev->dev;
    platform_set_drvdata(pdev, drvdata);

    match = of_match_device(jtag_fuse_match, dev);
    if (!match)
        return -EINVAL;

    if (!strcmp(match->compatible, JTAG_FUSE_VERSION_V2))
        drvdata->fuse_v2 = true;
    else if (!strcmp(match->compatible, JTAG_FUSE_VERSION_V3))
        drvdata->fuse_v3 = true;

    res = platform_get_resource_byname(pdev, IORESOURCE_MEM, "fuse-base");
    if (!res)
        return -ENODEV;

    drvdata->base = devm_ioremap(dev, res->start, resource_size(res));
    if (!drvdata->base)
        return -ENOMEM;

    dev_info(dev, "JTag Fuse initialized\n");
    return 0;
}
```

```c
#define fuse_writel(drvdata, val, off)  __raw_writel((val), drvdata->base + off)
#define fuse_readl(drvdata, off)    __raw_readl(drvdata->base + off)

#define OEM_CONFIG0         (0x000)
#define OEM_CONFIG1         (0x004)
#define OEM_CONFIG2         (0x008)

/* JTAG FUSE V1 */
#define ALL_DEBUG_DISABLE       BIT(21)
#define APPS_DBGEN_DISABLE      BIT(0)
#define APPS_NIDEN_DISABLE      BIT(1)
#define APPS_SPIDEN_DISABLE     BIT(2)
#define APPS_SPNIDEN_DISABLE        BIT(3)
#define DAP_DEVICEEN_DISABLE        BIT(8)

/* JTAG FUSE V2 */
#define ALL_DEBUG_DISABLE_V2        BIT(0)
#define APPS_DBGEN_DISABLE_V2       BIT(10)
#define APPS_NIDEN_DISABLE_V2       BIT(11)
#define APPS_SPIDEN_DISABLE_V2      BIT(12)
#define APPS_SPNIDEN_DISABLE_V2 BIT(13)
#define DAP_DEVICEEN_DISABLE_V2 BIT(18)

/* JTAG FUSE V3 */
#define ALL_DEBUG_DISABLE_V3        BIT(29)
#define APPS_DBGEN_DISABLE_V3       BIT(8)
#define APPS_NIDEN_DISABLE_V3       BIT(21)
#define APPS_SPIDEN_DISABLE_V3      BIT(5)
#define APPS_SPNIDEN_DISABLE_V3 BIT(31)
#define DAP_DEVICEEN_DISABLE_V3 BIT(7)

#define JTAG_FUSE_VERSION_V1        "qcom,jtag-fuse"
#define JTAG_FUSE_VERSION_V2        "qcom,jtag-fuse-v2"
#define JTAG_FUSE_VERSION_V3        "qcom,jtag-fuse-v3"
```

Diag protocol

Reverse-engineering the reverse-engineers

- SnoopSnitch (open source)
- Can enable protocol dumps on rooted devices
- Sends an obscure blob of QCDM commands through the harnessed /dev/diag interface 👉
- Changes baseband firmware configuration
- Can you explain what exactly this commands blob does to your mobile phone?
- I got curious

SnoopSnitch > app > src > main > java > de > srlabs > snoopsnitch > qdmon > SetupLoggingCmds.java

```
1   package de.srlabs.snoopsnitch.qdmon;
2
3   class SetupLoggingCmds {
4       static final byte[][] cmds = {
5               {(byte) 0x1D},
6               {(byte) 0x00},
7               {(byte) 0x7C},
8               {(byte) 0x0C},
9               {(byte) 0x63},
10              {(byte) 0x60, (byte) 0x00},
11              {(byte) 0x73, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x03, (byte
12              {(byte) 0x73, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x03, (byte
13              {(byte) 0x73, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x03, (byte
14              {(byte) 0x73, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x03, (byte
15              {(byte) 0x73, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x03, (byte
16              {(byte) 0x73, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x03, (byte
17              {(byte) 0x73, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x03, (byte
18              {(byte) 0x7D, (byte) 0x04, (byte) 0x00, (byte) 0x00, (byte) 0x61, (byte
19              {(byte) 0x7D, (byte) 0x04, (byte) 0xF4, (byte) 0x01, (byte) 0xFA, (byte
20              {(byte) 0x7D, (byte) 0x04, (byte) 0xE8, (byte) 0x03, (byte) 0xEF, (byte
21              {(byte) 0x7D, (byte) 0x04, (byte) 0xD0, (byte) 0x07, (byte) 0xD8, (byte
22              {(byte) 0x7D, (byte) 0x04, (byte) 0xB8, (byte) 0x0B, (byte) 0xC6, (byte
23              {(byte) 0x7D, (byte) 0x04, (byte) 0xA0, (byte) 0x0F, (byte) 0xAA, (byte
24              {(byte) 0x7D, (byte) 0x04, (byte) 0x94, (byte) 0x11, (byte) 0xAE, (byte
25              {(byte) 0x7D, (byte) 0x04, (byte) 0xF8, (byte) 0x11, (byte) 0x05, (byte
26              {(byte) 0x7D, (byte) 0x04, (byte) 0x88, (byte) 0x13, (byte) 0xA5, (byte
27              {(byte) 0x7D, (byte) 0x04, (byte) 0x7C, (byte) 0x15, (byte) 0x8C, (byte
28              {(byte) 0x7D, (byte) 0x04, (byte) 0x70, (byte) 0x17, (byte) 0xC0, (byte
29              {(byte) 0x7D, (byte) 0x04, (byte) 0x64, (byte) 0x19, (byte) 0x79, (byte
30              {(byte) 0x7D, (byte) 0x04, (byte) 0x58, (byte) 0x1B, (byte) 0x5B, (byte
31              {(byte) 0x7D, (byte) 0x04, (byte) 0xBC, (byte) 0x1B, (byte) 0xC7, (byte
```

Diag commands: known and unknown

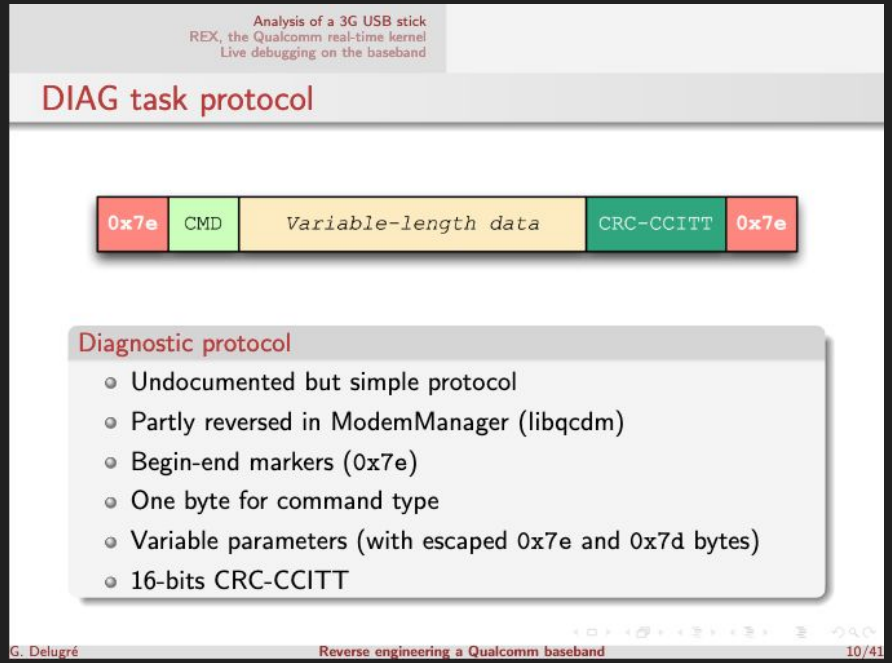
```
18  #ifndef LIBQCDM_DM_COMMANDS_H
19  #define LIBQCDM_DM_COMMANDS_H
20
21  enum {
22      DIAG_CMD_VERSION_INFO = 0,  /* Version info */
23      DIAG_CMD_ESN          = 1,  /* ESN */
24      DIAG_CMD_PEEKB        = 2,  /* Peek byte */
25      DIAG_CMD_PEEKW        = 3,  /* Peek word */
26      DIAG_CMD_PEEKD        = 4,  /* Peek dword */
27      DIAG_CMD_POKEB        = 5,  /* Poke byte */
28      DIAG_CMD_POKEW        = 6,  /* Poke word */
29      DIAG_CMD_POKED        = 7,  /* Poke dword */
30      DIAG_CMD_OUTP         = 8,  /* Byte output */
31      DIAG_CMD_OUTPW        = 9,  /* Word output */
32      DIAG_CMD_INP          = 10, /* Byte input */
33      DIAG_CMD_INPW         = 11, /* Word input */
34      DIAG_CMD_STATUS       = 12, /* Station status */
35      DIAG_CMD_LOGMASK      = 15, /* Set logging mask */
36      DIAG_CMD_LOG          = 16, /* Log packet */
37      DIAG_CMD_NV_PEEK      = 17, /* Peek NV memory */
38      DIAG_CMD_NV_POKE      = 18, /* Poke NV memory */
```

```
 99      DIAG_CMD_RPC              = 100, /* Used for RPC */
100      DIAG_CMD_GET_PROPERTY     = 101,
101      DIAG_CMD_PUT_PROPERTY     = 102,
102      DIAG_CMD_GET_GUID         = 103, /* GUID requests */
103      DIAG_CMD_USER_CMD         = 104, /* User callbacks */
104      DIAG_CMD_GET_PERM_PROPERTY = 105,
105      DIAG_CMD_PUT_PERM_PROPERTY = 106,
106      DIAG_CMD_PERM_USER_CMD    = 107, /* Permanent user callbacks */
107      DIAG_CMD_GPS_SESS_CTRL    = 108, /* GPS session control */
108      DIAG_CMD_GPS_GRID         = 109, /* GPS search grid */
109      DIAG_CMD_GPS_STATISTICS   = 110,
110      DIAG_CMD_TUNNEL           = 111, /* Tunneling command code */
111      DIAG_CMD_RAM_RW           = 112, /* Calibration RAM control using DM */
112      DIAG_CMD_CPU_RW           = 113, /* Calibration CPU control using DM */
113      DIAG_CMD_SET_FTM_TEST_MODE = 114, /* Field (or Factory?) Test Mode */
114      DIAG_CMD_LOG_CONFIG       = 115, /* New logging config command */
115      DIAG_CMD_EXT_BUILD_ID     = 124,
116      DIAG_CMD_EXT_MESSAGE_CONFIG= 125,
117      DIAG_CMD_EVENT_GET_MASK   = 129,
118      DIAG_CMD_EVENT_SET_MASK   = 130,
119      DIAG_CMD_SAMSUNG_IND      = 217, /* Unsolicited message seen on Samsung Z810 */
120  };
121
```

?

Command 146 (0x92) = qshrinked log message hash

Diag command format



0x7e (126) is another DIAG token, not a delimiter

```
def crc16( self, data ):
    crc = 0xffff
    for byte in data:
        crc = (crc >> 8) ^ CRC16_CCITT_TABLE[ (crc ^ ord(byte)) & 0xff ]
    return crc ^ 0xffff

def escape( self, data ):
    data = data.replace("\x7d", "\x7d\x5d")
    data = data.replace("\x7e", "\x7d\x5e")
    return data;

def unescape( self ):
    self.data = self.data.replace("\x7d\x5d", "\x7d")
    self.data = self.data.replace("\x7d\x5e", "\x7e")
```

Diag subsystems

- Diagnostic system extensions for baseband subsystems
  - ~100 subsystems + OEM reserved
  - Subsystems may register their custom handlers with DIAG task
  - Packet is opaque
- DIAG_CMD_SUBSYS = 75
- struct {
  - u8 subsys_id;
  - u16 subsys_cmd;
  - payload (variable len) }

```
/* WCDMA subsystem command codes */
enum {
    DIAG_SUBSYS_WCDMA_CALL_START  = 12, /* Starts a call */
    DIAG_SUBSYS_WCDMA_CALL_END    = 13, /* Ends an ongoing call */
    DIAG_SUBSYS_WCDMA_STATE_INFO  = 15, /* Gets WCDMA state */
};

/* HDR subsystem command codes */
enum {
    DIAG_SUBSYS_HDR_STATE_INFO  = 8, /* Gets EVDO state */
};

/* GSM subsystem command codes */
enum {
    DIAG_SUBSYS_GSM_STATE_INFO  = 1, /* Gets GSM state */
};
```

Diag subsystems (2)

```
/* Subsystem IDs used with DIAG_CMD_SUBSYS; these often obsolete many of
 * the original DM commands.
 */
enum {
    DIAG_SUBSYS_WCDMA           = 4,
    DIAG_SUBSYS_HDR             = 5,  /* High Data Rate (ie, EVDO) */
    DIAG_SUBSYS_GSM             = 8,
    DIAG_SUBSYS_UMTS            = 9,
    DIAG_SUBSYS_OS              = 12,
    DIAG_SUBSYS_GPS             = 13,
    DIAG_SUBSYS_SMS             = 14, /* Wireless Messaging Service */
    DIAG_SUBSYS_CM              = 15, /* Call manager */
    DIAG_SUBSYS_FS              = 19, /* File System (EFS2) */
    DIAG_SUBSYS_NOVATEL_6500    = 50, /* for Novatel Wireless MSM6500-based devices */
    DIAG_SUBSYS_LTE             = 68,
    DIAG_SUBSYS_ZTE             = 101, /* for ZTE EVDO devices */
    DIAG_SUBSYS_NOVATEL_6800    = 250 /* for Novatel Wireless MSM6800-based devices */
};
```

libqcdm

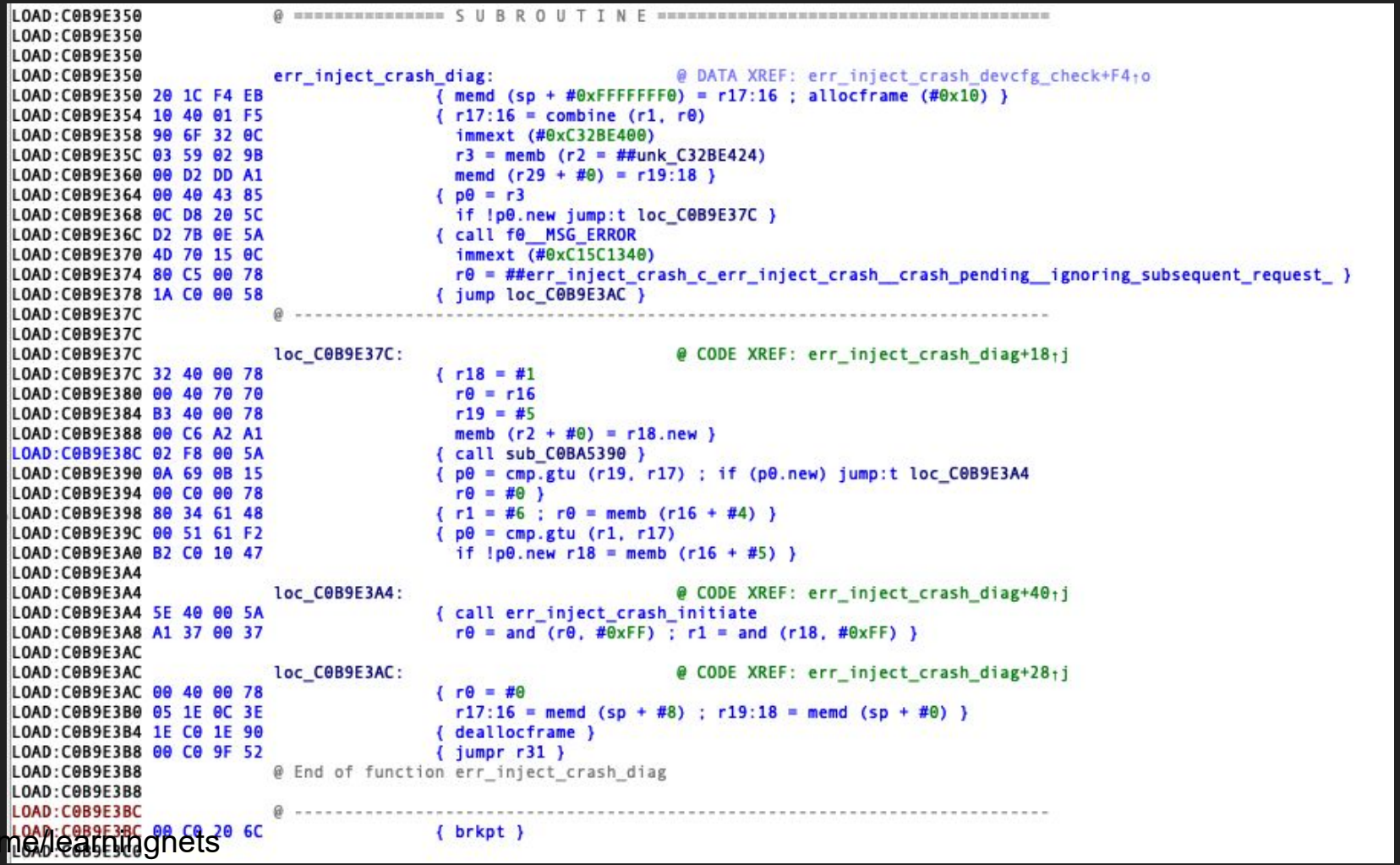
More:

18 = DIAG subsystem subsystem

37 = DEBUG

250+ = OEMs

DEBUG subsystem?



```
LOAD:C0B9E350                @ =============== S U B R O U T I N E =======================================
LOAD:C0B9E350
LOAD:C0B9E350
LOAD:C0B9E350                err_inject_crash_diag:                  @ DATA XREF: err_inject_crash_devcfg_check+F4↑o
LOAD:C0B9E350 20 1C F4 EB                    { memd (sp + #0xFFFFFFF0) = r17:16 ; allocframe (#0x10) }
LOAD:C0B9E354 10 40 01 F5                    { r17:16 = combine (r1, r0)
LOAD:C0B9E358 90 6F 32 0C                      immext (#0xC32BE400)
LOAD:C0B9E35C 03 59 02 9B                      r3 = memb (r2 = ##unk_C32BE424)
LOAD:C0B9E360 00 D2 DD A1                      memd (r29 + #0) = r19:18 }
LOAD:C0B9E364 00 40 43 85                    { p0 = r3
LOAD:C0B9E368 0C D8 20 5C                      if !p0.new jump:t loc_C0B9E37C }
LOAD:C0B9E36C D2 7B 0E 5A                    { call f0__MSG_ERROR
LOAD:C0B9E370 4D 70 15 0C                      immext (#0xC15C1340)
LOAD:C0B9E374 80 C5 00 78                      r0 = ##err_inject_crash_c_err_inject_crash__crash_pending__ignoring_subsequent_request_ }
LOAD:C0B9E378 1A C0 00 58                    { jump loc_C0B9E3AC }
LOAD:C0B9E37C                @ ---------------------------------------------------------------------------
LOAD:C0B9E37C
LOAD:C0B9E37C                loc_C0B9E37C:                           @ CODE XREF: err_inject_crash_diag+18↑j
LOAD:C0B9E37C 32 40 00 78                    { r18 = #1
LOAD:C0B9E380 00 40 70 70                      r0 = r16
LOAD:C0B9E384 B3 40 00 78                      r19 = #5
LOAD:C0B9E388 00 C6 A2 A1                      memb (r2 + #0) = r18.new }
LOAD:C0B9E38C 02 F8 00 5A                    { call sub_C0BA5390 }
LOAD:C0B9E390 0A 69 0B 15                    { p0 = cmp.gtu (r19, r17) ; if (p0.new) jump:t loc_C0B9E3A4
LOAD:C0B9E394 00 C0 00 78                      r0 = #0 }
LOAD:C0B9E398 80 34 61 48                    { r1 = #6 ; r0 = memb (r16 + #4) }
LOAD:C0B9E39C 00 51 61 F2                    { p0 = cmp.gtu (r1, r17)
LOAD:C0B9E3A0 B2 C0 10 47                      if !p0.new r18 = memb (r16 + #5) }
LOAD:C0B9E3A4
LOAD:C0B9E3A4                loc_C0B9E3A4:                           @ CODE XREF: err_inject_crash_diag+40↑j
LOAD:C0B9E3A4 5E 40 00 5A                    { call err_inject_crash_initiate
LOAD:C0B9E3A8 A1 37 00 37                      r0 = and (r0, #0xFF) ; r1 = and (r18, #0xFF) }
LOAD:C0B9E3AC
LOAD:C0B9E3AC                loc_C0B9E3AC:                           @ CODE XREF: err_inject_crash_diag+28↑j
LOAD:C0B9E3AC 00 40 00 78                    { r0 = #0
LOAD:C0B9E3B0 05 1E 0C 3E                      r17:16 = memd (sp + #8) ; r19:18 = memd (sp + #0) }
LOAD:C0B9E3B4 1E C0 1E 90                    { deallocframe }
LOAD:C0B9E3B8 00 C0 9F 52                    { jumpr r31 }
LOAD:C0B9E3B8                @ End of function err_inject_crash_diag
LOAD:C0B9E3B8
LOAD:C0B9E3BC                @ ---------------------------------------------------------------------------
LOAD:C0B9E3BC 00 C0 20 6C                    { brkpt }
```

Crash injection via Diag

```
LOAD:C0B9E3C0
LOAD:C0B9E3C0                 err_inject_crash_execute:                 @ CODE XREF: err_inject_crash_initiate:loc_C0B9E4AC↓p
LOAD:C0B9E3C0                                                           @ DATA XREF: err_inject_crash_devcfg_check+130↑o
LOAD:C0B9E3C0 10 1C 01 D1                 { r1 = memub (r0 + #1) ; allocframe (#8) }
LOAD:C0B9E3C4 0C 60 41 10                 { p0 = cmp.eq (r1, #0) ; if (!p0.new) jump:t loc_C0B9E3DC
LOAD:C0B9E3C8 00 D0 DD A1                   memd (r29 + #0) = r17:16 }
LOAD:C0B9E3CC 32 C0 20 5C                 { if !p0 jump default }
LOAD:C0B9E3D0 70 42 00 5A                 { call sub_C0B9E8B0
LOAD:C0B9E3D4 F1 6A 45 0C                   immext (#0xC45ABC40)
LOAD:C0B9E3D8 00 C0 00 78                   r0 = ##dword_C45ABC40 }
LOAD:C0B9E3DC
LOAD:C0B9E3DC                 loc_C0B9E3DC:                             @ CODE XREF: err_inject_crash_execute+4↑j
LOAD:C0B9E3DC 1C C1 01 10                 { p0 = cmp.eq (r1, #1) ; if (p0.new) jump:nt ERR_INJECT_WDOG_TIMEOUT }
LOAD:C0B9E3E0 1E 62 01 10                 { p0 = cmp.eq (r1, #2) ; if (p0.new) jump:t ERR_INJECT_NULLPTR
LOAD:C0B9E3E4 90 6F 32 0C                   immext (#0xC32BE400)
LOAD:C0B9E3E8 10 C5 00 78                   r16 = ##unk_C32BE428 }
LOAD:C0B9E3EC 22 E3 41 10                 { p0 = cmp.eq (r1, #3) ; if (!p0.new) jump:t default }
LOAD:C0B9E3F0 98 60 16 5B                 { call sub_C0452520
LOAD:C0B9E3F4 81 00 70 48                   r0 = #7 ; r1 = memw (r16 + #0) }
LOAD:C0B9E3F8 00 40 40 8B                 { r0 = convert_w2sf (r0)
LOAD:C0B9E3FC 01 C0 90 91                   r1 = memw (r16 + #0) }
LOAD:C0B9E400 22 40 60 8B                 { r2 = convert_sf2uw (r0):chop
LOAD:C0B9E404 D8 42 18 5B                   call log_0
LOAD:C0B9E408 4E 70 15 0C                   immext (#0xC15C1380)
LOAD:C0B9E40C 80 C4 00 78                   r0 = ##err_inject_crash_c_Should_have_crashed_due_to_div0__7__u__d_ }
LOAD:C0B9E410 16 C0 00 58                 { jump loc_C0B9E43C }
LOAD:C0B9E414                 @ ---------------------------------------------------------------------------
LOAD:C0B9E414
LOAD:C0B9E414                 ERR_INJECT_WDOG_TIMEOUT:                  @ CODE XREF: err_inject_crash_execute:loc_C0B9E3DC↑j
LOAD:C0B9E414 FE FC FF 5B                 { call sub_C0B9DE10 }
LOAD:C0B9E418 12 C0 00 58                 { jump loc_C0B9E43C }
LOAD:C0B9E41C                 @ ---------------------------------------------------------------------------
LOAD:C0B9E41C
LOAD:C0B9E41C                 ERR_INJECT_NULLPTR:                       @ CODE XREF: err_inject_crash_execute+20↑j
LOAD:C0B9E41C 4E 70 15 0C                 { immext (#0xC15C1380)
LOAD:C0B9E420 81 00 C0 49                   r0 = ##err_inject_crash_c_Should_have_crashed_due_to_null_access____x____x_ ; r1 = memw (r16 + #0) }
LOAD:C0B9E424 C6 42 18 5B                 { call log_0
LOAD:C0B9E428 81 00 12 00                   r2 = memw (r1 + #0) ; r1 = memw (r16 + #0) }
LOAD:C0B9E42C 08 C0 00 58                 { jump loc_C0B9E43C }
LOAD:C0B9E430                 @ ---------------------------------------------------------------------------
LOAD:C0B9E430
LOAD:C0B9E430                 default:                                  @ CODE XREF: err_inject_crash_execute+C↑j
LOAD:C0B9E430                                                           @ err_inject_crash_execute+2C↑j
LOAD:C0B9E430 10 58 17 5B                 { call log_md5
LOAD:C0B9E434 4E 70 15 0C                   immext (#0xC15C1380)
LOAD:C0B9E438 80 C2 00 78                   r0 = ##err_inject_crash_c_err_inject_crash__unhandled_crash_ID___x_ }
LOAD:C0B9E43C
LOAD:C0B9E43C                 loc_C0B9E43C:                             @ CODE XREF: err_inject_crash_execute+50↑j
LOAD:C0B9E43C                                                           @ err_inject_crash_execute+58↑j ...
LOAD:C0B9E43C 00 40 00 78                 { r0 = #0
LOAD:C0B9E440 10 40 DD 91                   r17:16 = memd (r29 + #0)
LOAD:C0B9E444 90 6F 32 0C                   immext (#0xC32BE400)
LOAD:C0B9E448 24 C4 A0 48                   memb (##unk_C32BE424) = r0.new }
LOAD:C0B9E44C 1E C0 1E 90                 { deallocframe }
LOAD:C0B9E450 00 C0 9F 52                 { jumpr r31 }
LOAD:C0B9E450                 @ End of function err_inject_crash_execute
LOAD:C0B9E450
```

Examples

SetupLoggingCmds.jar

```
class SetupLoggingCmds {
    static final byte[][] cmds = {
            {(byte) 0x1D}, // DIAG_CMD_TIMESTAMP
            {(byte) 0x00}, // DIAG_CMD_VERSION_INFO
            {(byte) 0x7C}, // DIAG_CMD_EXT_BUILD_ID
            {(byte) 0x0C}, // DIAG_CMD_STATUS
            {(byte) 0x63}, // DIAG_CMD_STATUS_SNAPSHOT
            {(byte) 0x60, (byte) 0x00}, // DIAG_CMD_EVENT_REPORT
            {(byte) 0x73, (byte) 0x00, ... , (byte) 0x00}, // DIAG_CMD_LOG_CONFIG
            // more entries of DIAG_CMD_LOG_CONFIG
            {(byte) 0x7D, (byte) 0x04, ... , (byte) 0x00}, // DIAG_CMD_EXT_MESSAGE_CONFIG
            // more entries of DIAG_CMD_EXT_MESSAGE_CONFIG
            {(byte) 0x60, (byte) 0x00} // DIAG_CMD_EVENT_REPORT
    };
}
```

How do cellular protocol dumps work?

DIAG_CMD_LOG_CONFIG (0x73)

- Enable cellular protocol dumps
- Command format is partially documented in libqcdm 👇
- Op = subcommand
- Equipment ID
  - GSM (5), UMTS (7), TDSCDMA (13)

```
struct DMCmdLogConfig {
    uint8_t code;
    uint8_t pad[3];
    uint32_t op;
    uint32_t equipid;
    uint32_t num_items;
    uint8_t mask[0];
} __attribute__ ((packed));
typedef struct DMCmdLogConfig DMCmdLogConfig;
```

libqcdm

Example decode of a command packet (SnoopSnitch)

```
{(byte) 0x73,                          // DIAG_CMD_LOG_CONFIG
(byte) 0x00, (byte) 0x00, (byte) 0x00,     // pad[3]
// subcommand = set mask:
(byte) 0x03, (byte) 0x00, (byte) 0x00, (byte) 0x00,
// equipment id:
(byte) 0x0D, (byte) 0x00, (byte) 0x00, (byte) 0x00,
// ???
(byte) 0x07, (byte) 0x02, (byte) 0x00, (byte) 0x00,
// mask
(byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00,
(byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00,
(byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00,
(byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00,
(byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00,
(byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00,
(byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00,
(byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00,
(byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00,
(byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00,
(byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00,
(byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00,
(byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00,
(byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00,
(byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00,
(byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00,
(byte) 0x00},
```

What about message logs?

DIAG_CMD_EXT_MESSAGE_CONFIG (0x7D)

- Enable/disable and configure textual message logging
- Command format is **undocumented**
- Operates on subcommands
  - Set logging mask
- Logging mask
  - Applied bitwise& against reported loglevel of message
  - Mask 0x0 disables all logging, 0xFFFFFFFF enables all
- SSID
  - Filter by subsystem ID
  - Not same as DIAG subsystems

Example decode of command packet (SnoopSnitch)

```
{(byte) 0x7D, // DIAG_CMD_EXT_MESSAGE_CONFIG
(byte) 0x04,  // subcommand: set mask
(byte) 0x40, (byte) 0x1F,  // ssid start
(byte) 0x40, (byte) 0x1F,  // ssid end
(byte) 0x00, (byte) 0x00,  // pad
(byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00}, // mask
```

Enable all text message logging on all SSIDs (my code)

```
def logging_enable_all():
    config = []
    ssid = 0
    while ( ssid < 0x2888 ):
        cmd = '\x7d\x04'
        cmd += struct.pack('<H', ssid);
        cmd += struct.pack('<H', ssid);
        cmd += struct.pack('<H', 0);
        cmd += '\xff\xff\xff\xff'
        ssid += 1
        config.append(cmd);
    return config
```

Parsing incoming log messages

DIAG_CMD_LOG_MESSAGE (0x79/121)

- u8 cmd = 0x79
- u8 type
- u8 n_args
- u8 dropcount
- u64 timestamp
- u16 ssid
- u16 line
- u32 unknown
- … args: n_args-1 * u32
- **ASCII log message string**

DIAG_CMD_LOG_HASH (0x92/146)

- u8 cmd = 0x92
- u8 type
- u8 n_args
- u8 dropcount
- u64 timestamp
- u16 ssid
- u16 line
- u32 unknown
- **u32 md5 hash of log message**
- …args

```
# DIAG_CMD_LOG_MESSAGE:
# 79000300 - 009c41af 7f3de700 - 6c00 - 0000 - 04000000 - 4c010000 00000000 00000000
646566696e65645f666561747572653a2066756e6374696f6e20656e747279207769746820666561747572652020564006e765f746f5f676c6f62616c2e6300eb317e
# 79000000 - 0030bf4f 933de700 - 9d02 - 5100 - 04000000 - - 5450433a204c61743a3130312e3033303030302c204c6f6e3a3130312e3438393939382c20416c743a30 ...

# DIAG_CMD_LOG_HASH:
# 92000300 - 0058ae52 153de700 - 900f - 0000 - 04000000 - 4a067d5d - 0f050000 00120000 00010000 00 0fe5 7e
```

Crash injection

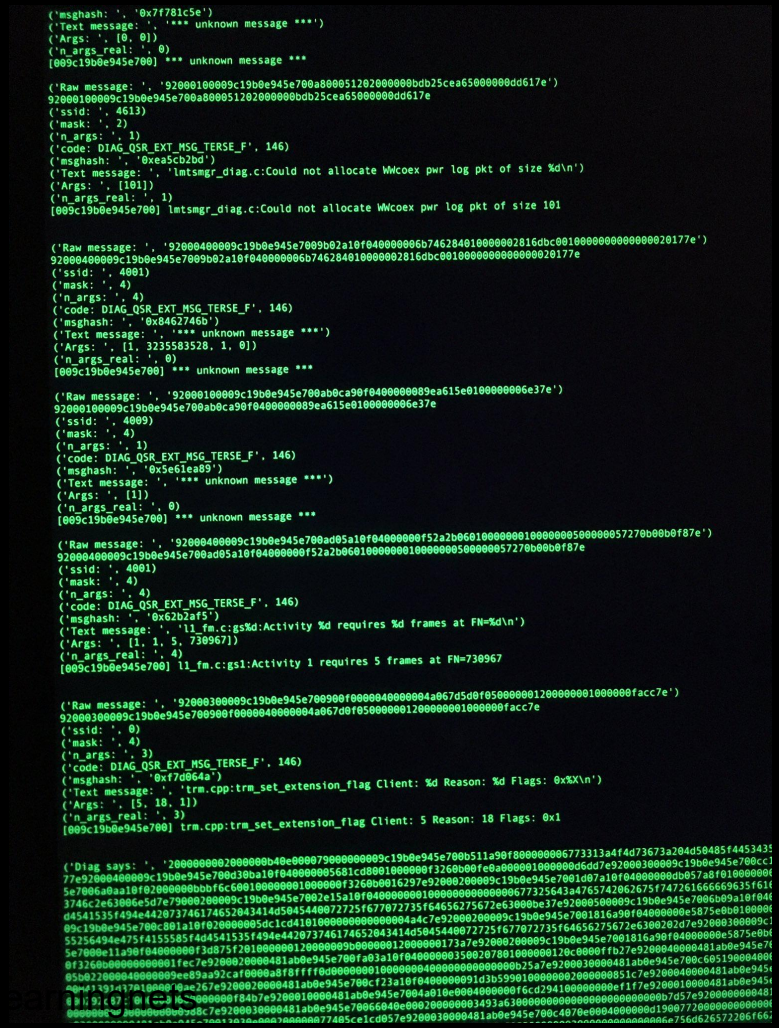
```
4B // DIAG_CMD_SUBSYS
25 // subsystem ID: DEBUG
03 00 // subcommand: core dump
00 // crash type
00 // crash delay
```

Crash types:
0 = halt/panic,
1 = watchdog timeout,
2 = nullptr access,
3 = divide by 0
exception

- Didn’t work on my Nexus
- On production devices this is expected to reboot the phone, no dumps left
- Might (and probably should) work on very old or obscure Qualcomm modems

diagtalk

Decoded diag logs



Conclusions

Future work

Qualcomm Hexagon

- Other Qualcomm-proprietary diagnostic protocols
- QEMU Hexagon (modern!) full system emulation
- Software-based debugger for prod-fused devices
- JTAG fusing bypass
- OTA vectors
- MP -> AP escalation
- Decompiler
- Binary patch diffing

Basebands: community research

- CDMA BTS software implementation
- Update osmocommBB a little bit
- **Open-source cellular ecosystem for the open spectrum**
  - Technically, nothing prevents cellular protocols from operating on unlicensed radio bands
  - But basebands are locked to radio bands via hardware capabilities + firmware programming
  - Some mobile phones seem to support 2.4Ghz/5Ghz (bb) on paper
  - So it should be possible to reconfigure the baseband radio layer
  - BTS on SDR makes no assumptions

References

[CCC2011] Reverse engineering a Qulacomm baseband
https://fahrplan.events.ccc.de/congress/2011/Fahrplan/attachments/2022_11-ccc-qcombbdbg.pdf

[HEXAGONDSP] Qualcomm Hexagon DSP: An architecture optimized for mobile multimedia and communications
https://developer.qualcomm.com/download/hexagon/hexagon-dsp-architecture.pdf

[HEXAGONISA] Hexagon V62 Programmer's Reference Manual
https://developer.qualcomm.com/download/hexagon/hexagon-v62-programmers-reference-manual.pdf

[MODKIT] Exploring Qualcomm Baseband via ModKit (2018)
https://cansecwest.com/slides/2018/Exploring%20Qualcomm%20Baseband%20via%20ModKit%20-%20Peter%20Pi,%20XiLing%20Gong,%20and%20Gmxp,%20Tencent%20Security%20Platform%20Department.pdf

[HYPERVISORS] Hypervisor vulnerability research: state of the art
https://alisa.sh/slides/HypervisorVulnerabilityResearch2020.pdf

[PRODFUSING] Leaked Qualcomm memo on production fusing https://pastebin.com/DUEbnuTf

[SNAPDRAGON] Qualcomm® Snapdragon™ 820E Processor (APQ8096SGE)
https://developer.qualcomm.com/download/sd820e/qualcomm-snapdragon-820e-processor-apq8096sge-device-specification.pdf

Q&A

Twitter: @alisaesage

E-mail: contact@0days.engineer