Reversing & Emulating Samsung's Shannon Baseband

Grant Hernandez & Marius Muench

Tyler Tucker, Hunter Searle, Weidong Zhu
Patrick Traynor & Kevin Butler

hardwear.io
Hardware Security Conference and Training
October 1st, 2020

Intros

Grant Hernandez

- Recent Ph.D. graduate (University of Florida)
- Works on Android security and firmware analysis
- Now with Qualcomm's product security team (QPSI)

Marius Muench

- Ph.D. graduate (EURECOM)
- By now PostDoc @ VUSec
- Builds tooling for better dynamic analysis of embedded systems

Outline

- Intro to cellular and basebands
- Motivation and tooling overview
- Overview of the hardware platform
- Digging into the firmware
- ShannonOS overview
- Changing behavior with the ModKit
- Fuzzing with AFL
- Conclusions

Cellular & Baseband Overview

- Phones communicate wirelessly with cellular networks using the cellular protocols
  - Over 30 years of cellular standards across multiple generations (2G - 5G)
    - GSM (2G), WCDMA/UMTS (3G), LTE (4G), 5G-NR
  - Designed by committee (the 3GPP standards body)
- These protocols define the physical layer and up for cellular operators and devices
- **“Basebands” implement one half of the protocols**
  - Every phone today that has a SIM card has a baseband processor
- Protocol implementation is left up to individual manufacturers (below)
  - Shannon **is-a** particular implementation of these standards

Qualcomm

HISILICON

MEDIATEK

intel

infineon

Getting Insight into Basebands

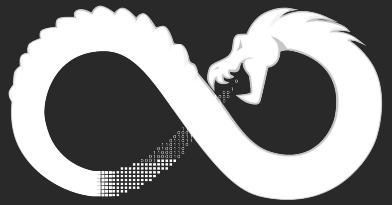
- Basebands are a large non-free component of mobile devices today
  - Can we replace them with FOSS equivalents?
  - How do they work and are they secure?

- Manual reverse engineering is an option, but basebands are huge (doesn't scale)
  - Shannon firmware is a binary blob of ~40MB in size
  - See https://github.com/grant-h/shannon_s5000/ -> **At least 4,000,000 SLoC**

- Dynamic analysis would speed up our understanding
  - ✗ On device debugging requires exploitation (debug is fused out)

- **Skip on-target testing and let's build an emulator where we have control!**
  - Lot's of custom hardware to deal with

Tooling



- Great for static reverse engineering
- Reasonable collaboration features
- No pricy licensing scheme

- Dynamic reverse engineering framework based on QEMU
- Callbacks during emulation
- Recently released python wrappers

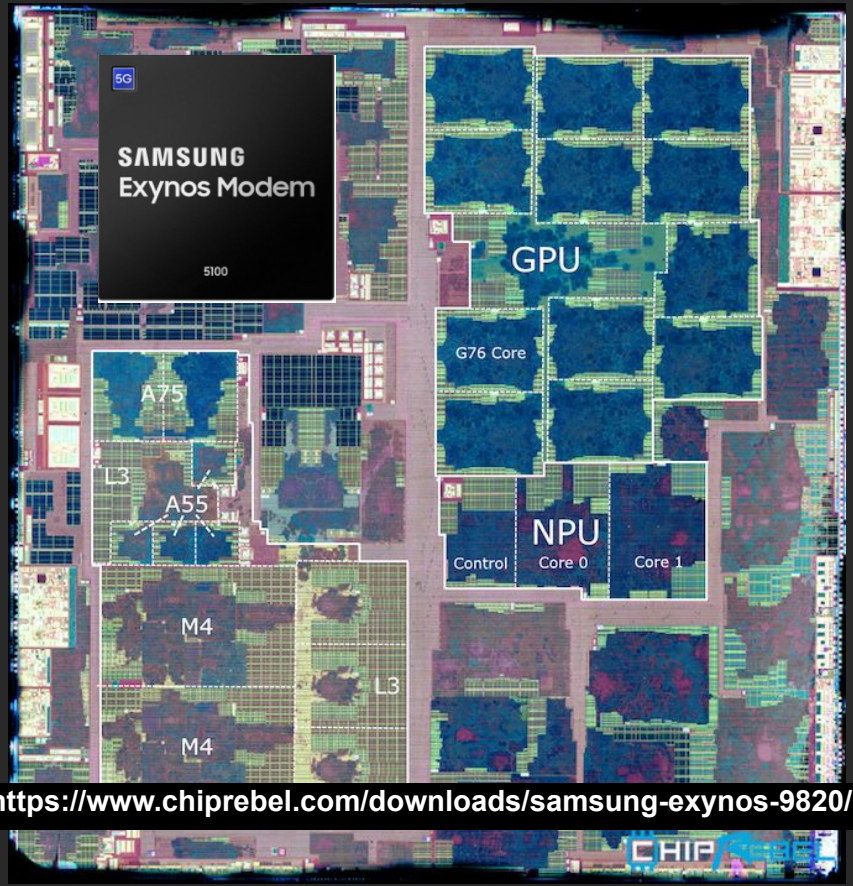
avatar²

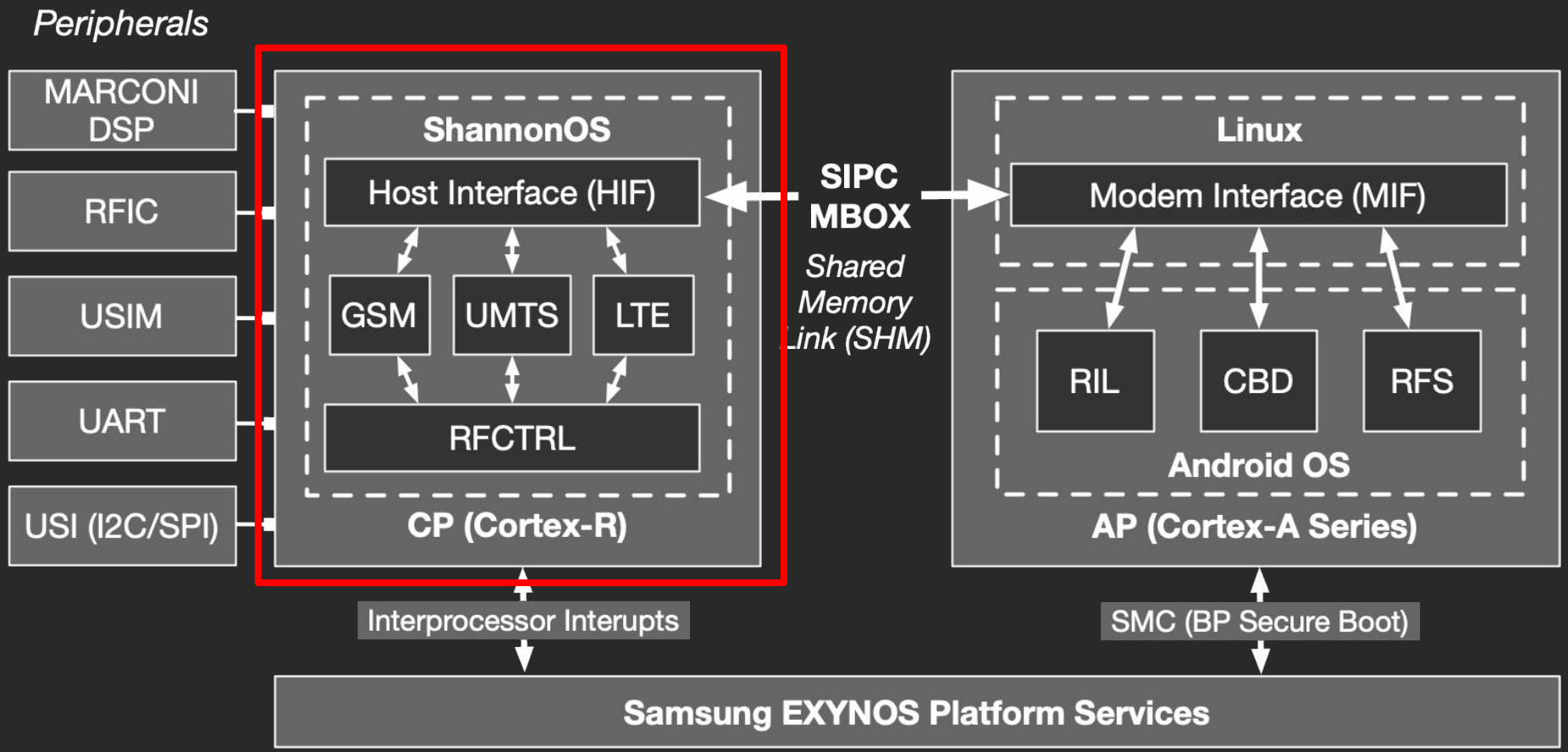
- Python framework with wrappers around various emulators (& more)
- Configurable Machine to quickly support targets
- Python based peripheral modeling

Shannon Hardware Overview

Samsung EXYNOS & Shannon

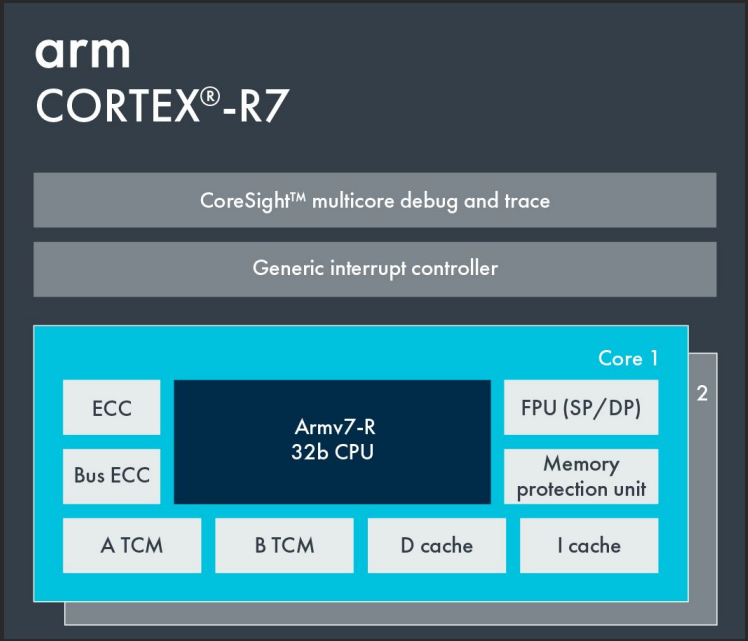
- Samsung U.S. devices prefer Qualcomm SoC's (and therefore their modem)
- Global edition devices instead use EXYNOS - Samsung LSI's in-house SoC
- Shannon co-exists in the SoC floorplan as an IP block
  - The modem is divided into digital and analog
  - The analog component (also known as RF Front-end) is on a separate chip
- We focused on Samsung S10 (SM-G973F) and S7 (SM-G935)




Peripherals
MARCONI DSP
RFIC
USIM
UART
USI (I2C/SPI)
ShannonOS
Host Interface (HIF)
GSM
UMTS
LTE
RFCTRL
CP (Cortex-R)
SIPC MBOX
Shared Memory Link (SHM)
Linux
Modem Interface (MIF)
RIL
CBD
RFS
Android OS
AP (Cortex-A Series)
Interprocessor Interupts
SMC (BP Secure Boot)
Samsung EXYNOS Platform Services

Shannon CPU Support

- ARM Cortex-R7/R8 (identified from strings and previous work)
- PANDA doesn't support these ARM variant
  - This seems like it could derail things before they even get started
- PANDA does support the R5
- **What would be required to get the R7 and R8 working?**



Taken from https://developer.arm.com/ip-products/processors/cortex-r/cortex-r7

```
diff --git a/target/arm/cpu.c b/target/arm/cpu.c
index 04b062c..9f6d05e 100644
--- a/target/arm/cpu.c
+++ b/target/arm/cpu.c
@@ -1132,6 +1132,15 @@ static void cortex_r5_initfn(Object *obj)
     define_arm_cp_regs(cpu, cortexr5_cp_reginfo);
 }

+static void cortex_r7_initfn(Object *obj)
+{
+    ARMCPU *cpu = ARM_CPU(obj);
+
+    cortex_r5_initfn(obj);
+    cpu->pmsav7_dregion = 32;
+}
+
 static const ARMCPRegInfo cortexa8_cp_reginfo[] = {
     { .name = "L2LOCKDOWN", .cp = 15, .crn = 9, .crm = 0, .opc1 = 1, .opc2 = 0,
       .access = PL1_RW, .type = ARM_CP_CONST, .resetvalue = 0 },
@@ -1573,6 +1582,7 @@ static const ARMCPUInfo arm_cpus[] = {
     { .name = "cortex-m4",    .initfn = cortex_m4_initfn,
                               .class_init = arm_v7m_class_init },
     { .name = "cortex-r5",    .initfn = cortex_r5_initfn },
+    { .name = "cortex-r7",    .initfn = cortex_r7_initfn },
     { .name = "cortex-a7",    .initfn = cortex_a7_initfn },
```

Not too many changes needed!

MPU setup and layout

```
0x00000000-0x00007fff id=00 perm=r-x
0x04000000-0x0401ffff id=01 perm=r-x
0x04800000-0x04803fff id=02 perm=rw-
0x40000000-0x47ffffff id=03 perm=rwx
0x40000000-0x40ffffff id=04 perm=r-x
0x48000000-0x4bffffff id=05 perm=rw-
0x4b800000-0x4bffffff id=06 perm=---
0x4b800000-0x4b8fffff id=07 perm=rw-
0x45600000-0x456fffff id=08 perm=rw-
0x46000000-0x467fffff id=09 perm=rw-
0x46000000-0x460fffff id=10 perm=rw-
0x46800000-0x46ffffff id=11 perm=rw-
0x47000000-0x47ffffff id=12 perm=rw-
0x47300000-0x473fffff id=13 perm=rw-
0x47400000-0x477fffff id=14 perm=r--
0x47780000-0x477fffff id=15 perm=rw-
0x80000000-0xffffffff id=16 perm=rw-
0xe0000000-0xe1ffffff id=17 perm=r--
0x50000000-0x50ffffff id=18 perm=rw-
0x51000000-0x513fffff id=19 perm=rw-
0x51400000-0x515fffff id=20 perm=rw-
0x4a800000-0x4affffff id=21 perm=rwx
0x4b000000-0x4b1fffff id=22 perm=rwx
0x00100000-0x0011ffff id=23 perm=rw-
```

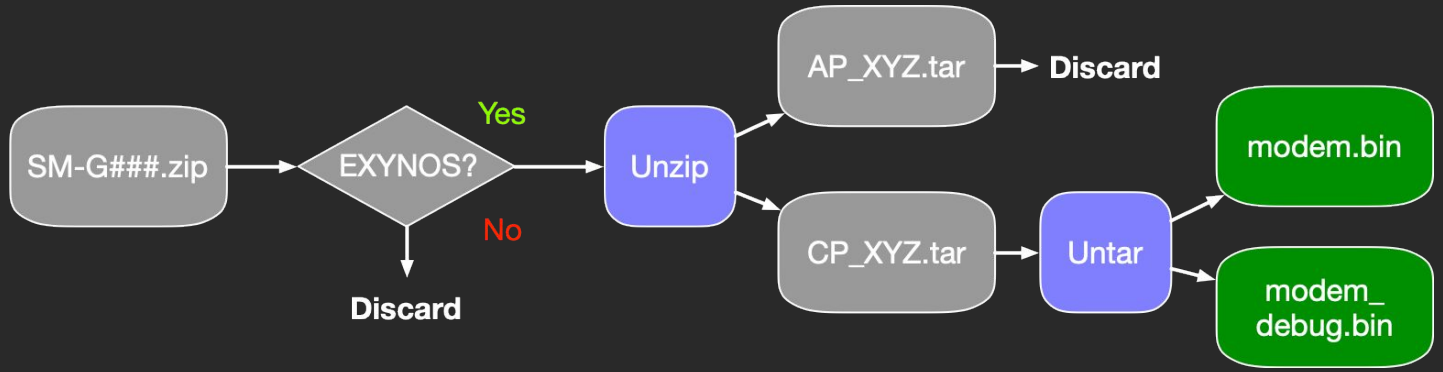
Example MPU configuration

- Well reverse engineered already:
  - Breaking Band (N. Golde, D. Komaromy - REcon'16)
- But, firmware was extracted from RAM dump
  - How do we get there without it?

The Firmware

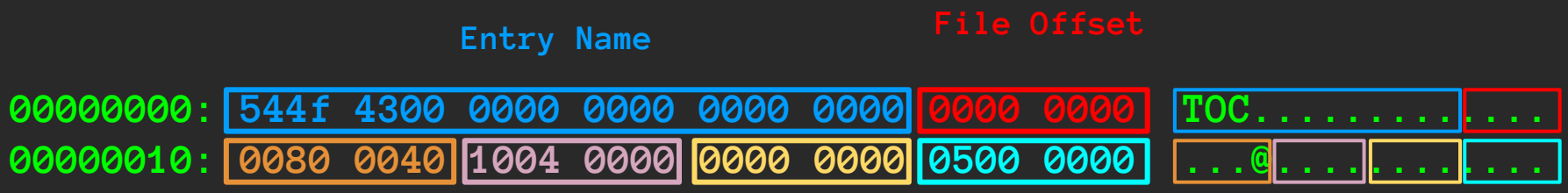
Extracting modem.bin

- Firmware available from third-party websites like SamMobile



- Previous researchers and papers mentioned that modem.bin is encrypted
  - Apparently, Samsung opted out of this recently ¯_(ツ)_/¯
- Firmware can also be extracted from the RADIO partition on a running device (if you have root)

```
00000000: 544f 4300 0000 0000 0000 0000 0000 0000  TOC.............
00000010: 0080 0040 1004 0000 0000 0000 0500 0000  ...@............


00000020: 424f 4f54 0000 0000 0000 0000 2004 0000  BOOT........ ...
00000030: 0000 0040 401e 0000 d597 ad57 0100 0000  ...@@......W....
00000040: 4d41 494e 0000 0000 0000 0000 6022 0000  MAIN........`"..
00000050: 0000 0140 a079 5402 3fb1 20ef 0200 0000  ...@.yT.?. .....
00000060: 5653 5300 0000 0000 0000 0000 009c 5402  VSS...........T.
00000070: 0000 8047 60f6 5d00 04e5 2907 0300 0000  ...G`.]...).....
00000080: 4e56 0000 0000 0000 0000 0000 0000 0000  NV..............
00000090: 0000 6045 0000 1000 0000 0000 0400 0000  ..`E............
000000a0: 4f46 4653 4554 0000 0000 0000 00aa 0700  OFFSET..........
000000b0: 0000 0000 0056 0800 0000 0000 0500 0000  .....V..........
```




```
00000020: 424f 4f54 0000 0000 0000 0000 2004 0000  BOOT........ ...
00000030: 0000 0040 401e 0000 d597 ad57 0100 0000  ...@@......W....
00000040: 4d41 494e 0000 0000 0000 0000 6022 0000  MAIN........`"..
00000050: 0000 0140 a079 5402 3fb1 20ef 0200 0000  ...@.yT.?. .....
00000060: 5653 5300 0000 0000 0000 0000 009c 5402  VSS...........T.
00000070: 0000 8047 60f6 5d00 04e5 2907 0300 0000  ...G`.]...).....
00000080: 4e56 0000 0000 0000 0000 0000 0000 0000  NV..............
00000090: 0000 6045 0000 1000 0000 0000 0400 0000  ..`E............
000000a0: 4f46 4653 4554 0000 0000 0000 00aa 0700  OFFSET..........
000000b0: 0000 0000 0056 0800 0000 0000 0500 0000  .....V..........
```


Entry Name
File Offset
00000000: 544f 4300 0000 0000 0000 0000 0000 0000 TOC.........
00000010: 0080 0040 1004 0000 0000 0000 0500 0000 ...@........
Load Address
Size
CRC
Count/Entry ID
00000020: 424f 4f54 0000 0000 0000 0000 2004 0000 BOOT.....
00000030: 0000 0040 401e 0000 d597 ad57 0100 0000 ...@@...W
00000040: 4d41 494e 0000 0000 0000 0000 6022 0000 MAIN.....`"
00000050: 0000 0140 a079 5402 3fb1 20ef 0200 0000 ...@.yT.?.
00000060: 5653 5300 0000 0000 0000 0000 009c 5402 VSS.....T.
00000070: 0000 8047 60f6 5d00 04e5 2907 0300 0000 ...G`.]..).
00000080: 4e56 0000 0000 0000 0000 0000 0000 0000 NV.....
00000090: 0000 6045 0000 1000 0000 0000 0400 0000 ..`E.....
000000a0: 4f46 4653 4554 0000 0000 0000 00aa 0700 OFFSET.....
000000b0: 0000 0000 0056 0800 0000 0000 0500 0000 .....V.....

```
00000020: 424f 4f54 0000 0000 0000 0000 2004 0000  BOOT........ ...
00000030: 0000 0040 401e 0000 d597 ad57 0100 0000  ...@@......W....
```

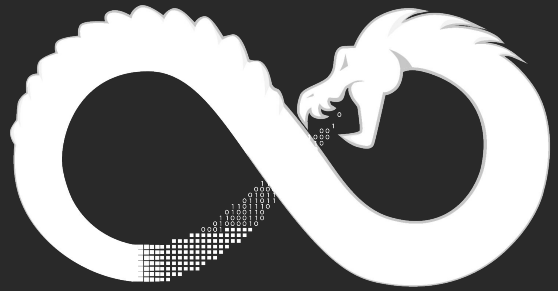
```
00000020: 424f 4f54 0000 0000 0000 0000 2004 0000  BOOT........ ...
00000030: 0000 0040 401e 0000 d597 ad57 0100 0000  ...@@......W....
00000040: 4d41 494e 0000 0000 0000 0000 6022 0000  MAIN........`"..
00000050: 0000 0140 a079 5402 3fb1 20ef 0200 0000  ...@.yT.?. .....
```

```
00000020: 424f 4f54 0000 0000 0000 0000 2004 0000  BOOT........  ...
00000030: 0000 0040 401e 0000 d597 ad57 0100 0000  ...@@...  ...W....
```

… … …

```
00000420: 3c00 00ea d8f1 9fe5 d8f1 9fe5 d8f1 9fe5  <...............
00000430: d8f1 9fe5 feff ffea d4f1 9fe5 d4f1 9fe5  ................
00000440: f81b 0000 fc1b 0000 241c 0000 0000 0000  ........$.......
00000450: 004c 4254 0000 0000 0000 0000 0000 0000  .LBT............
00000460: 0000 0000 0000 0000 0000 0000 0000 0000  ................
```

Looks like ARM “always” conditionals!
Let’s disassemble from here :)



Exception Vectors

```
40000000 3c 00 00 ea     b          boot_RESET
                      -- Flow Override: CALL_RETURN (CALL_TERMINATOR)

                      LAB_40000004                                    XREF[1]:
40000004 d8 f1 9f e5     ldr        pc=>boot_UDI,[DAT_400001e4]
40000008 d8 f1 9f e5     ldr        pc=>boot_SWI,[DAT_400001e8]
4000000c d8 f1 9f e5     ldr        pc=>boot_PREFETCH,[DAT_400001ec]
40000010 d8 f1 9f e5     ldr        pc=>boot_DATA_ABORT,[DAT_400001f0]

                      boot_NA                                         XREF[1]:
40000014 fe ff ff ea     b          boot_NA
40000018 d4 f1 9f e5     ldr        pc=>boot_IRQ,[DAT_400001f4]
4000001c d4 f1 9f e5     ldr        pc=>boot_FIQ,[DAT_400001f8]
```

BOOT

Debugging Strings

- Over 150K debug messages found in modem.bin!
- Wrote Ghidra Python script to auto-type regions starting with DBT:

```
ty = fapi.getDataTypes("TraceEntry")
caddrs = fapi.findBytes(caddr, "DBT:")

for caddr in caddrs:
   te = fapi.createData(caddr, ty)
   filename = te.getComponent(...)
   message = te.getComponent(...)
   name = "TraceEntry::%s::%s" %
     (filename, message)
   fapi.createLabel(caddr, name, ... )
```

```
                    TraceEntry::tphy_CMeasurementGapManager::GetRe... XREF
41ca521c 44 42 54        TraceEntry
         3a 22 01
         00 00 04 ...
   41ca521c 44 42 54 3a     uint      3A544244h                 magic
   41ca5220 22 01 00 00     uint      122h                      unk1
   41ca5224 04 00 00 00     uint      4h                        unk2
   41ca5228 98 ba cd fe     uint      FECDBA98h                 unk_magic
   41ca522c 97 99 ca 41     char *    s_GetReconfirmAbortInt... message
   41ca5230 23 23 00 00     uint      2323h                     linenum
   41ca5234 e4 9b 4b 40     char *    s_../../../HEDGE/TL1/T... file
```

```
44 42 54 3a 1e 01 00 00 02 00 00 00 98 ba cd fe   DBT:............
08 96 ca 41 da 1d 00 00 e4 9b 4b 40 44 42 54 3a   ...A......K@DBT:
22 01 00 00 04 00 00 00 98 ba cd fe 3f 96 ca 41   "...........?..A
b1 1e 00 00 e4 9b 4b 40 44 42 54 3a 22 01 00 00   ......K@DBT:"...
04 00 00 00 98 ba cd fe 86 96 ca 41 b3 1e 00 00   ...........A....
e4 9b 4b 40 44 42 54 3a 19 01 00 00 01 00 00 00   ..K@DBT:........
```

First emulation steps

- We have everything needed for emulation!
  - CPU architecture
  - Memory map
  - Entry point
  - Bonus: debugging strings

```
from avatar2 import *
class ShannonMachine():
    self.avatar = Avatar(arch=ARM_CORTEX_R7)
    self.emu    = avatar.add_target(PyPandaTarget, entry_address=0x40000000)

    for entry in self.modem_file.entries:
        self.add_memory_range(entry.load_address, entry.size,
            file=section_path, name=entry.name)
    self.avatar.init_targets()
    self.avatar.cont()
```

Displaying the Boot UART

```
if (boot_mode == DUMP_MODE) {
  boot_unk_common_setup();
  uart_putc('#');
  boot_crash_or_dump();
  boot_unk_crash();
  uart_puts(s_Done_40000550);
  FUN_40000d84(&DAT_00002e00);
}
else {
  if (boot_mode == BOOT_MODE) {
    boot_unk_common_setup();
    uart_putc('#');
```

```
self.create_peripheral(UARTPeripheral,
0x84000000, 0x1000, name='boot_uart')
```

```
      boot_stage2(nextFnPointer);
      goto LAB_400004f0;
    }
    r0 = s_XxX!_40000570;
  }
  else {
    r0 = s_Unknown_4000055c;
  }
  uart_puts(r0);
}
```

```
class UARTPeripheral(ShannonPeripheral):
    def hw_read(self, offset, size):
        if offset == 0x18:
            return self.status

        return 0

    def hw_write(self, offset, size, value):
        if offset == 0:
            sys.stderr.write(chr(value & 0xff))
            sys.stderr.flush()
        else:
            self.log_write(value, size, "UART")

        return True

    def __init__(self, name, address, size, **kwargs):
        super(UARTPeripheral, self).__init__(name,
           address, size)
        self.status = 0
        self.write_handler[0:size] = self.hw_write
        self.read_handler[0:size] = self.hw_read
```

Baseband Initialization: pal_init1()

- A huge monolith function that starts all modem subsystems and tasks
  - Activates malloc heap
  - Loads NV items
  - Starts timers
  - Initializes DSP(s) and other peripherals
  - Starts all tasks

- Crucial for setting up the baseband state

Bas

- A
  - O
  - O
  - O
  - O
  - O
- Cr

```
1
2  void pal_init1(void)
3
4  {
5    int iVar1;
6    undefined4 uVar2;
7    LogCtx local_4110;
8    byte local_4108 [4];
9    undefined local_4104 [4];
10   undefined auStack16640 [16460];
11   undefined auStack180 [164];
12
13   local_4104[0] = 0;
14   local_4108[0] = 0;
15   memclear(auStack16640,&DAT_0000404c);
16   memclear(auStack180,0x4c);
17   FUN_4162c4e0(0xac283f);
18   iVar1 = FUN_406d0408();
19   if (iVar1 != 0) {
20     iVar1 = 2;
21   }
22   FUN_406cf8f8(iVar1);
23   FUN_4072f5e2();
24   FUN_40fc8ea2();
25   FUN_406cfbfa();
26   FUN_406cfd34();
27   pal_MemDriverPmd();
28   pal_MemDriverRtk();
29   FUN_406d0ff4();
30   FUN_40fc09e4();
31   DAT_43549238._0_2_ = thunk_FUN_40735b1a();
32   FUN_40fc1e74();
33   pal_RegInit();
34   FUN_406cfae4();
35   gsm_unknown_complicated(&PTR_s_CP_BOOT_419bf1d0,&DAT_402e36e8,0,"RegInit completed");
36   UTIL_Math();
37   uVar2 = FUN_40fc8798("S355AP_DREAM");
38   FUN_408b6d26("S355AP_DREAM",uVar2);
39   DAT_4169abbc = 1;
40   FUN_40fc2e32(&DAT_000017be,auStack16640);
41   DAT_4169abb4 = 1;
42   iVar1 = thunk_FUN_040053b0(auStack16640,auStack180,0x4c);
43   if (iVar1 == 0) {
44     pal_Security_memory();
45     local_4110.te = &TraceEntry::pal_main::Protected_pal_RegInit_Secure_Block_Def;
46     local_4110.flags = &DAT_00001582;
47     log_printf(&local_4110,&DAT_fecdba98);
48   }
49   else {
50     local_4110.te = &TraceEntry::pal_main::Protected_pal_RegInit_Secure_Block_Ini;
51     local_4110.flags = &DAT_00001582;
52     log_printf(&local_4110,&DAT_fecdba98);
53   }
54   FUN_412e9810();
55   pal_Csc__defined();
56   TestConfigurationService__TCS_Init_NVtype_Features();
57   FUN_40fbb05e();
58   Mti_Common_Define__Initialization();
59   FUN_40fc2e32(0x30,local_4104);
60   FUN_40fc2e32(0x4e,local_4108);
61   FUN_40fc8e74((uint)local_4108[0]);
62   FUN_406ce62e();
63   uart_main(2,&DAT_4310a644,0x400);
64   dbg_Core__Hello();
65   memReleaseNonSecureMemoryRx();
66   FUN_409cc58a();
67   DmTraceMsg__Default();
68   FUN_40735a6a();
69   modem_uart_printf("UpTimer Init: ");
70   FUN_408142ac();
71   modem_uart_printf("Complete.\r\n");
72   thunk_FUN_04001bfc();
73   thunk_FUN_04004584();
74   FUN_406cf82e();
75   do {
76     iVar1 = thunk_FUN_04015f06();
77   } while (iVar1 == 0);
78   hal_modem__InitModem();
79   ps_Controller__llps_2g_peri_client_id();
80   FUN_40fc22be();
81   pal_WatchdogKick();
82   FUN_406ef7fa();
83   FUN_406cf750();
84   thunk_FUN_04001d66();
85   tcs_rfCfg();
86   tcs_rfCfg();
87   local_4110.te = &TraceEntry::pal_main::BandSet_Done;
88   local_4110.flags = &DAT_00001582;
89   log_printf(&local_4110,&DAT_fecdba98);
90   FUN_40711d14();
91   FUN_406ce8bc();
92   hw_Device();
93   hw_Acpm__pal_SmCreateEventGroup();
94   FUN_408aede0();
95   FUN_40718528();
96   hw_ClkFindSysClkCofigInfoIndex(2,1);
97   FUN_40fb6380();
98   DSP_DM_CreateTimers();
99   FUN_406d0b82();
100  FUN_4077748a();
101  FUN_406d0eda();
102  DAT_416b525c = thunk_FUN_04015538(0x4e);
103  thunk_FUN_04003f9a();
104  thunk_FUN_0400400e(DAT_416b525c,&LAB_406cee78+1);
105  thunk_FUN_04003fa4(DAT_416b525c);
106  hw_Power__Information();
107  FUN_406cf9b8();
108  FUN_40fc2e32(0x5a,&DAT_416a0564);
109  gsm_unknown_complicated(&PTR_s_CP_BOOT_419bf1d0,&DAT_402e36f8,0,"pal_init1 completed");
110  FUN_40fc1200(&LAB_406cee34+1);
111  return;
```

pal_init1() – peripherals

- Iteratively emulated peripherals by watching for crash strings or infinite loops
- MMIO monitoring used to see which peripherals needed more modeling
- Simple automated cyclic-bit pattern heuristic
- **Partially emulated: PMIC, CLK, DSP, SOC, SIPC/SHM, TIMER, GIC**

```
class ShannonPeripheral():
    def log_read(self, value, size, offset_name):
        log.log(self.log_level,
            "%s: %0" + str(size*2) + "x <- %s[%s]",
            self.format_address(self.pc), value,
            self.name, offset_name)

    def log_write(self, value, size, offset_name):
        log.log(self.log_level, "%s: %s[%s] <- %0" +
            str(size*2) + "x",
            self.format_address(self.pc), self.name,
            offset_name, value)

    def cyclic_bit(self, pattern=1, cycle_len=32):
        val = (pattern << self.cycle_idx ) % 0xffffffff
        self.cycle_idx = (self.cycle_idx + 1) % cycle_len
        return val
```

We reached the banner!

```
[VARIANT/PALVar/Platform_EV/CHIPSET/S5000AP/device/User/src/pal_main.c] - BandSet: Done.
================================================
            DEVELOPMENT PLATFORM
 - ARM Emulation Baseboard | Cortex-R7

 - Software Build Date : Jul 19 2019 05:03:07
 - Software Builder    :
 - Compiler Version    : ARM RVCT 50.6 [Build 422]

    Platform Abstraction Layer (PAL) Powered by
               CP Platform Part
================================================
[INFO] OS_Create_Event_Group(0x4177b710, ACPM_PMIC_RX_EVENT)
[VARIANT/PALVar/Platform_EV/HAL/Common/driver/Acpm/src/hw_Acpm.c] -
[ACPM] Shannon OS [_ShannonOS_3.2_R8_AC5]
```


Shannon0S

Tasks

- All metadata about tasks are stored in TaskStructs
- A global array contains the TaskStructs for each task
  - Around 100 tasks present in the baseband!
- Each task has its own stack
  - The stackbase is guarded by the magic constant '\xef\xbe\xad\xde'
- Communicate with each other via Queues
  - Over 290 different ones!

```
0x00: Start of structure
 [...]
 0x0c: TaskID
 0x10: Stackbase
 [...]
 0x24: Name Pointer
 [...]
 0x2c: Stacksize
 0x30: Main function
 0x34: Pre-main function
 [...]
0x108: End of structure
```

Acpm	CC	LTEL2LRx	REG_SAP	IMS_CC
Default	MM	LTEL2LTx	AS_SAP	LPP
DM	SM	LTEL2TCM	SMS_SAP	SHM
DM_TX	SS	LTEL2IDLE	CC_SS_SAP	UL2CC
BDA	L1C	LTEL2HTx	SIM_SAP	UL2DL
CIQD	PPP	LTEL2HRx	DBG_SAP	UL2UL
CIQD_FE	GDA	LTE_TLP	DS_REG_SAP	UDATA
Background	CDH	LTE_MTM	DS_AS_SAP	UBMCTask
TpTest	VSUP	LTE_DM	DS_SMS_SAP	ephyFramework
TaskReg	VCG	EDFS	DS_CC_SS_SAP	syncTask
DBGUNS	VCE	URRC	DS_SIM_SAP	recMailTask
DBGCMD	SAEL3	HSPA_CALIBRATION	DS_DBG_SAP	sendMailTask
DBGCMD2	DS_SAEL3	LLC	MMC	BTL
InitPacketHandler	PDNMGR	GRR	MMC_IF	CLM
PacketHandler	SIM	RLC	SR_IF	CLTCP
PBM	DS_SIM	GMAC	LTE_MMC_GL1	SecuCh
DS_PBM	LteRrm	GLAPD	USAT	SHUB_MSG
ATI	LTE_L1LC	SNDCP	DS_USAT	SSH
MTI	LteRrc	SRM	LTE_TCPIP	CPCOP
SMS	LteRrc_DS	LCSM	LTE_SISO_ASYNC	PROXIMITY

Scheduling & Interrupts

- Tasks block (yield) on receiving IPC messages
- Can be preempted by higher priority tasks or interrupts (if not disabled)
- Interrupt Flow
  - IRQ Received
  - Find and call IRQ handler OR set task to wakeup
  - Call OS_irq_schedule_task
  - End of interrupt
- Over 260 IRQ handlers!
  - A sobering look at how hard it would be to fully emulate
  - https://github.com/grant-h/ShannonBaseband/blob/master/firmware/extdata/G973FXXU3ASG8/interrupts.h

Timers

- Monumental component for scheduler
  - Emulation hangs without them
  - Expected to raise IRQ on expiry
- Do NOT follow the specification of R7/R8 timers
- Prototyping in Python, then implementation in PANDA

```
static void shannon_timer_write(void *opaque, hwaddr offset,
                                uint64_t value, unsigned size) {
  [...]
  switch (offset) {
    case 0x00: /* TimerLoad */
      s->limit = value;
        ptimer_set_limit(s->timer, s->limit, 1);
        break;
    case 0x04: /* TimerControl */
      if (s->control & STIMER_CTRL_ENABLE)
        ptimer_stop(s->timer);
      s->control = value;
      freq = s->freq;
      ptimer_set_limit(s->timer, s->limit, 1);
      ptimer_set_freq(s->timer, freq);
      if (s->control & STIMER_CTRL_ENABLE)
        ptimer_run(s->timer, (s->control & STIMER_CTRL_PERIODIC) == 0);
      break;
    case 0x0c: //STOP?
      ptimer_stop(s->timer);
      break;
    case 0x10:
      s->int_level = 0;
      shannon_timer_update(s);
      break;
    case 0x14: /* TIM_IRQ_LEVEL */
      s->int_level = value;
      break;
    [...]
```

The ModKit

ModKit

- Inject custom tasks inside the emulated baseband
  - Dynamically loaded during emulation
  - Allows for interaction with the baseband

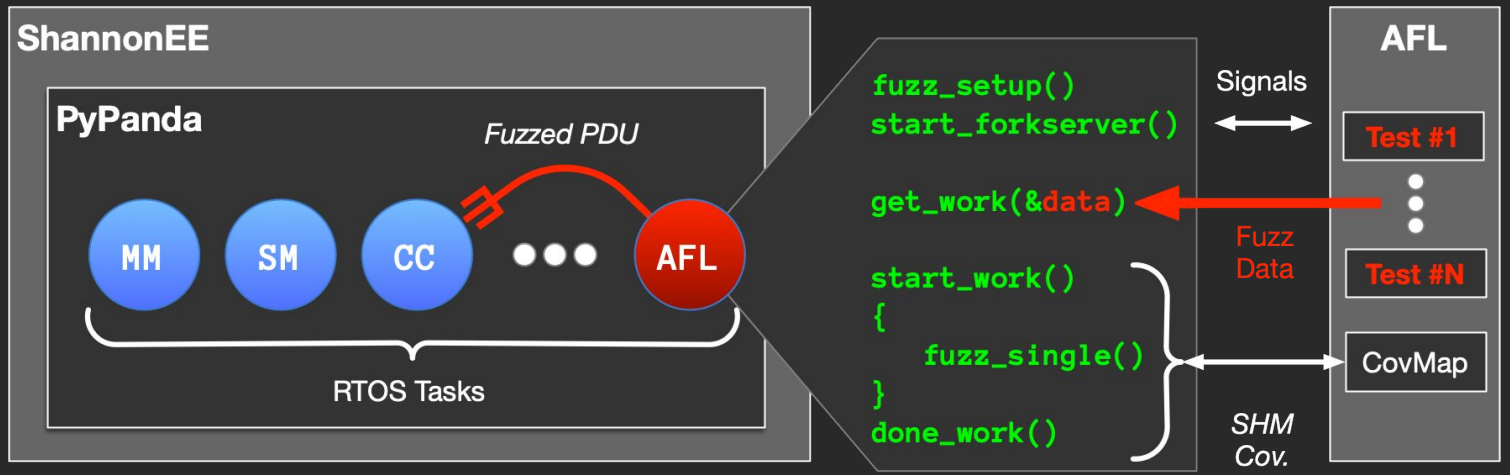
- C-based compiled modules
  - Dynamic symbol resolution via patterns
  - Creative use of preprocessor macros:

```
DYNAMIC_SYMBOLS = {
      [...]
      "OS_fatal_error" : {
      "pattern" : "70 b5 05 46 ???????? ?? 48
?? 24",
      },
      "pal_MemAlloc" : {
      "pattern" : "2d e9 f0 4f  0d 00  83 b0
99 46  92 46  80 46",
     [...]
}
```

```
#define MODKIT_FUNCTION_SYMBOL(ret, name, ...) \
  ret (*__SYMREQ_FUNC_ ## name)(__VA_ARGS__) = (ret (*)(__VA_ARGS__))0xaaaaaaaa; \
  static ret (*name)(__VA_ARGS__) __attribute__ ((weakref, alias ("__SYMREQ_FUNC_" # name)));
```

Using the ModKit for the AFL Task

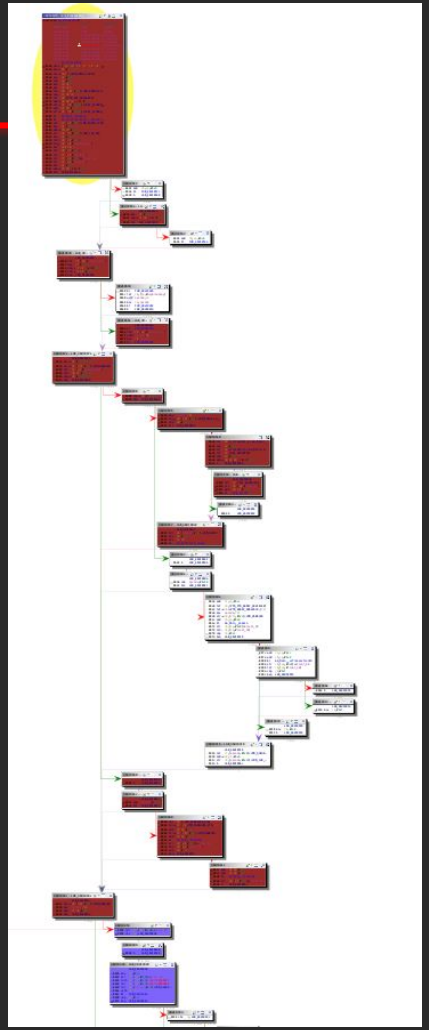
- Dynamic baseband symbol resolution for a nice FFI
  - Provides access to recovered Shannon symbols
- Mods dynamically linked into RWX page in baseband memory



Fuzzing with AFL

- Campaigns
  - GSM SM (GPRS)
  - GSM MM
  - GSM CC
- Harnesses < 50 lines of C thanks to our ModKit
- Able to export coverage from emulator to debug fuzzer hangs in the DragonDance plugin

```
afl-fuzz -i inputs -o campaign -- ./shannon_emu.py
--restore-snapshot g973 --fuzz gsm_mm --fuzz-input @@

# Get coverage and debug output of `input’
./shannon_emu.py --restore-snapshot g973 --fuzz-triage
gsm_mm --fuzz-input input
```



Fruits of our labor

- Could replicate several n-days
  - E.g., the bug discussed in “A Walk with Shannon” (Amat Cama - Infiltrate)

- Found 0day: SVE-2020-18098
  - using a fuzzing harness on the GSM CC task

SVE-2020-18098: Buffer overflow vulnerability in Baseband with abnormal SETUP message

Severity: Critical
Affected versions: O(8.x), P(9.0), Q(10.0) devices with Exynos chipsets
Reported on: June 19, 2020
Disclosure status: Privately disclosed.
A possible buffer overflow vulnerability in baseband allows arbitrary code execution.
The patch adds the proper validation of the buffer length.

Let’s wrap it up!

Summary

- Baseband firmware is huge and complicated
  - But reversing is still possible!
- Emulation is tedious to set up
  - But rewards with detailed insights
  - Allows for additional security testing!
- Choose your tooling wisely
  - Ghidra: collaborative RE
  - PANDA: full-system emulation capabilities
  - Avatar²: peripheral modeling and orchestration

Releases

https://github.com/grant-h/ShannonBaseband

- GHIDRA Tools
  - Shannon firmware loader
  - Debug annotation script
  - Auto-renamer
- Shannon reversing details
  - Export of 2017 Shannon image (reversed)
  - On-device log parser (.BTL) and file format info

https://github.com/grant-h/shannon_s5000/

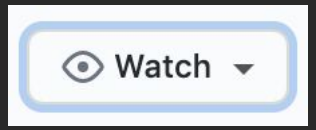
- Code skeleton

https://github.com/grant-h/ShannonFirmware

- Firmware repository

https://github.com/grant-h/ShannonEE

- **ShannonEE** (will be released at a later time)



Acknowledgements

Semiconductor Research Corporation

NSF/SRC CNS-1815883

NWO 628.001.030 ("TROPICS")

AFOSR FA8560-18-S-1201

AFRL/AFOSR-2018-003

ONR OTA-N00014-20-1-2205

Thanks!

@Digital_Cold

@nSinusR

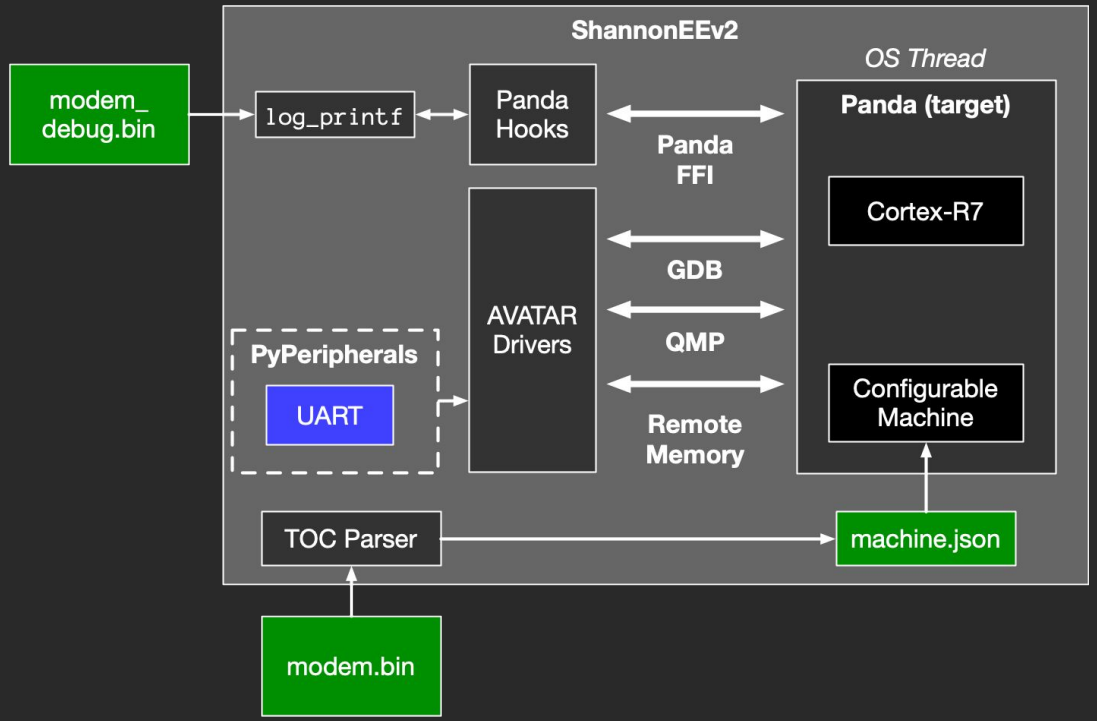
Backup Slides


Demo

```
[21.57221][EDFS] 0x40fc2d79 0b10: [pal_Reg.c] - Reg_ID: 5710 , pal_RegItemWrite file ../../../PALCommon/C-Common/RF/RF_Drv/HSPA/RF_HSPA_NV.c line 599
[21.57245][EDFS] 0x40fc2d79 0b10: [pal_Reg.c] - Reg_ID: 5711 , pal_RegItemWrite file ../../../PALCommon/C-Common/RF/RF_Drv/HSPA/RF_HSPA_NV.c line 602
[21.57263][EDFS] 0x40fc2d79 0b10: [pal_Reg.c] - Reg_ID: 5712 , pal_RegItemWrite file ../../../PALCommon/C-Common/RF/RF_Drv/HSPA/RF_HSPA_NV.c line 604
[21.57279][EDFS] 0x40fc2d79 0b10: [pal_Reg.c] - Reg_ID: 5713 , pal_RegItemWrite file ../../../PALCommon/C-Common/RF/RF_Drv/HSPA/RF_HSPA_NV.c line 606
[21.57299][EDFS] 0x40fc2d79 0b10: [pal_Reg.c] - Reg_ID: 5714 , pal_RegItemWrite file ../../../PALCommon/C-Common/RF/RF_Drv/HSPA/RF_HSPA_NV.c line 608
[21.57316][EDFS] 0x40fc2d79 0b10: [pal_Reg.c] - Reg_ID: 5715 , pal_RegItemWrite file ../../../PALCommon/C-Common/RF/RF_Drv/HSPA/RF_HSPA_NV.c line 611
[21.57332][EDFS] 0x40fc2d79 0b10: [pal_Reg.c] - Reg_ID: 5716 , pal_RegItemWrite file ../../../PALCommon/C-Common/RF/RF_Drv/HSPA/RF_HSPA_NV.c line 613
[21.57348][EDFS] 0x40fc2d79 0b10: [pal_Reg.c] - Reg_ID: 5717 , pal_RegItemWrite file ../../../PALCommon/C-Common/RF/RF_Drv/HSPA/RF_HSPA_NV.c line 615
[21.57364][EDFS] 0x40fc2d79 0b10: [pal_Reg.c] - Reg_ID: 5718 , pal_RegItemWrite file ../../../PALCommon/C-Common/RF/RF_Drv/HSPA/RF_HSPA_NV.c line 617
[21.57380][EDFS] 0x40fc2d79 0b10: [pal_Reg.c] - Reg_ID: 5699 , pal_RegItemWrite file ../../../PALCommon/C-Common/RF/RF_Drv/HSPA/RF_HSPA_NV.c line 575
[21.57397][EDFS] 0x40fc2d79 0b10: [pal_Reg.c] - Reg_ID: 5700 , pal_RegItemWrite file ../../../PALCommon/C-Common/RF/RF_Drv/HSPA/RF_HSPA_NV.c line 577
[21.57413][EDFS] 0x40fc2d79 0b10: [pal_Reg.c] - Reg_ID: 5701 , pal_RegItemWrite file ../../../PALCommon/C-Common/RF/RF_Drv/HSPA/RF_HSPA_NV.c line 579
[21.57429][EDFS] 0x40fc2d79 0b10: [pal_Reg.c] - Reg_ID: 5702 , pal_RegItemWrite file ../../../PALCommon/C-Common/RF/RF_Drv/HSPA/RF_HSPA_NV.c line 581
[21.57445][EDFS] 0x40fc2d79 0b10: [pal_Reg.c] - Reg_ID: 5703 , pal_RegItemWrite file ../../../PALCommon/C-Common/RF/RF_Drv/HSPA/RF_HSPA_NV.c line 584
[21.57461][EDFS] 0x40fc2d79 0b10: [pal_Reg.c] - Reg_ID: 5704 , pal_RegItemWrite file ../../../PALCommon/C-Common/RF/RF_Drv/HSPA/RF_HSPA_NV.c line 586
[21.57477][EDFS] 0x40fc2d79 0b10: [pal_Reg.c] - Reg_ID: 5705 , pal_RegItemWrite file ../../../PALCommon/C-Common/RF/RF_Drv/HSPA/RF_HSPA_NV.c line 588
[21.57492][EDFS] 0x40fc2d79 0b10: [pal_Reg.c] - Reg_ID: 5706 , pal_RegItemWrite file ../../../PALCommon/C-Common/RF/RF_Drv/HSPA/RF_HSPA_NV.c line 590
[21.57509][EDFS] 0x40fc2d79 0b10: [pal_Reg.c] - Reg_ID: 5707 , pal_RegItemWrite file ../../../PALCommon/C-Common/RF/RF_Drv/HSPA/RF_HSPA_NV.c line 593
[21.57525][EDFS] 0x40fc2d79 0b10: [pal_Reg.c] - Reg_ID: 5708 , pal_RegItemWrite file ../../../PALCommon/C-Common/RF/RF_Drv/HSPA/RF_HSPA_NV.c line 595
[21.57542][EDFS] 0x40fc2d79 0b10: [pal_Reg.c] - Reg_ID: 5709 , pal_RegItemWrite file ../../../PALCommon/C-Common/RF/RF_Drv/HSPA/RF_HSPA_NV.c line 597
[21.57562][EDFS] 0x40fc2d79 0b10: [pal_Reg.c] - Reg_ID: 5710 , pal_RegItemWrite file ../../../PALCommon/C-Common/RF/RF_Drv/HSPA/RF_HSPA_NV.c line 599
[21.57578][EDFS] 0x40fc2d79 0b10: [pal_Reg.c] - Reg_ID: 5711 , pal_RegItemWrite file ../../../PALCommon/C-Common/RF/RF_Drv/HSPA/RF_HSPA_NV.c line 602
[21.57594][EDFS] 0x40fc2d79 0b10: [pal_Reg.c] - Reg_ID: 5712 , pal_RegItemWrite file ../../../PALCommon/C-Common/RF/RF_Drv/HSPA/RF_HSPA_NV.c line 604
[21.57610][EDFS] 0x40fc2d79 0b10: [pal_Reg.c] - Reg_ID: 5713 , pal_RegItemWrite file ../../../PALCommon/C-Common/RF/RF_Drv/HSPA/RF_HSPA_NV.c line 606
[21.57626][EDFS] 0x40fc2d79 0b10: [pal_Reg.c] - Reg_ID: 5714 , pal_RegItemWrite file ../../../PALCommon/C-Common/RF/RF_Drv/HSPA/RF_HSPA_NV.c line 608
[21.57642][EDFS] 0x40fc2d79 0b10: [pal_Reg.c] - Reg_ID: 5715 , pal_RegItemWrite file ../../../PALCommon/C-Common/RF/RF_Drv/HSPA/RF_HSPA_NV.c line 611
[21.57658][EDFS] 0x40fc2d79 0b10: [pal_Reg.c] - Reg_ID: 5716 , pal_RegItemWrite file ../../../PALCommon/C-Common/RF/RF_Drv/HSPA/RF_HSPA_NV.c line 613
[21.57674][EDFS] 0x40fc2d79 0b10: [pal_Reg.c] - Reg_ID: 5717 , pal_RegItemWrite file ../../../PALCommon/C-Common/RF/RF_Drv/HSPA/RF_HSPA_NV.c line 615
[21.57690][EDFS] 0x40fc2d79 0b10: [pal_Reg.c] - Reg_ID: 5718 , pal_RegItemWrite file ../../../PALCommon/C-Common/RF/RF_Drv/HSPA/RF_HSPA_NV.c line 617
[21.57706][EDFS] 0x40fc2d79 0b10: [pal_Reg.c] - Reg_ID: 5699 , pal_RegItemWrite file ../../../PALCommon/C-Common/RF/RF_Drv/HSPA/RF_HSPA_NV.c line 575
[21.57722][EDFS] 0x40fc2d79 0b10: [pal_Reg.c] - Reg_ID: 5700 , pal_RegItemWrite file ../../../PALCommon/C-Common/RF/RF_Drv/HSPA/RF_HSPA_NV.c line 577
[21.57738][EDFS] 0x40fc2d79 0b10: [pal_Reg.c] - Reg_ID: 5701 , pal_RegItemWrite file ../../../PALCommon/C-Common/RF/RF_Drv/HSPA/RF_HSPA_NV.c line 579
[21.57753][EDFS] 0x40fc2d79 0b10: [pal_Reg.c] - Reg_ID: 5702 , pal_RegItemWrite file ../../../PALCommon/C-Common/RF/RF_Drv/HSPA/RF_HSPA_NV.c line 581
[21.57769][EDFS] 0x40fc2d79 0b10: [pal_Reg.c] - Reg_ID: 5703 , pal_RegItemWrite file ../../../PALCommon/C-Common/RF/RF_Drv/HSPA/RF_HSPA_NV.c line 584
[21.57785][EDFS] 0x40fc2d79 0b10: [pal_Reg.c] - Reg_ID: 5704 , pal_RegItemWrite file ../../../PALCommon/C-Common/RF/RF_Drv/HSPA/RF_HSPA_NV.c line 586
[21.57801][EDFS] 0x40fc2d79 0b10: [pal_Reg.c] - Reg_ID: 5705 , pal_RegItemWrite file ../../../PALCommon/C-Common/RF/RF_Drv/HSPA/RF_HSPA_NV.c line 588
```

https://drive.google.com/file/d/1PD5o2qhfHE0n_QOSEAvq26uu5MuBNUrc/view?usp=sharing

Emulation infrastructure



On-Device Replication

Over-the-air Confirmation

- **SDR**: bladeRF x40 ($420)
- **Base station**: YateBTS (modified with exploit payload)
- **Target device**: Samsung S10 / S7
- **SIM Card**: Osmocom USIM (for easy phone connection)
- Thanks to Tyler Tucker for recording the demo and managing the test setup!

https://youtu.be/Bhs054ma5OQ

Boot Mode & IPC

Shannon Boot Mode

- Bootloader debug prints using UART
  - Saved to early boot log, can be dumped from kernel as well
- DUMP mode activated during crash dump
- BOOT mode for normal startup
- **Who signals these boot modes?**

```
if (boot_mode == DUMP_MODE) {
  boot_unk_common_setup();
  uart_putc('#');
  boot_crash_or_dump();
  boot_unk_crash();
  uart_puts(s_Done_40000550);
  FUN_40000d84(&DAT_00002e00);
}
else {
  if (boot_mode == BOOT_MODE) {
    boot_unk_common_setup();
    uart_putc('#');
    boot_prepare_mpu_next_ram();
    uart_puts(s_Boot_40000568);
    nextFnPointer = boot_comm_ap();
    if ((void *)0x10000000 < nextFnPointer) {
      boot_stage2(nextFnPointer);
      goto LAB_400004f0;
    }
    r0 = s_XxX!_40000570;
  }
  else {
    r0 = s_Unknown_4000055c;
  }
  uart_puts(r0);
}
```

A side-quest to the Samsung Kernel

- Examined the Samsung S10's kernel sources to better understand the modem's boot
  - Samsung S10 Kernel - https://github.com/grant-h/SM-G973F-Kernel
- `drivers/misc/modem_v1`
  - modem_io_device.c - dev node ioctl handler (for booting), read/write for commands
  - link_device_memory.h - **Contains structs on shared memory regions**
- Cellular Boot Daemon (CBD) communicates with `/dev/umts_boot0` (modem_v1)
  - IOCTL_MODEM_RESET - Reset and await new boot
  - IOCTL_SECURITY_REQUEST - Establish modem bootloader secure boot
  - IOCTL_MODEM_ON, IOCTL_MODEM_BOOT_ON - Start boot process
  - IOCTL_MODEM_DL_START - Download code to modem memory
  - IOCTL_MODEM_BOOT_OFF - Finalizes boot process

Cross-core Comms

- Modem driver in Linux maps shared memory region between CP and AP for DMA-based IPC (ring buffers)
- Commands are queued in a full-duplex fashion
- Commands come from Samsung RIL

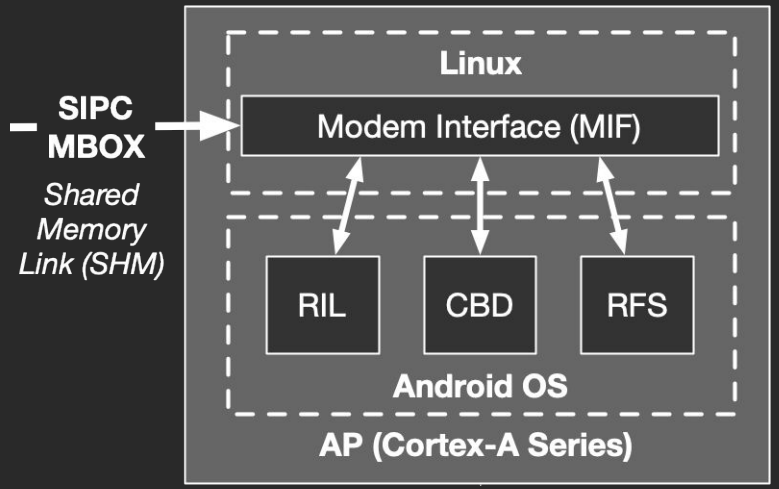
```
#define MEM_IPC_MAGIC      0xAA
#define MEM_CRASH_MAGIC    0xDEADDEAD
#define MEM_BOOT_MAGIC     0x424F4F54
#define MEM_DUMP_MAGIC     0x44554D50
```

link_device_memory.h

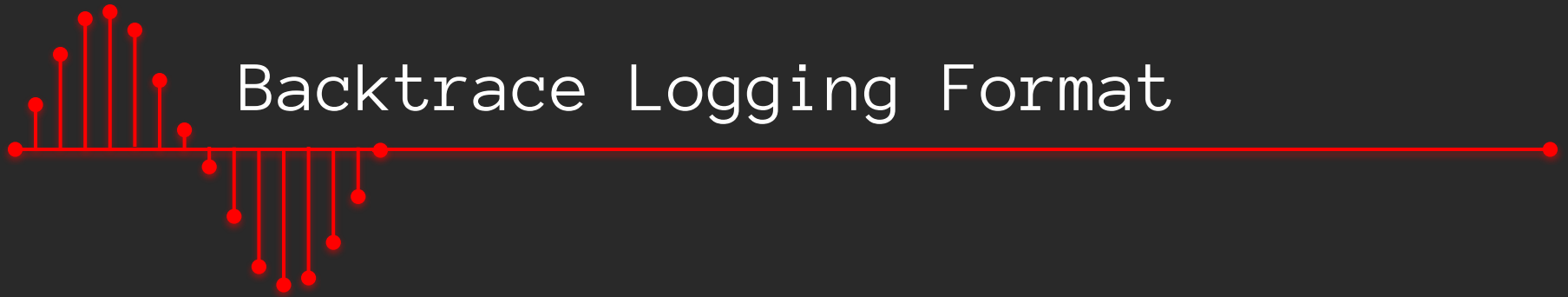
```
struct __packed shmem_4mb_phys_map {
    u32 magic;
    u32 access;

    u32 fmt_tx_head, fmt_tx_tail;
    u32 fmt_rx_head, fmt_rx_tail;

    u32 raw_tx_head, raw_tx_tail;
    u32 raw_rx_head, raw_rx_tail;
     ...
    char fmt_tx_buff[SHM_4M_FMT_TX_BUFF_SZ];
    char fmt_rx_buff[SHM_4M_FMT_RX_BUFF_SZ];
     ...
    char raw_tx_buff[SHM_4M_RAW_TX_BUFF_SZ];
    char raw_rx_buff[SHM_4M_RAW_RX_BUFF_SZ];
};
```

Samsung IPC (SIPC) Clients

- SIPC flows bidirectionally over this SHM link
- Cellular Boot Daemon (CBD)
  - Maps modem from local Android partition into baseband memory
  - Sends configuration / debugging commands
- Remote File System (RFS)
  - Provides open/read/write to Android filesystem (a.k.a. EFS) from modem
  - Stores modem config (PLMN, bandset, etc.)
- Radio Interface Layer (RIL)
  - Provides a HAL for Android telephony services
  - Significant configuration API for modem
- FOSS version of RIL - https://redmine.replicant.us/projects/replicant/wiki/Libsamsung-ril



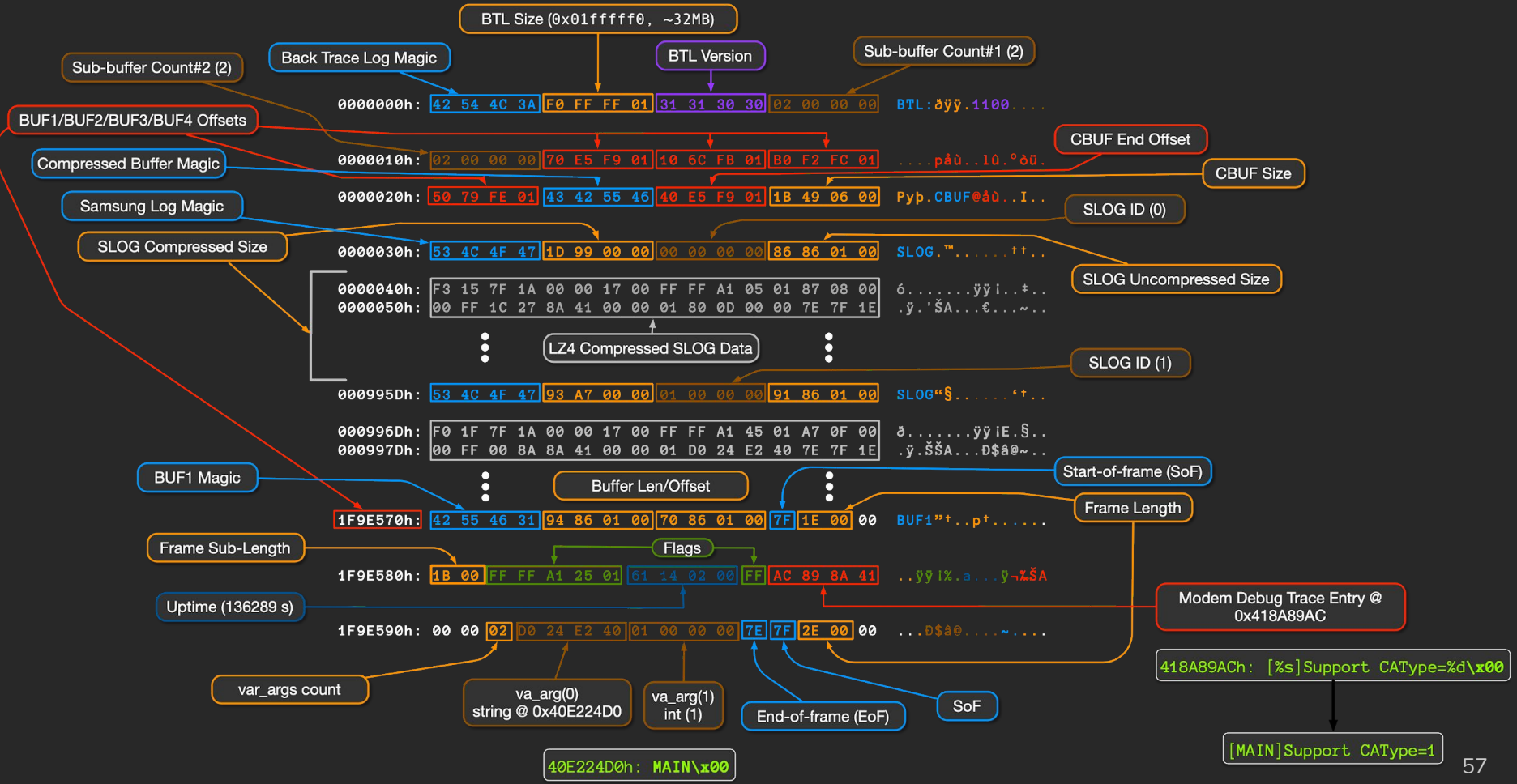
Replicant

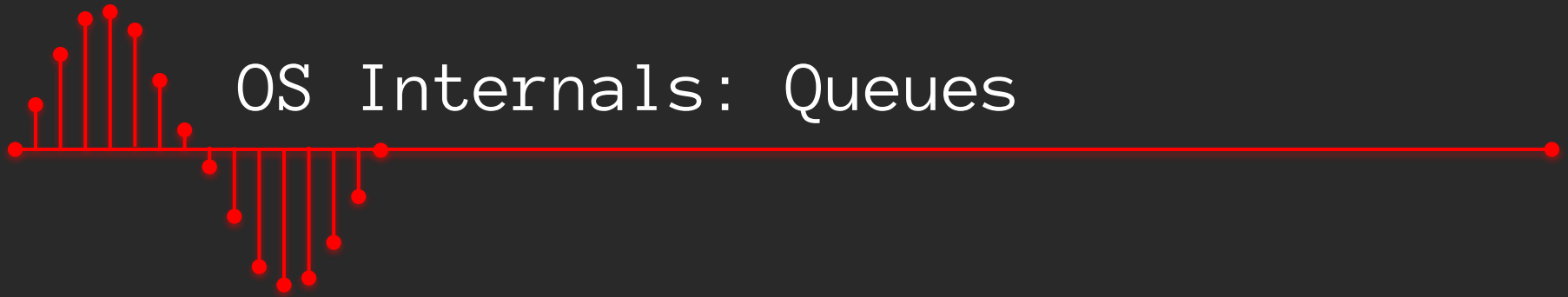


Backtrace Logging Format

Modem Backtrace Logging (BTL)

- Dumped on crash, but can be manually extracted (.BTL files)
- Reverse engineered BTL task
- LZ4 encoded log ring buffer in baseband memory
  - Log entries are pointers to format strings in modem_debug.bin
  - Varargs are pointers or literals, depending on format specifier
- Not 100% reverse engineered, but good enough to compare ShannonEE with real modem logging


BTL Size (0x01fffff0, ~32MB)
Back Trace Log Magic
BTL Version
Sub-buffer Count#1 (2)
Sub-buffer Count#2 (2)
BUF1/BUF2/BUF3/BUF4 Offsets
CBUF End Offset
Compressed Buffer Magic
CBUF Size
Samsung Log Magic
SLOG ID (0)
SLOG Compressed Size
SLOG Uncompressed Size
LZ4 Compressed SLOG Data
SLOG ID (1)
BUF1 Magic
Buffer Len/Offset
Start-of-frame (SoF)
Frame Length
Frame Sub-Length
Flags
Uptime (136289 s)
Modem Debug Trace Entry @ 0x418A89AC
var_args count
va_arg(0) string @ 0x40E224D0
va_arg(1) int (1)
End-of-frame (EoF)
SoF
0000000h: 42 54 4C 3A F0 FF FF 01 31 31 30 30 02 00 00 00 BTL:ðÿÿ.1100....
0000010h: 02 00 00 00 70 E5 F9 01 10 6C FB 01 B0 F2 FC 01 ....påù..lû.°òü.
0000020h: 50 79 FE 01 43 42 55 46 40 E5 F9 01 1B 49 06 00 Pyþ.CBUF@åù..I..
0000030h: 53 4C 4F 47 1D 99 00 00 00 00 00 00 86 86 01 00 SLOG.™......††..
0000040h: F3 15 7F 1A 00 00 17 00 FF FF A1 05 01 87 08 00 ó.......ÿÿ¡..‡..
0000050h: 00 FF 1C 27 8A 41 00 00 01 80 0D 00 00 7E 7F 1E .ÿ.'ŠA...€...~..
000995Dh: 53 4C 4F 47 93 A7 00 00 01 00 00 00 91 86 01 00 SLOG“§......‘†..
000996Dh: F0 1F 7F 1A 00 00 17 00 FF FF A1 45 01 A7 0F 00 ð.......ÿÿ¡E.§..
000997Dh: 00 FF 00 8A 8A 41 00 00 01 D0 24 E2 40 7E 7F 1E .ÿ.ŠŠA...Ð$â@~..
1F9E570h: 42 55 46 31 94 86 01 00 70 86 01 00 7F 1E 00 00 BUF1”†..p†......
1F9E580h: 1B 00 FF FF A1 25 01 61 14 02 00 FF AC 89 8A 41 ..ÿÿ¡%.a...ÿ¬‰ŠA
1F9E590h: 00 00 02 D0 24 E2 40 01 00 00 00 7E 7F 2E 00 00 ...Ð$â@....~....
418A89ACh: [%s]Support CAType=%d\x00
[MAIN]Support CAType=1
40E224D0h: MAIN\x00



OS Internals: Queues

Queues

- IPC is realized via Queues
- Queue messages allocated by sender and freed by called
- Over 290 queue destinations!
  - https://github.com/grant-h/ShannonBaseband/blob/master/firmware/extdata/G973FXXU3ASG8/pal_queues.h
- Message payload variable depending on msgID

```
struct qitem_header {
  union {
    struct {
      uint16_t src; // source queue
      uint16_t dst; // destination queue
    };
    uint32_t dir;
  };
  uint16_t size; // size of the payload
  uint16_t msgId; // [msgGroup:8][msgNumber:8]
} PACKED;
```


5G

What about 5G?

- Shannon also present in Samsung's Next-gen 5G Modem (51XX)
  - Present for instance in Galaxy S20

- Radical change or processor profile!
  - Cortex-R instead of Cortex-A
  - Full-blown MMU

- Lot's of new code for handling 5G:
  - At least 500k lines of new code
  - But not everything changed!