Basebanheimer

Now I Am Become Death, The Destroyer of Chains

TASZK
WE FIND NEEDLES

hardwear.io
Hardware Security Conference and Training

#whoami

- Daniel Komaromy @kutyacica
- Head of Research, TASZK Security Labs
- !SpamAndHex, Pwn2Own, Black Hat, Recon, Ekoparty, QCSS, etc.
- *There’s no crying in baseband* since 2010

Prior Art (Public)

Remote Code Execution

- RPW (Qualcomm 2010, Intel 2010)
- Nico Golde (Samsung 2015, Intel 2018)
- Marco Grassi and Kira

  (Huawei 2017, Mediatek 2019, Samsung 2021)
- Amat Cama (Samsung 2017/18, Intel 2018)
- Grant Hernandez, Dominik Maier, Marius Muench et al. @

  Ruhr Uni Bochum, TU Berlin, VU NL, UF (Samsung 2020-)
- Frédéric Basse (Samsung 2020)
- KAIST (Samsung 2022)
- Vendors (!)
- …
- Me and my teams (Samsung 2015, Huawei 2021, Mediatek 2022)

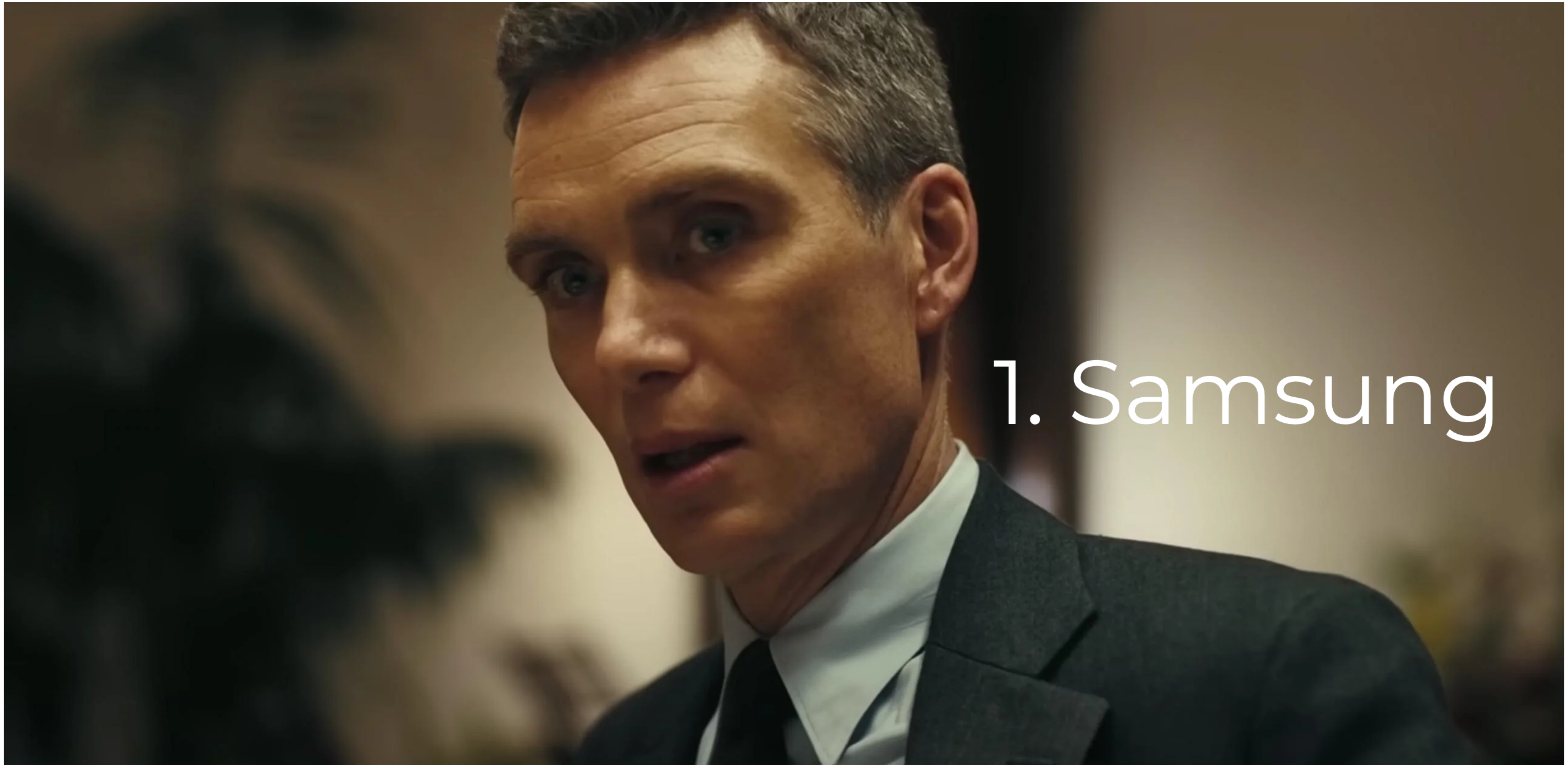
Sandbox Escape / AP Pivot

- Replicant “Backdoor” (Samsung 2014)
- Me and my teams (2017, 2021, 2022)
- …. and that’s it
- (related: Gal Beniamini Broadcom WiFi 2017; Tencent Qualcomm WiFi 2019 “we found a perfect pivot bug”, never(?) released it)

Goals

- New architectures, new baseband attack surfaces, new exploit techniques, complete chains to AP

- Targets: Samsung Exynos 21/2200 and Mediatek Dimensity


1. Samsung

Re-Breaking Band

- Shannon is the codename of the RTOS in Exynos basebands, the chip is a Cortex-A ARM
- Reverse engineering
  - Good news: firmware format, RTOS internals, CP Ramdump feature all largely intact since “Breaking Band” [Recon 2016, Nico Golde and myself]
  - Even better news: lot of public tooling
    - my 2016 released IDA tools, updated to Ghidra thanks to Grant Hernandez; various emu options
- Exploitation
  - Samsung made progress! (switched to MMU, NX fixed, SSP added)
  - No prior publication on heap exploitation

Step 1: Fail Immediately.

- 2023 March: dust off / upgrade RE tools for new chip, come up with attack surface, find a chain of RCE vulns, use new exploit approaches, report the completed exploit to vendor …

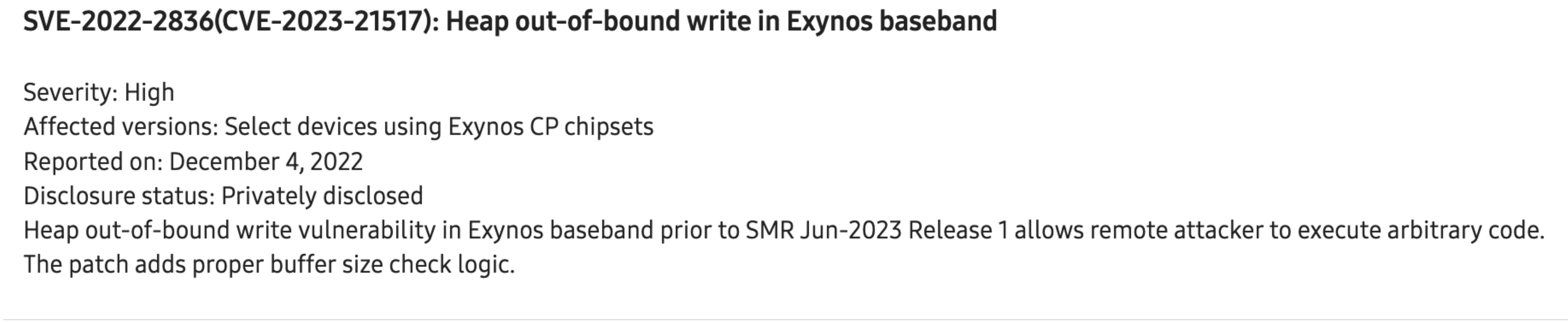
- … while waiting, not one but two Exynos baseband heap overflow exploits are published …

- … the embargo is still ongoing, can't present at hardwear.io

- Instead, I'll show an RCE vulnerability in 4G NAS that got fixed recently

CVE-2023-21517

- Heap OOB Write in LTE ESM
- Released in June 2023 bulletin, but wasn't discussed



SVE-2022-2836(CVE-2023-21517): Heap out-of-bound write in Exynos baseband

Severity: High
Affected versions: Select devices using Exynos CP chipsets
Reported on: December 4, 2022
Disclosure status: Privately disclosed
Heap out-of-bound write vulnerability in Exynos baseband prior to SMR Jun-2023 Release 1 allows remote attacker to execute arbitrary code.
The patch adds proper buffer size check logic.

CVE-2023-21517

- I reversed the patch based on the bulletin
- The bulletin description is not too informative… but the patch is!

```
 (…)
   trace_msg.id = uVar13;
 pal_LogMsg(&trace_msg,uVar19,&SUB_fecdba98);
 break;
}
if (0xd < (byte)puVar8[5]) {
       /* NumComponent = %d exceeds max -> SAEQM_SYNTACT_ERROR_IN_PKT_FILTER [%d] */
 trace_msg.dbt_m = &dbt_msg_42f6fde0;
 iVar7 = FUN_4133fac4(0xffffffff);
 if (iVar7 < 0) {
  uVar13 = 0x243;
 }
 else {
  (…)
 }
 trace_msg.id = uVar13;
 pal_LogMsg(&trace_msg,(uint)*(byte *)(*piVar1 + 5),uVar19,&SUB_fecdba98);
 break;
}
if (uVar13 == 3) {
 if ((1 << uVar19 & uVar16) != 0) {
       /* No Matched PID for Replacing --> No Error : Add pkt filter */
  trace_msg.dbt_m = &dbt_msg_42f6fe38;
  goto LAB_419a47c6;
 }
}
```

Traffic Flow Template IE

- 3GPP TS 24.008 10.5.6.12
- ACTIVATE DEDICATED EPS BEARER CONTEXT REQUEST, MODIFY EPS BEARER CONTEXT REQUEST
- Traffic Flow Templates > Packet Filters > Components
- Components define source and destination ports and/or IP ranges
- Specification doesn't simply set "Max Component Num", it defines mutual exclusivity rules
    - You can calculate the theoretical max if you follow the spec
- But the Shannon code neither followed the exclusivity rules nor verified the counter, ergo:

Maximum_Packet_Filter_Size / Min_Component_Size > Packet_Filter_Component_Array_Size

Heap Feng Sh(annon)ui

- Debugging the Shannon heap
  - Debug High Mode generates heap event trace in memory
  - CP Ramdump
- Heap Shaping with TFT primitives
  - Lifespan of TFT packet filter and component allocations are attacker controlled
  - Add, Delete at will individually with Modify/Delete TFT requests
  - Max 9 dedicated bearers, Max 15 Packet Filters for each TFT, Max 10 Components (w/o corruption) for each of those
  - Trivial to create checkered allocation patterns with this
  - Pool size is 2048 bytes in Shannon heap (32 slots for the smallest size), so 9*15*10 is more than enough for initial heap spraying too

What to overwrite?

- Technique published in 2023
  - classic unsafe unlinking write4 after corrupting 1st byte of chunk header to 0x01
  - doesn't work here as our writes are (mod4 zero LSB) pointers, so can't get 0x01
- Find allocations with callback fn / fn table pointers
- Find alternative linked list implementations, target a linked-list hijack against:
  - a single linked list to create a fake element
  - a vuln doubly linked list to get write4 when unsafe unlinking the inserted fake
- Overlap a size/offset field with the pointer's values
- … WIP.

Finding CP-to-AP Bugs

- RIL IPC is the most obvious / largest attack surface from the Cellular to the Application Processor

- Significant attention in years gone by, but in the other direction (AP->CP corruptions)

- Standard is AT commands, but most vendors have their own proprietary

IPC: I Pwn Cellular :)

- For Samsung, it’s SIPC

- Implemented by the Radio Interface Layer Daemon

- Bug hunting in Android C++ code in 2023

  - standard ELF shared libraries with symbols, root for free with Magisk, Frida support out of the box, hundreds of functions to look at instead of tens of thousands … man, this userspace stuff is amazing compared to basebands :)

  - tl;dr: manual reversing, following the flow from the read syscall to the parsers. Symbol names super helpful, `IpcModem::GetIpcMessage` etc.

CVE-2023-30644

Stack Buffer Overflow in CdmaSmsParser

```
CdmaSmsParser::CdmaSmsParser
   (CdmaSmsParser *this,CdmaSmsMessage *param_1,uchar *packet_data,int packet_len)

{
[...]

   *(int *)(this + 0x1c) = packet_len;
   if (1 < *(int *)puVar1) {
      __android_log_buf_print(1,3,uVar2,"mBearerDataLen : %d",*(undefined4 *)(this
      + 0x1c));
   }
   *(uchar **)(this + 0x438) = packet_data;

   // [1] The memcpy copies the IPC data into the stack buf without a length check
   memcpy(this + 0x38,packet_data,(long)packet_len);
   *(CdmaSmsMessage **)(this + 0x30) = param_1;
   if (*(int *)(this + 0x2c) != 0) {
      DisplayDebugBearerData(this);
      return;
   }
   return;
}
```

- Max IPC message size: 0x408000
- Stack buffer size: 1096

CVE-2023-30649

Heap Buffer Overflow in RmtUimNeedApdu

```
RmtUimNeedApdu * __thiscall
IpcProtocol41Sap::IpcRxPRSimApdu
   (IpcProtocol41Sap *this,sipc_ipc_msg *param_1,int param_2,int *param_3,
   RegistrantType *param_4)

{
   RmtUimNeedApdu *this_00;

   this_00 = (RmtUimNeedApdu *)operator.new(0x120);
   // [1] The length field is retrieved from the IPC packet without verification
   RmtUimNeedApdu::RmtUimNeedApdu
         (this_00,*(int *)(this + 0x70) + 1,0,
         (uint)*(ushort *)(param_1->data + 1),
         (char *)(param_1->data + 3));
   *param_4 = 0x8e;
   return this_00;

}
```

```
void __thiscall
RmtUimNeedApdu::RmtUimNeedApdu
   (RmtUimNeedApdu *this,int param_1,int param_2,int len,char *src)

{
   undefined *puVar1;
   // [2] The length field is retrieved from the IPC packet without
   verification
   memcpy(this + 0x18,src,(long)len);
   return;
}
```

- Max length: 0xFFFF
- Heap buffer size: 0x120

Crafting POCs

- Inject into fd directly using Frida
- Tweetable POCs, 2023 edition

```
09040000121400ffff + 41 * 0x400
```

Exploiting RILD in 2023

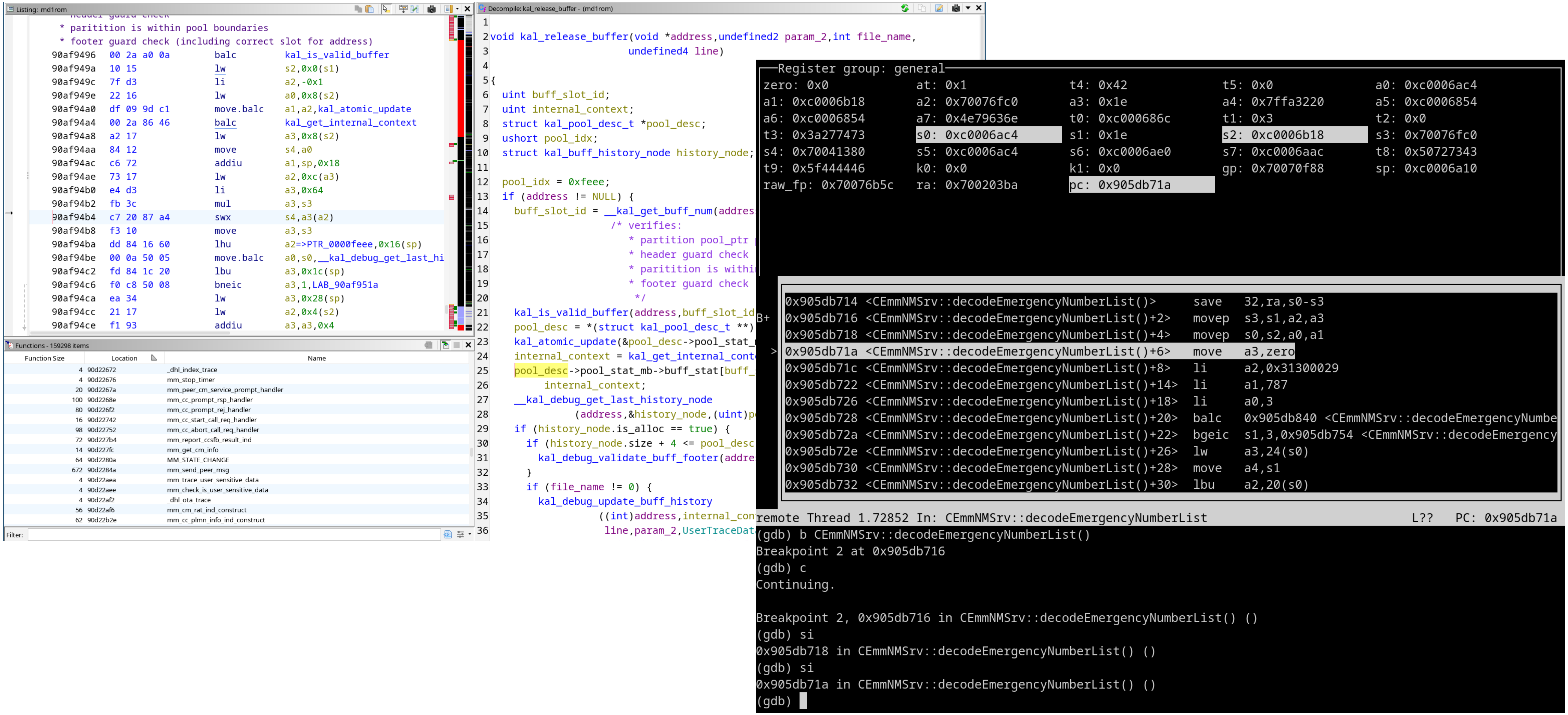
- Android remote with no javascript in 2023? SSP, Scudo, CFI, …
  - Good luck with that?!
  - Or maybe … we just need a good enough set of bugs.
- 6 bugs fixed in July 2023: 2 stack BOFs, 4 heap OF/OOB Write
- Some are under embargo
- … TBC another time.

2. Mediatek

nanoMIPS, (na)noProblem

- Mediatek Dimensity vs Helio
  - same Nucleus RTOS, same fw format
  - ISA changed from MIPS16e2 to nanoMIPS
- Had to rebuild our tool arsenal
  - implemented new modules for Ghidra, Qemu
  - roadmaps: our 2022 MIPS16e2 decompiler plugin, Ivan Fratric qdsp6 patch for qemu-libafl-bridge

nanoMIPS, (na)noProblem



Listing: md1rom

```
  * paritition is within pool boundaries
  * footer guard check (including correct slot for address)
90af9496   00 2a a0 0a    balc        kal_is_valid_buffer
90af949a   10 15          lw          s2,0x0(s1)
90af949c   7f d3          li          a2,-0x1
90af949e   22 16          lw          a0,0x8(s2)
90af94a0   df 09 9d c1    move.balc   a1,a2,kal_atomic_update
90af94a4   00 2a 86 46    balc        kal_get_internal_context
90af94a8   a2 17          lw          a3,0x8(s2)
90af94aa   84 12          move        s4,a0
90af94ac   c6 72          addiu       a1,sp,0x18
90af94ae   73 17          lw          a2,0xc(a3)
90af94b0   e4 d3          li          a3,0x64
90af94b2   fb 3c          mul         a3,s3
90af94b4   c7 20 87 a4    swx         s4,a3(a2)
90af94b8   f3 10          move        a3,s3
90af94ba   dd 84 16 60    lhu         a2=>PTR_0000feee,0x16(sp)
90af94be   00 0a 50 05    move.balc   a0,s0,__kal_debug_get_last_hi
90af94c2   fd 84 1c 20    lbu         a3,0x1c(sp)
90af94c6   f0 c8 50 08    bneic       a3,1,LAB_90af951a
90af94ca   ea 34          lw          a3,0x28(sp)
90af94cc   21 17          lw          a2,0x4(s2)
90af94ce   f1 93          addiu       a3,a3,0x4
```

Functions - 159298 items

Function Size	Location	Name
4	90d22672	_dhl_index_trace
4	90d22676	mm_stop_timer
20	90d2267a	mm_peer_cm_service_prompt_handler
100	90d2268e	mm_cc_prompt_rsp_handler
80	90d226f2	mm_cc_prompt_rej_handler
16	90d22742	mm_cc_start_call_req_handler
98	90d22752	mm_cc_abort_call_req_handler
72	90d227b4	mm_report_ccsfb_result_ind
14	90d227fc	mm_get_cm_info
64	90d2280a	MM_STATE_CHANGE
672	90d2284a	mm_send_peer_msg
4	90d22aea	mm_trace_user_sensitive_data
4	90d22aee	mm_check_is_user_sensitive_data
4	90d22af2	_dhl_ota_trace
56	90d22af6	mm_cm_rat_ind_construct
62	90d22b2e	mm_cc_plmn_info_ind_construct

Filter:

Decompile: kal_release_buffer - (md1rom)

```

void kal_release_buffer(void *address,undefined2 param_2,int file_name,
                       undefined4 line)

{
  uint buff_slot_id;
  uint internal_context;
  struct kal_pool_desc_t *pool_desc;
  ushort pool_idx;
  struct kal_buff_history_node history_node;

  pool_idx = 0xfeee;
  if (address != NULL) {
    buff_slot_id = __kal_get_buff_num(addres
                    /* verifies:
                       * partition pool_ptr
                       * header guard check
                       * paritition is withi
                       * footer guard check
                        */
    kal_is_valid_buffer(address,buff_slot_id
    pool_desc = *(struct kal_pool_desc_t **)
    kal_atomic_update(&pool_desc->pool_stat_
    internal_context = kal_get_internal_conte
    pool_desc->pool_stat_mb->buff_stat[buff_
         internal_context;
    __kal_debug_get_last_history_node
              (address,&history_node,(uint)p
    if (history_node.is_alloc == true) {
      if (history_node.size + 4 <= pool_desc
        kal_debug_validate_buff_footer(addre
      }
      if (file_name != 0) {
        kal_debug_update_buff_history
                  ((int)address,internal_con
                   line,param_2,UserTraceDat
```

```
┌─Register group: general─
 zero: 0x0              at: 0x1               t4: 0x42              t5: 0x0               a0: 0xc0006ac4
 a1: 0xc0006b18         a2: 0x70076fc0        a3: 0x1e              a4: 0x7ffa3220        a5: 0xc0006854
 a6: 0xc0006854         a7: 0x4e79636e        t0: 0xc000686c        t1: 0x3               t2: 0x0
 t3: 0x3a277473         s0: 0xc0006ac4        s1: 0x1e              s2: 0xc0006b18        s3: 0x70076fc0
 s4: 0x70041380         s5: 0xc0006ac4        s6: 0xc0006ae0        s7: 0xc0006aac        t8: 0x50727343
 t9: 0x5f444446         k0: 0x0               k1: 0x0               gp: 0x70070f88        sp: 0xc0006a10
 raw_fp: 0x70076b5c     ra: 0x700203ba        pc: 0x905db71a

     0x905db714 <CEmmNMSrv::decodeEmergencyNumberList()>     save   32,ra,s0-s3
B+   0x905db716 <CEmmNMSrv::decodeEmergencyNumberList()+2>   movep  s3,s1,a2,a3
     0x905db718 <CEmmNMSrv::decodeEmergencyNumberList()+4>   movep  s0,s2,a0,a1
  >  0x905db71a <CEmmNMSrv::decodeEmergencyNumberList()+6>   move   a3,zero
     0x905db71c <CEmmNMSrv::decodeEmergencyNumberList()+8>   li     a2,0x31300029
     0x905db722 <CEmmNMSrv::decodeEmergencyNumberList()+14>  li     a1,787
     0x905db726 <CEmmNMSrv::decodeEmergencyNumberList()+18>  li     a0,3
     0x905db728 <CEmmNMSrv::decodeEmergencyNumberList()+20>  balc   0x905db840 <CEmmNMSrv::decodeEmergencyNumbe
     0x905db72a <CEmmNMSrv::decodeEmergencyNumberList()+22>  bgeic  s1,3,0x905db754 <CEmmNMSrv::decodeEmergency
     0x905db72e <CEmmNMSrv::decodeEmergencyNumberList()+26>  lw     a3,24(s0)
     0x905db730 <CEmmNMSrv::decodeEmergencyNumberList()+28>  move   a4,s1
     0x905db732 <CEmmNMSrv::decodeEmergencyNumberList()+30>  lbu    a2,20(s0)
remote Thread 1.72852 In: CEmmNMSrv::decodeEmergencyNumberList                      L??   PC: 0x905db71a
(gdb) b CEmmNMSrv::decodeEmergencyNumberList()
Breakpoint 2 at 0x905db716
(gdb) c
Continuing.

Breakpoint 2, 0x905db716 in CEmmNMSrv::decodeEmergencyNumberList() ()
(gdb) si
0x905db718 in CEmmNMSrv::decodeEmergencyNumberList() ()
(gdb) si
0x905db71a in CEmmNMSrv::decodeEmergencyNumberList() ()
(gdb) 
```

RCE Prior Art

- RPW (Qualcomm 2010, Intel 2010)
- Nico Golde (Samsung 2015, Intel 2018)
- Marco Grassi and Kira (Huawei 2017, Mediatek 2019, Samsung 2021)
- Amat Cama (Samsung 2017/18, Intel 2018)
- Grant Hernandez, Dominik Maier et al @ Ruhr Uni Bochum, TU Berlin (Samsung 2020-)
- Frédéric Basse (Samsung 2020)
- KAIST (Samsung 2022)
- Vendors
- Me and my teams (Samsung 2016, Huawei 2021, Mediatek 2022)

RCE Prior Art: Layer 2

- ~~RPW (Qualcomm 2010, Intel 2010)~~
- ~~Nico Golde (Samsung 2015, Intel 2018)~~
- ~~Marco Grassi and Kira (Huawei 2017, Mediatek 2019, Samsung 2021)~~
- ~~Amat Cama (Samsung 2017/18, Intel 2018)~~
- ~~Grant Hernandez, Dominik Maier et al @ Ruhr Uni Bochum, TU Berlin (Samsung 2020-)~~
- ~~Frédéric Basse (Samsung 2020)~~
- ~~KAIST (Samsung 2022)~~
- ~~Vendors~~
- ~~Me and my teams (Samsung 2016, Huawei 2021, Mediatek 2022)~~

RCE Prior Art: Layer 2

“Below layer 3, there usually is little potential for exploitable memory corruptions, as the messages transmitted are too short.”

- RPW, Usenix WOOT 2012

RCE Prior Art: Layer 2

CHALLENGE ACCEPTED

Pwning on the EDGE

- Disclosure timeline
  - Done on Helio (MIPS16e2), verified bug liveness in nanoMIPS disassembly on Dimensity
  - One month (January 2022), reported bugs start of February 2022
  - Fixes: July 2022 Bulletin
  - Dimensity tool building done in 2023
  - I can disclose the findings in this talk \o/

CVE-2022-21744

- “In Modem 2G RR, there is a possible out of bounds write (...) when decoding GPRS Packet Neighbour Cell Data (PNCD) improper neighbouring cell size“
  - It’s a day ending in Y, so the bulletin description is wrong
  - PNCD is in RLC not RR, there is no such thing as a “cell size” in a PNCD message
- 3GPP TS 44.060 11.2.9e

CVE-2022-21744

- PNCD is an optional message sent by the network in an RLC Control Block on the PACCH to provide system information required for initial access in a neighbouring cell

```
< Packet Neighbour Cell Data message content > ::=
  < PAGE_MODE : bit (2) >
  {0 <GlobalTFI:<GlobalTFIIE>>
    { < CONTAINER_ID : bit (2) >
      < spare : bit (1)
      < CONTAINER_INDEX : bit (5) >
      {0|1 <ARFCN:bit(10)>
           < BSIC : bit (6) > }
      < CONTAINER : < Container repetition struct > >
      < padding bits >
      ! < Non-distribution part error : bit (*) = < no string > > }
    ! < Address information part error : bit (*) = < no string > }
  ! < Distribution part error : bit (*) = < no string > > ;

< Container repetition struct > ::=
 {
   {   <PD:bit(3)>
       < CD_LENGTH : { bit (5) exclude 00000 exclude 11111 } >
       < CONTAINER_DATA : octet (val(CD_LENGTH)) > -- Final container segment. Next container follows.

   |   <PD:bit(3)>
       < CD_LENGTH : { bit (5) := 11111 } >
       < CONTAINER_DATA : octet ** > } ** -- Container continued in next message.

   {   < spare bit (3) >                         -- Repetition of the container repetition struct continues until:
       < CD_LENGTH : { bit (5) := 00000 } > }    -- A) val(CD_LENGTH) = 0 or
 }//;                                            -- B) end of PNCD message.
```

```
void FDD_rr_put_pncd_qmsg(char *src, uint size, queue_t *qmsg){
  // allocate the pointer array if this is the first element
  if (qmsg->current_num == 0)(
    // this is enough space for 32 pointers
    qmsg->ptr_to_msg_ptr_array = get_ctrl_buffer_ext(0x80);
    qmsg->valid = 1;
  }

  // allocate and zero-init memory for current message content
  ptr_to_msg_ptr_array[current_num] = get_ctrl_buffer_ext(28);
  memset(ptr_to_msg_ptr_array[current_num], 0, 28);

  // copy the data (size is always 21
  // when called from PNCD functions)
  memcpy(ptr_to_msg_ptr_array[current_num]->msg, src, size);
  ptr_to_msg_ptr_array[current_num]->msg_len = size * 8;
  ptr_to_msg_ptr_array[current_num]->offset = 0;

  // advance the queue state - no max check
  qmsg->current_num += 1;
  qmsg->total_num = qmsg->current_num;
}
```

Crafting POCs

- Delivery: modified Osmocom, added arbitrary RLC Control Block injection support

- Verification: `dmesg` fatal error on baseband crash

```
[ 1700.716689] (4)[220:ccci_fsm1][ccci1/fsm]MD exception stage 2!
[ 1700.716765] (4)[220:ccci_fsm1][ccci1/mcd]md_boot_stats len 2
[ 1700.716797] (4)[220:ccci_fsm1][ccci1/mcd]md_boot_stats0 / 1:0x5443000C / 0x53320000
[ 1700.717979] (4)[220:ccci_fsm1][ccci0/mcd]settle = 0; ret = 18446744073709551600
[ 1700.738359] (4)[220:ccci_fsm1][ccci0/mcd]settle = 0; ret = 0
[ 1700.738637] (4)[220:ccci_fsm1][mt gic dump] irq = 110\x0a[mt gic dump] enable = 1\x0a[mt gic dump] group = 1 (0x1:irq,0x0:fiq)
\x0a[mt gic dump] priority = d0\x0a[mt gic dump] sensitivity = 1 (edge:0x1, level:0x0)\x0a[mt gic dump] pending = 0\x0a[mt gic
dump] active status = 0\x0a[mt gic dump] tartget cpu mask = 0xffff
[ 1700.738702] (4)[220:ccci_fsm1][ccci1/fsm]mdee_info_prepare_v5, ee_case(0x0)
[ 1700.740593] (4)[220:ccci_fsm1][ccci1/fsm]brief_info: core_name = MCU_core1,vpe0,tc0(VPE3)
[ 1700.740604] (4)[220:ccci_fsm1][ccci0/fsm]offender: MD Offender:RR_FDD\x0a
[ 1700.740633] (4)[220:ccci_fsm1][ccci1/fsm]fatal error code 1,2,3 = [0x00000840, 0x289691C8, 0xCCCCCCCC]MD Offender:RR_FDD\x0a
[ 1700.744621] (4)[220:ccci_fsm1][ccci1/fsm]ccci_aed_v5 end!
[ 1700.744642] (4)[220:ccci_fsm1][ccci1/fsm]MD exception stage 2:end
[ 1700.744667] (4)[220:ccci_fsm1][ccci1/fsm]command 4 is completed 1 by fsm_routine_exception
```

Heap OFs in Nucleus

- Heap implementation described in detail in 2022 TASZK Security Labs blog post
  - includes generic heap metadata exploit techniques we came up with
  - the CVE-2022-21744 corruption primitive was similar to the Samsung one anyway, previous metadata techniques didn’t apply
- For 2023, Mediatek made changes, to address my described attacks!
- Cut for time, but check out our upcoming blog post on this

CP-to-AP: One and Done

- Linux Kernel attack surface: smaller, but more powerful!
- RIL IPC is built on top of shared memory communication
- Implemented with ringbuffers by the Kernel
- Ringbuffer control fields (tx/rx offsets and lengths) are also in the shared memory
- Just one thing to get right …

CVE-2022-21765

Stack BOF in ccci_ringbuf_readable

```
int ccci_ringbuf_readable(int md_id, struct ccci_ringbuf *ringbuf)
{
  unsigned char *rx_buffer, *outptr;
  unsigned int read, write, ccci_pkg_len, ccif_pkg_len;
  unsigned int footer_pos, length;
  unsigned int header[2] = { 0 };
  unsigned int footer[2] = { 0 };
  int size;

  [...]
  // [0] Offsets and lengths are retrieved from SHMEM
  read = (unsigned int)(ringbuf->rx_control.read);
  write = (unsigned int)(ringbuf->rx_control.write);
  length = (unsigned int)(ringbuf->rx_control.length);
  rx_buffer = ringbuf->buffer;
  size = write - read;
  if (size < 0)
    size += length;

  CCCI_DEBUG_LOG(md_id, TAG,
  "rbrdb:rbf=%p,rx_buf=0x%p,read=%d,write=%d,len=%d\n",
  ringbuf, rx_buffer, read, write, length);

  // [1] Size can be an arbitrary controlled value, this check can pass
  if (size < CCIF_HEADER_LEN + CCIF_FOOTER_LEN + CCCI_HEADER_LEN)
    return -CCCI_RINGBUF_EMPTY;
  outptr = (unsigned char *)header;
  // [2] The header is read into the stack buffer
  CCIF_RBF_READ(rx_buffer, outptr, CCIF_HEADER_LEN, read, length);
  [...]
}
```

```
#define CCIF_RBF_READ(bufaddr, output_addr, read_size, read_pos,
buflen)\
do {\
  // [3] If read_pos (rx-read) is larger than buflen (rx-length)
the else branch get executed
  if (read_pos + read_size < buflen) {\
    rbf_memcpy((unsigned char *)output_addr,\
      (unsigned char *)(bufaddr) + read_pos, read_size);\
  } else {\
    // [4] If read_pos (rx-read) larger than buflen (rx-length)
the size underflows, resulting in a stack overflow
    rbf_memcpy((unsigned char *)output_addr,\
    (unsigned char *)(bufaddr) + read_pos, buflen - read_pos);\
    output_addr = (unsigned char *)output_addr + buflen -
read_pos;\
    rbf_memcpy((unsigned char *)output_addr, \
      (unsigned char *)(bufaddr),\
      read_size - (buflen - read_pos));\
  } \
} while (0)
```

Exploitable? Meh

- No stack cookie, no KASLR, controlled values written. LFG?
- Integer wrap means the copy size is always a negative value
- Guaranteed to write way beyond the end of the stack
- In the normal case, a kernel thread stack frame allocated with vmalloc is 4 pages plus a guard page
- Maybe we could preempt the execution, maybe on some devices there's no guard page …
- … cope harder. Forget this, let's find something better!

CVE-2022-21766/21769

- OOB R/W in ccci_ringbuf_read/write

```
int ccci_ringbuf_write(int md_id, struct ccci_ringbuf *ringbuf,
unsigned char *data, int data_len)
{
   int aligned_data_len;
   unsigned int read, write, length;
   unsigned char *tx_buffer;
   unsigned char *h_ptr;

   unsigned int header[2] = { CCIF_PKG_HEADER, 0x0 };
   unsigned int footer[2] = { CCIF_PKG_FOOTER, CCIF_PKG_FOOTER };

   if (ringbuf == NULL || data_len == 0 || data == NULL)
      return -CCCI_RINGBUF_PARAM_ERR;
   if (ccci_ringbuf_writeable(md_id, ringbuf, data_len) <= 0)
      return -CCCI_RINGBUF_NOT_ENOUGH;

   // [0] Offsets and lengths are retrieved from SHMEM
   read = (unsigned int)(ringbuf->tx_control.read);
   write = (unsigned int)(ringbuf->tx_control.write);
   length = (unsigned int)(ringbuf->tx_control.length);

   // [1] Tx buffer location is calculated
   tx_buffer = ringbuf->buffer + ringbuf->rx_control.length;
   header[1] = data_len;
   h_ptr = (unsigned char *)header;

   // [2] Header is written to tx_buffer + write offset
   CCIF_RBF_WRITE(tx_buffer, h_ptr, CCIF_HEADER_LEN, write, length);
   write += CCIF_HEADER_LEN;

   // [3] length value is also controlled
   if (write >= length)
      write -= length;

   // [4] message data is written to tx_buffer + write offset
   CCIF_RBF_WRITE(tx_buffer, data, data_len, write, length);

   [...]

   return data_len;

}

#define CCIF_RBF_WRITE(bufaddr, data_addr, data_size, write_pos, buflen)\
   do {\
      // [5] Since the length is controlled this check can always pass
      if (write_pos + data_size < buflen) {\
         rbf_memcpy((unsigned char *)(bufaddr) + write_pos,\
               (unsigned char *)data_addr, data_size);\
      } else {\
         rbf_memcpy((unsigned char *)(bufaddr) + write_pos,\
               (unsigned char *)data_addr,  buflen - write_pos);\
         data_addr = (unsigned char *)data_addr + buflen - write_pos;\
         rbf_memcpy((unsigned char *)(bufaddr), (unsigned char *)data_addr,\
               data_size - (buflen - write_pos));\
      } \
   } while (0)
```

Exploiting CCCI

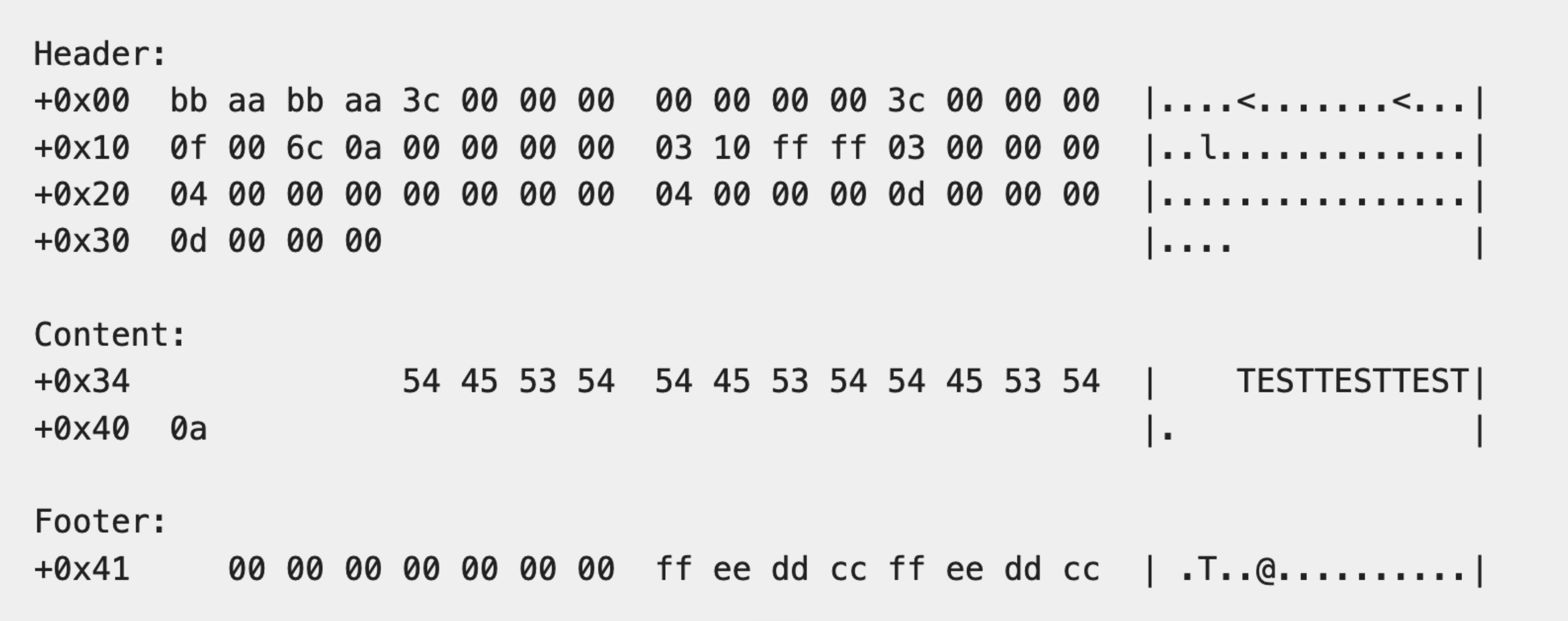
- Controlled write/read address, up to 2*UINT_MAX offset
- But these are actions done by the Kernel, not us!
- How to control the written data / leak back OOB read?
- How to trigger at the right moment?
- What can we target with an overwrite?

Exploiting CCCI

- Several ringbuffers, some very noisy
- But: a dedicated channel for “Remote Filesystem” operations (for storing NVRAM items), not busy after init
- Use the regular File Write API exposed to the baseband to prepare fix data
- Trigger File Read API to read it “back” - to the desired address via the OOB write
- Leak Kernel memory in reverse

Overwrite Values

- Written value almost entirely controlled
- Ringbuf writes include a custom header and footer, have to take that into consideration



```
Header:
+0x00  bb aa bb aa 3c 00 00 00  00 00 00 00 3c 00 00 00  |....<.......<...|
+0x10  0f 00 6c 0a 00 00 00 00  03 10 ff ff 03 00 00 00  |..l.............|
+0x20  04 00 00 00 00 00 00 00  04 00 00 00 0d 00 00 00  |................|
+0x30  0d 00 00 00                                       |....            |

Content:
+0x34              54 45 53 54  54 45 53 54 54 45 53 54  |    TESTTESTTEST|
+0x40  0a                                                |.               |

Footer:
+0x41     00 00 00 00 00 00 00  ff ee dd cc ff ee dd cc  | .T..@..........|
```

Overwrite Targets

- Helio kernels are still 32 bit, have no KASLR: WIN
- Dimensity kernels have KASLR, 64 bit
  - the ringbuffer is iomapped
  - iomap uses same address space as vmalloc/vmap!
  - no KASLR on this address, only the entropy of order of vmallocations
  - scan /proc/vmallocinfo, look for targets
  - inspired by Brandon Azad (2020 P0)

Overwrite Targets

- Our allocation is in a fix address always (very early vmalloc)
- Ton of viable targets within range: thread stacks, bpf programs, …
- Not at constant offsets - but we can also read!

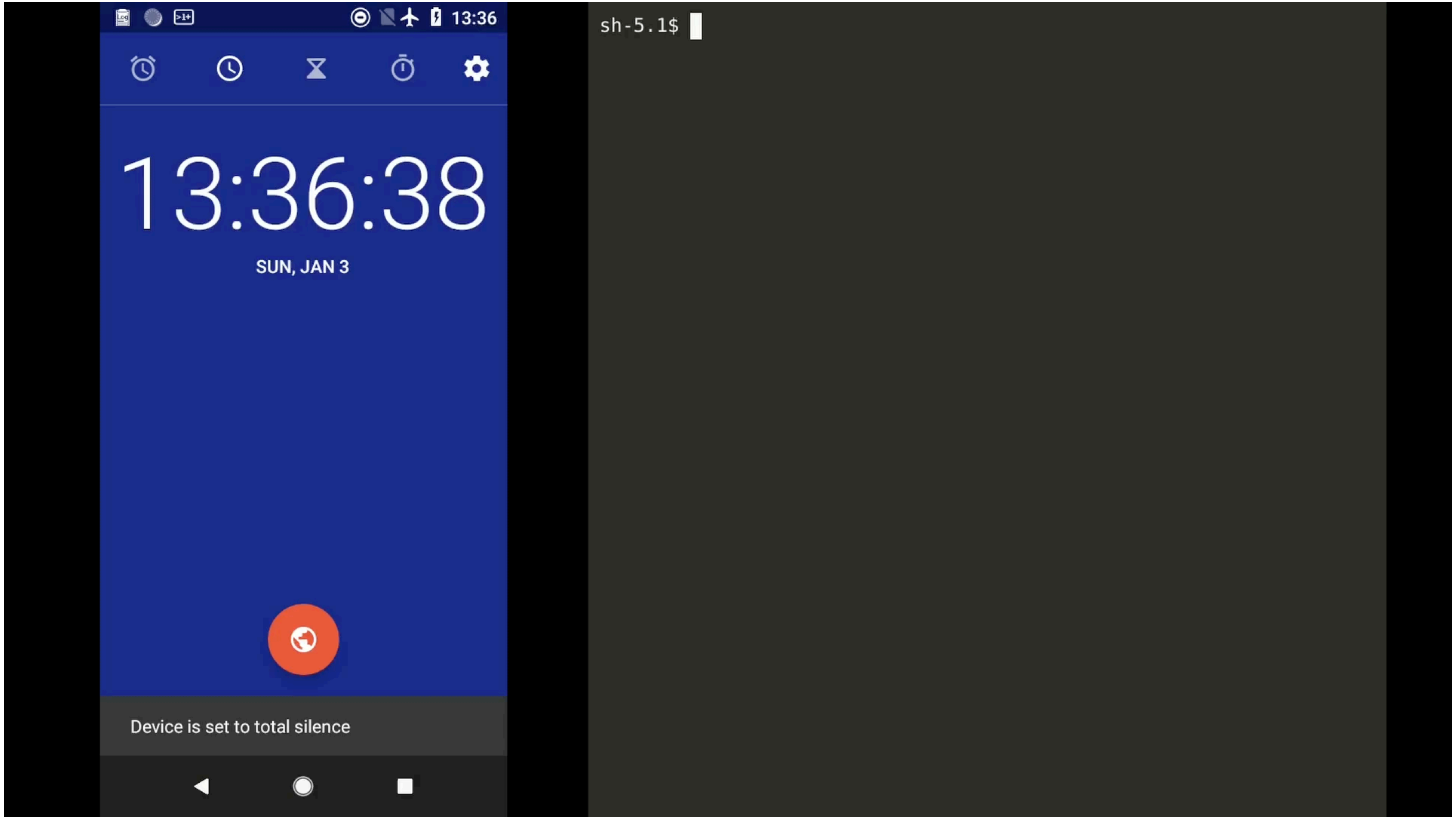
```
0xffffff800ad01000-0xffffff800aeb2000 1773568 vmap_reserved_mem+0xe8/0x124 vmap
0xffffff800aeb2000-0xffffff800aeb4000    8192 mt_i2c_probe+0x638/0x11fc phys=0x000000011d10000 ioremap
0xffffff800aeb4000-0xffffff800aeb6000    8192 mt_i2c_probe+0x100c/0x11fc user
0xffffff800aeb6000-0xffffff800aeb8000    8192 mt_i2c_probe+0x5a0/0x11fc phys=0x000000011d21000 ioremap
0xffffff800aeb8000-0xffffff800aebd000   20480 _do_fork+0x88/0x4a8 pages=4 vmalloc
0xffffff800aebd000-0xffffff800aebf000    8192 mt_i2c_probe+0x61c/0x11fc phys=0x0000000010217000 ioremap
0xffffff800aec0000-0xffffff800aec5000   20480 _do_fork+0x88/0x4a8 pages=4 vmalloc
(…)
0xffffff800eef9000-0xffffff800ef00000   28672 kbase_ioctl+0x1d34/0x2750 pages=6 vmalloc
0xffffff800ef98000-0xffffff800ef9d000   20480 _do_fork+0x88/0x4a8 pages=4 vmalloc
0xffffff800ef9d000-0xffffff800ef9f000    8192 bpf_prog_create_from_user+0x5c/0x1e4 pages=1 vmalloc
(…)
0xffffff800f000000-0xffffff800f801000 8392704 do_one_initcall+0x170/0x310 phys=0x000000008d000000 ioremap
```

Overwrite Targets

- Also some at constant offsets!
  - e.g. a do_one_initcall allocated during kernel module initialization can leak kernel addresses:

```
ffffff800f000000  02 00 00 00 00 00 00 00  c0 2e 27 00 00 00 00 00  ..........'.....
ffffff800f000010  00 00 00 00 00 00 00 00  03 00 00 00 00 00 00 00  ................
ffffff800f000020  0b 00 00 00 00 00 00 00  00 f0 1d 8d 00 00 00 00  ................
ffffff800f000030  e8 ff 04 00 00 00 00 00  80 2e c0 a1 e5 ff ff ff  ................   <<<
ffffff800f000040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
ffffff800f000050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
ffffff800f000060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
```

Demo Time



13:36:38

SUN, JAN 3

Device is set to total silence

```
sh-5.1$ 
```

Questions

Thank you!

@TaszkSecLabs
@kutyacica
@kutyacica.bsky.social

https://labs.taszk.io

https://taszk.io/contact