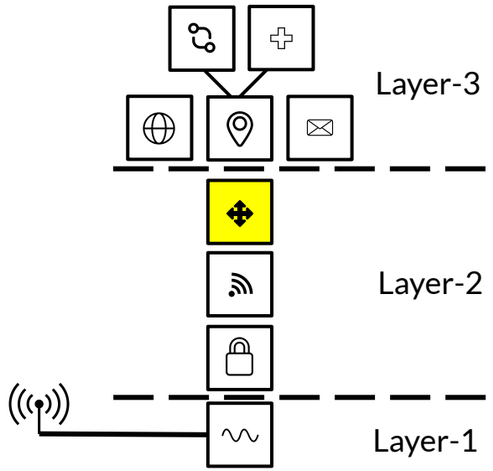
Fuzzing GPRS Layer-2 for Fun and Profit

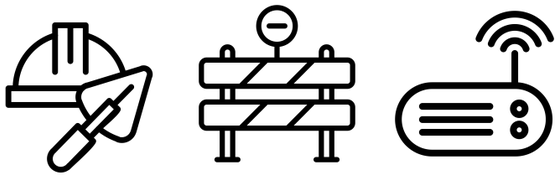
hardwear.io

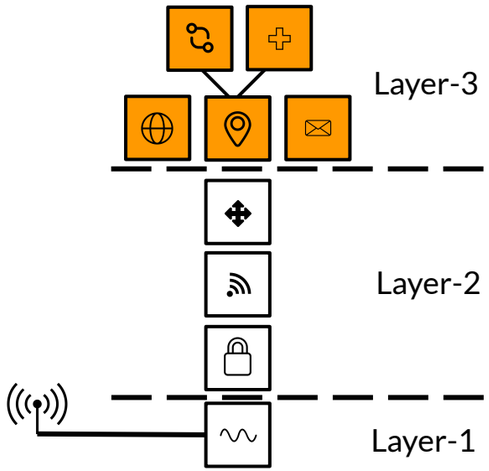
Hardware Security Conference and Training

Dyon Goos & Marius Muench

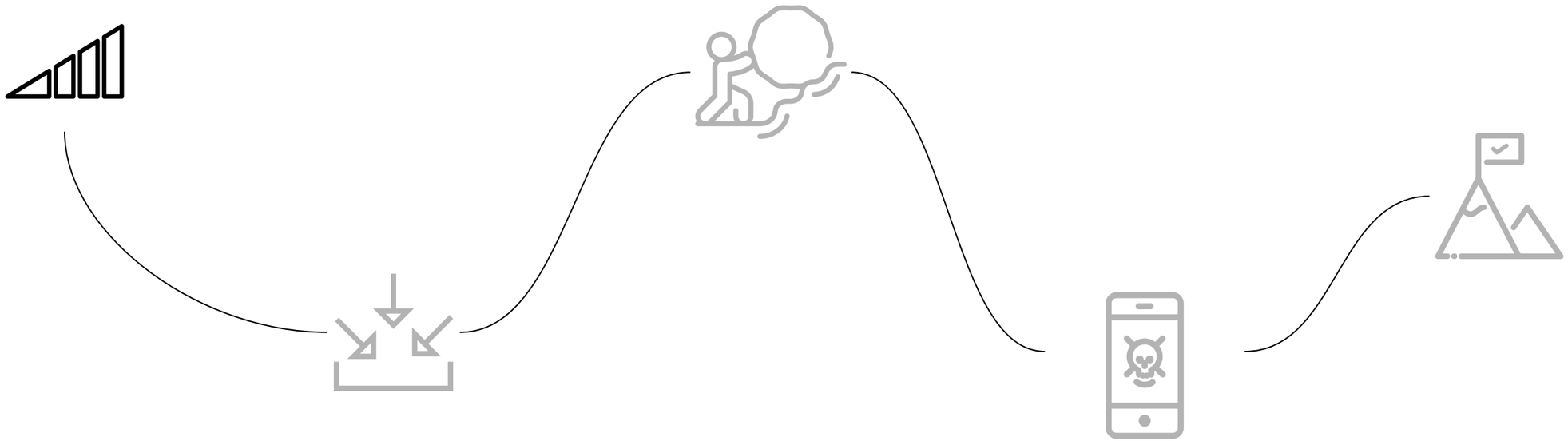
This talk





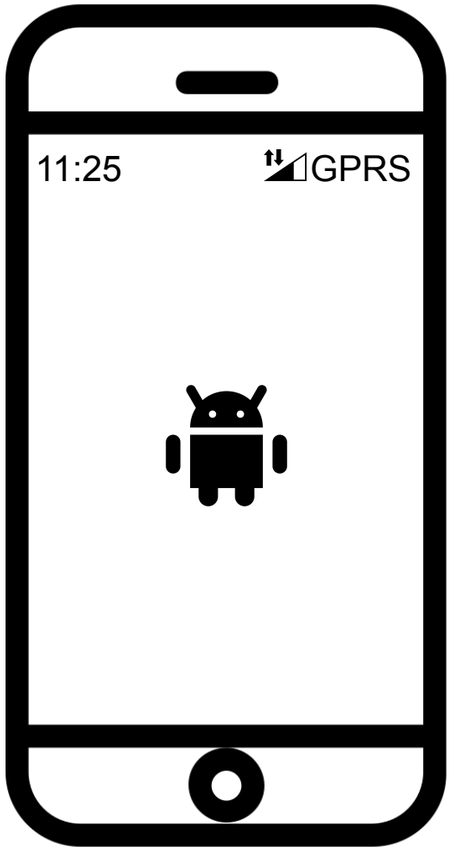


Basebands



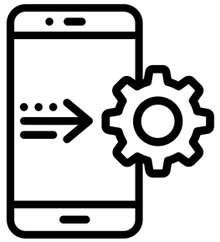
Basebands

- Modern phones are a collection of processors
  - Including: Application Processor (AP) & Cellular Processor (CP)
- CP also referred to as “Baseband”
  - Implements most layers of cellular communication stack
- Lucrative attack surface
  - Myriad of parsers, legacy code, obscure features



The code running on basebands

Custom Real-Time Operating Systems (RTOS), providing:

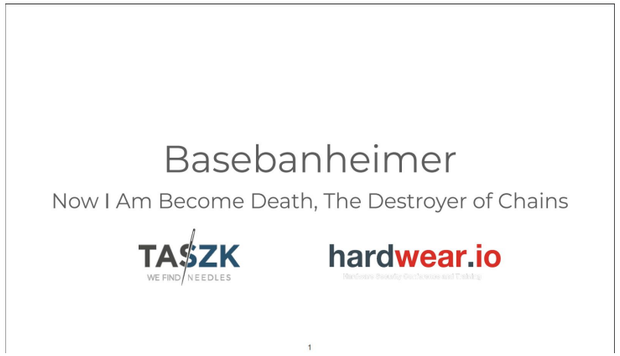


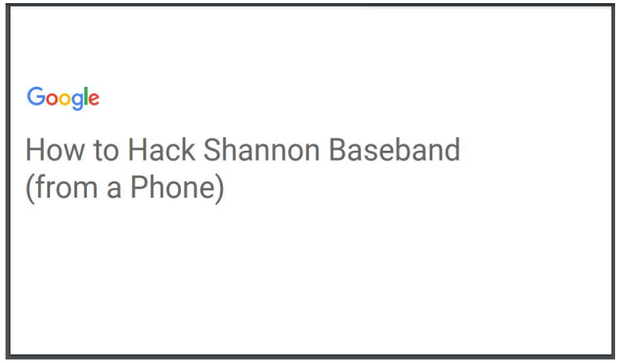
- Core OS functionality:
  - Scheduler, timers, interrupts
  - Messaging

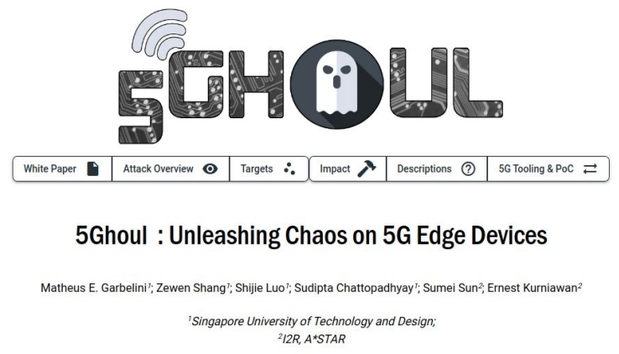
- Cellular stack implementation:
  - Stack is split into “tasks”
  - Tasks communicate via message queues

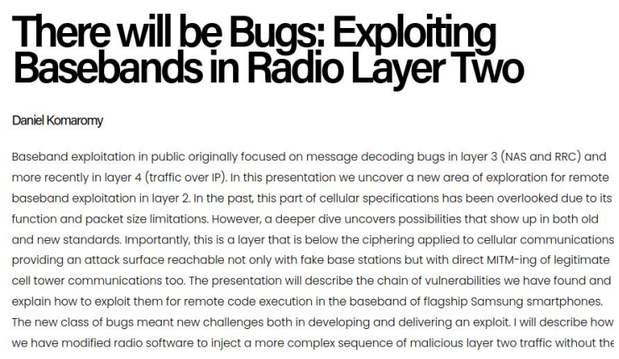
Baseband Security Research

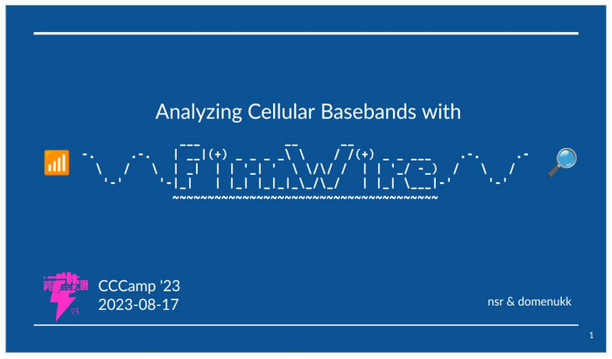
Plenty of attention in recent years, e.g.:

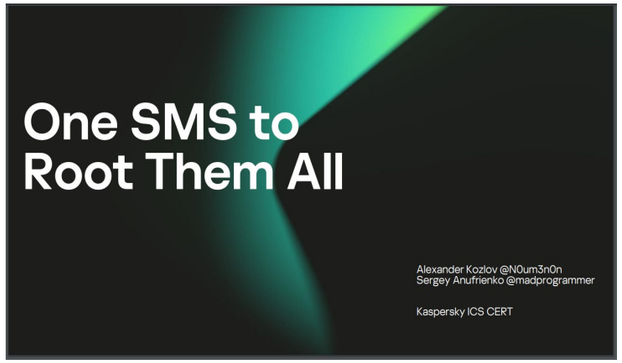












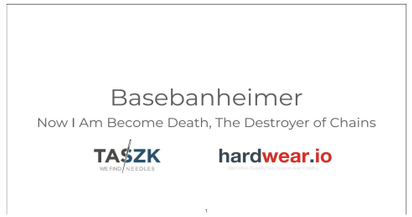
What about Layer-2?

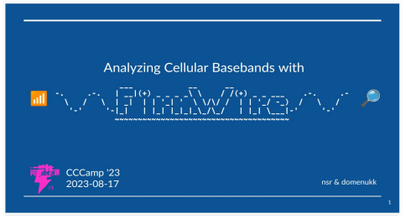
When we started, most research/findings focus on cellular L3 (or higher)

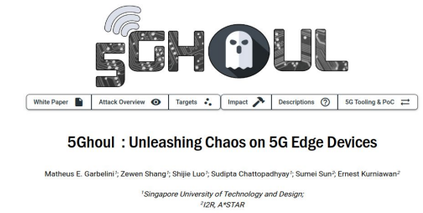
🤔 Let's have a look at layer-2 ourselves!

⇒ Let's start with legacy stacks first:

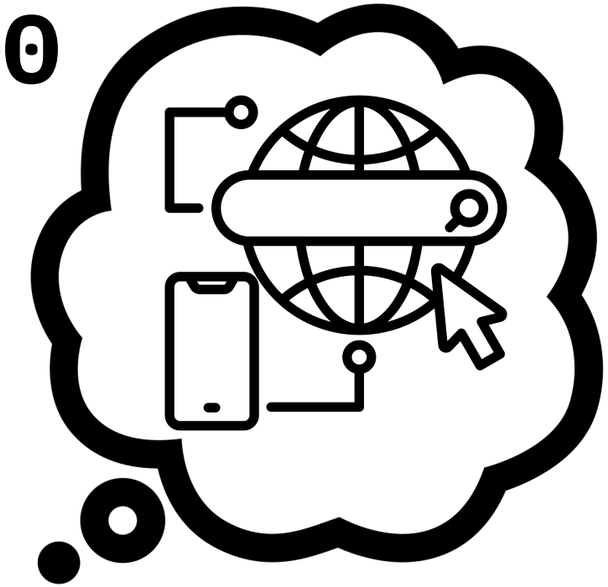
- GSM Layer-2
- GPRS Layer-2







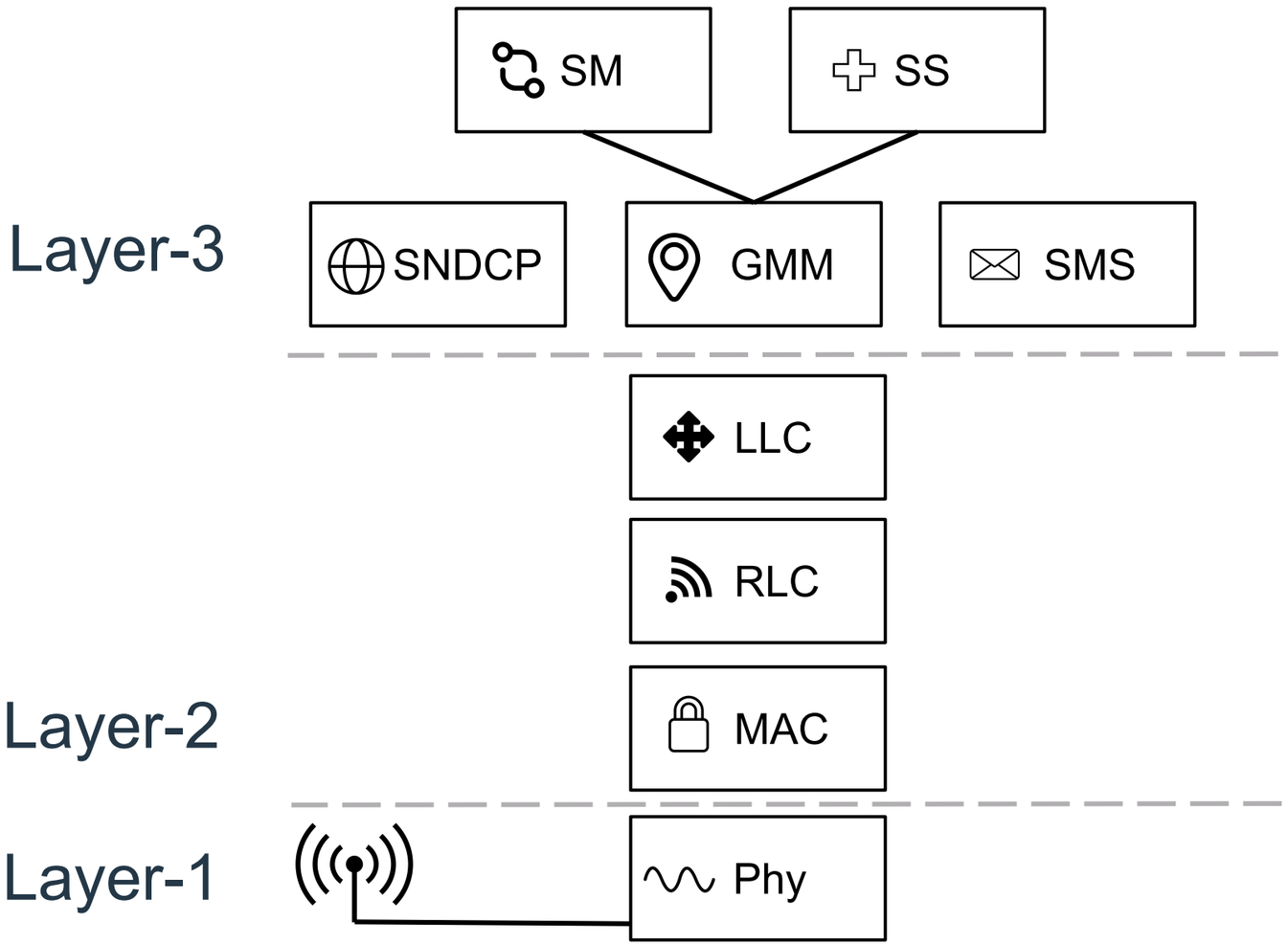
Around 2000



GSM: Circuit Switched

Solution : GSM++ aka GPRS aka General Packet Radio Service aka 2.5G

GPRS Protocol stack



SM: Session Management

SS: Supplementary Services

SNDCP: Sub Network Dependent Convergence Protocol

GMM: GPRS Mobility Management

SMS : Short Messaging Service

LLC: Logical Link Control

RLC: Radio Link Control

MAC: Medium Access Control

Phy : Physical

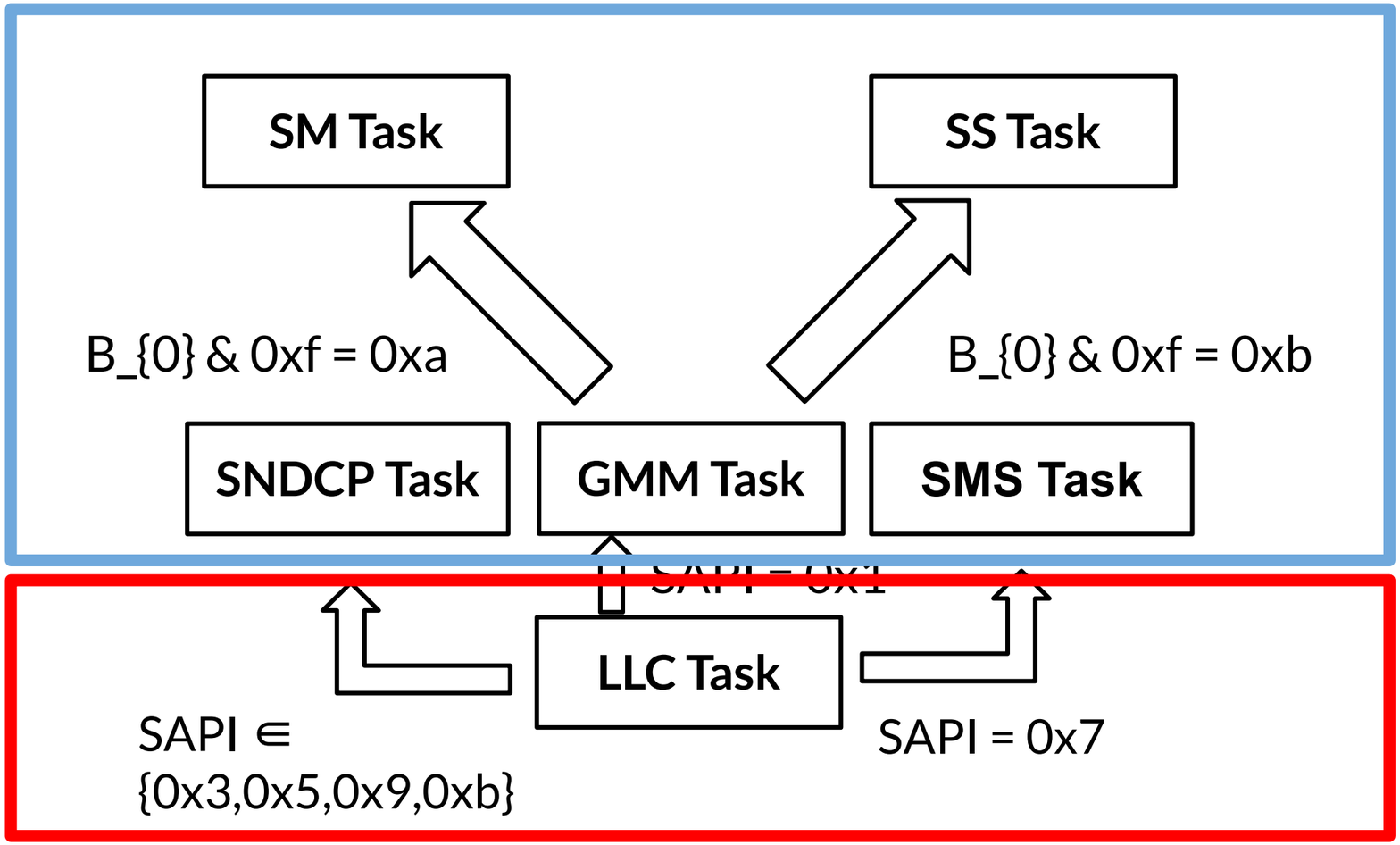
GPRS Layer 2

```
struct qitem_llc{
    struct qitem_header header;
    uint32_t tlli;
    struct qitem_llc_pdu *pdu;
    uint32_t length;
} PACKED;
```

```
struct qitem_llc_pdu{
    uint8_t address;
    uint16_t control;
    char information[N];
    uint8_t[3] CRC;
} PACKED;
```

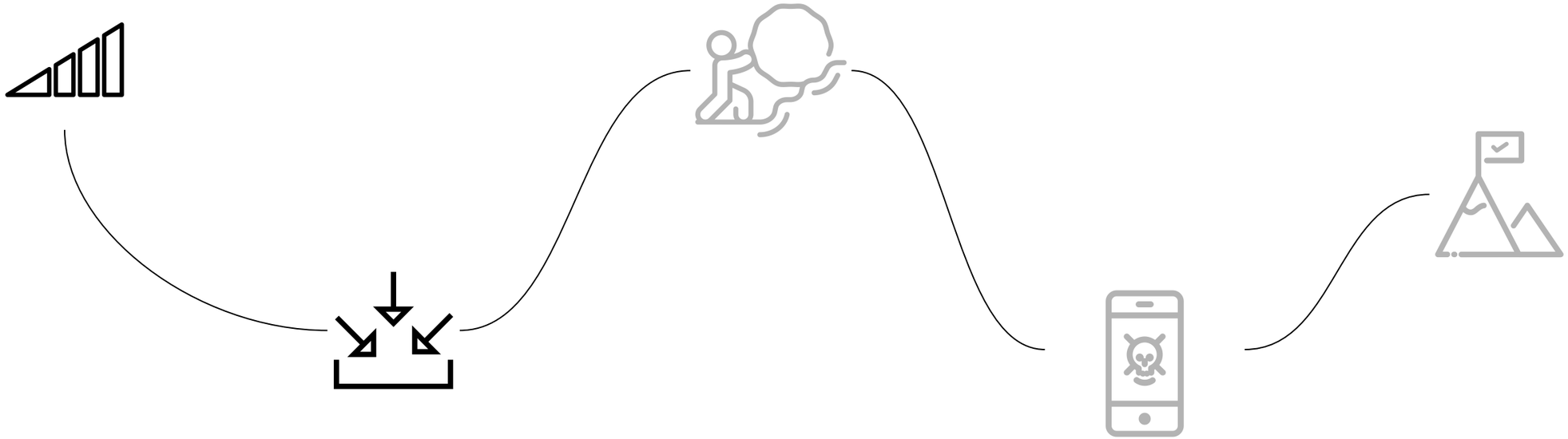
- **TLLI :** Temporary Logical Link Identifier
- **PDU:** Protocol Data Unit
  - **Information:** L3 data
  - **CRC:** 3 bytes Cyclic Redundancy Check (CRC)
  - **Address Field**
    - **SAPI :** Service Access Point Identifier

GPRS Layer 2/3



- **Information:**
  B_{0} … B_{N}

Our Approach to Fuzzing



Our Fuzzing Campaigns: FirmWire

- Full-system baseband emulator
  - Baseband emulation from boot
  - Fuzzing support via AFL++
  - Support for MTK & Exynos firmware

- Advantages:
  - Analyzable logs
  - Coverage tracking
  - Task-interaction

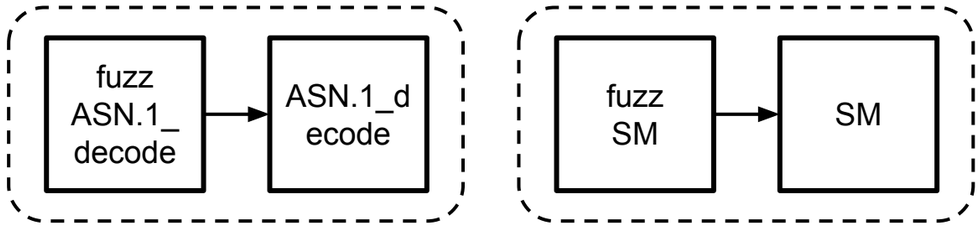
Fuzzing in FirmWire

- Requires injection of “Fuzzing Task”
  - Written in C, AFL wrapper present
  - Appears like an ordinary task for emulated CP
  - Sends messages to other tasks

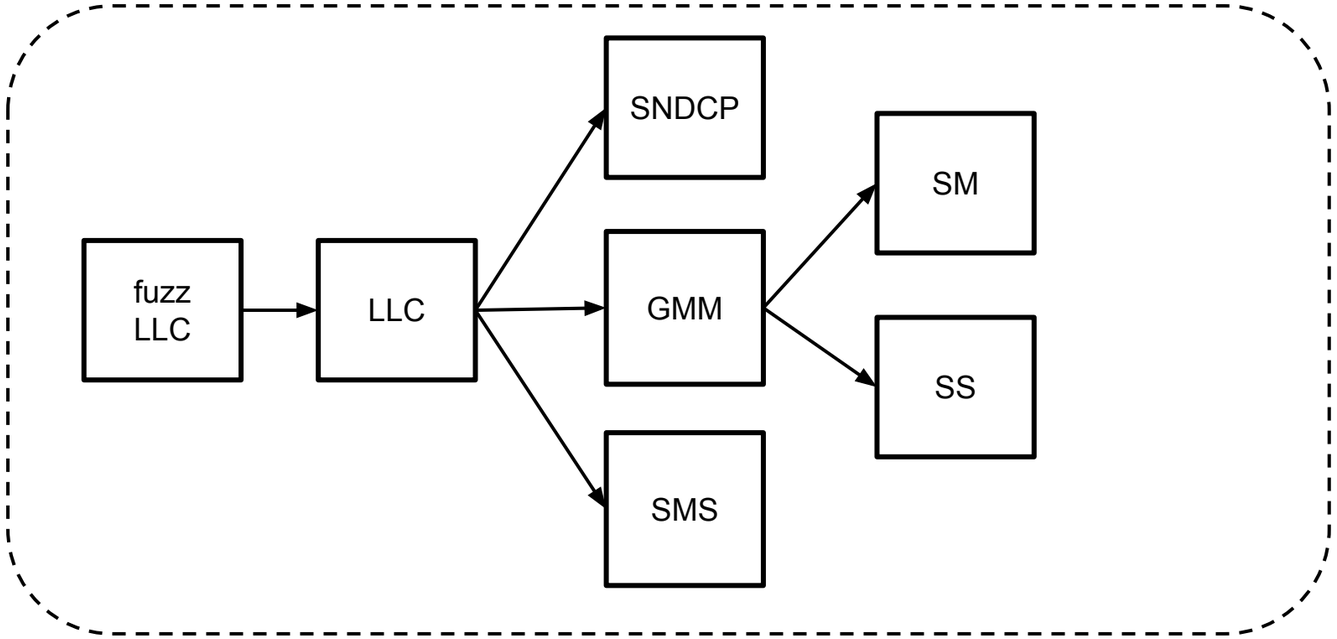
```
int fuzz_single_setup()
{
   qid = queuename2id("TARGET_TASK_QUEUE");
   struct qitem_target * init =
      pal_MemAlloc(4, sizeof(struct
      qitem_target), __FILE__, __LINE__);
   // setup init payload
   [...]
   pal_MsgSendTo(qid, init, 2);
   return 1;
}
```

The Plan: Fuzzing Layer-2

Existing Fuzzers (non-OTA)



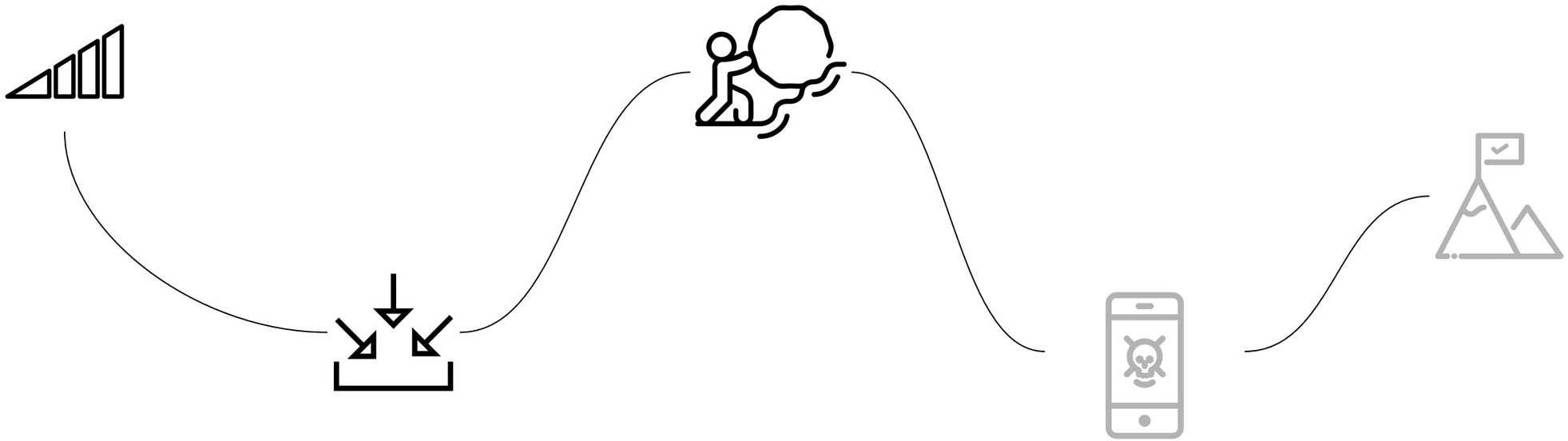
Our Approach (GPRS)



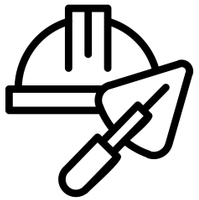
The Target: Galaxy S10e Firmware

- Latest phone model supported by FirmWire
- Released date: 2019
- Firmware date: March 2023
  - Original FirmWire bugs are patched

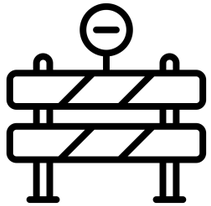
Challenges



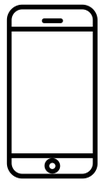
The Challenges



Need to create fuzzing task



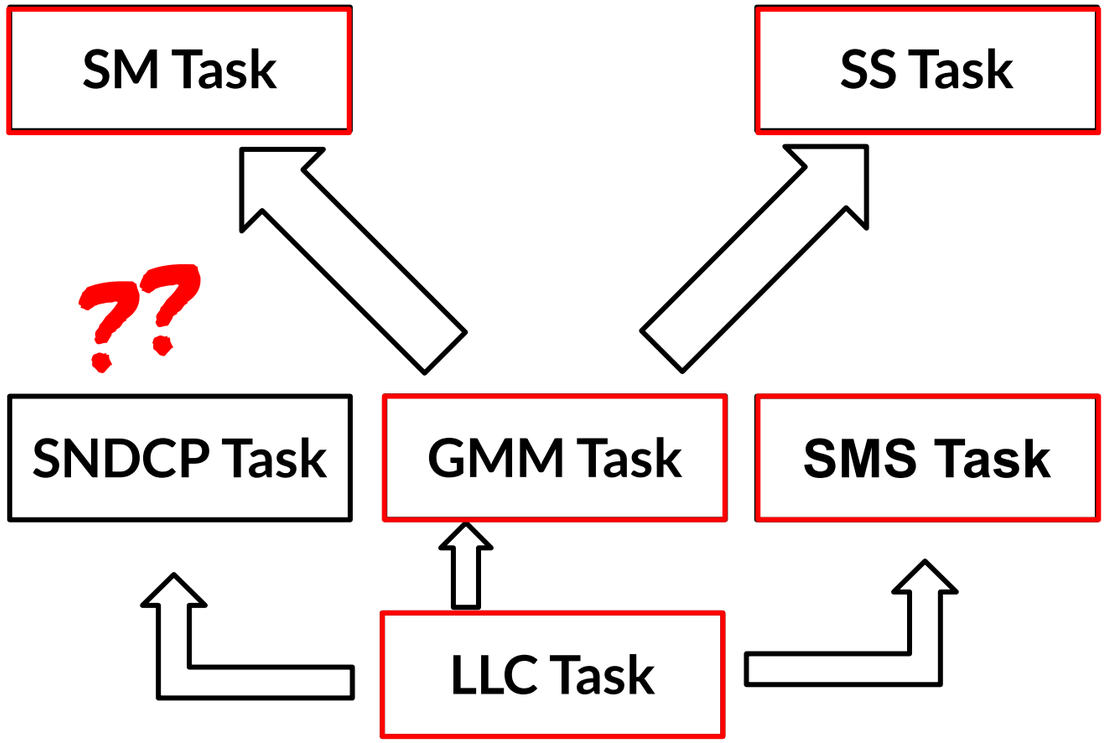
How to deal with complex baseband state



No support for recent phones

Challenge 1A: Creating Fuzzing Task

```
//Initialize LLC task

//Initialize SMS task

//Initialize GMM task

//Initialize SS task

//Initialize SM task
struct qitem_sm * init = pal_MemAlloc(4, sizeof(struct
qitem_sm), __FILE__, __LINE__);
init->header.op = 0;
init->header.size = 1;
init->header.msgGroup = 0x3407;

pal_MsgSendTo(queuename2id("SM"), init, 2);
```



Challenge 1B: Creating Fuzzing Task ++

- Some ta
- P

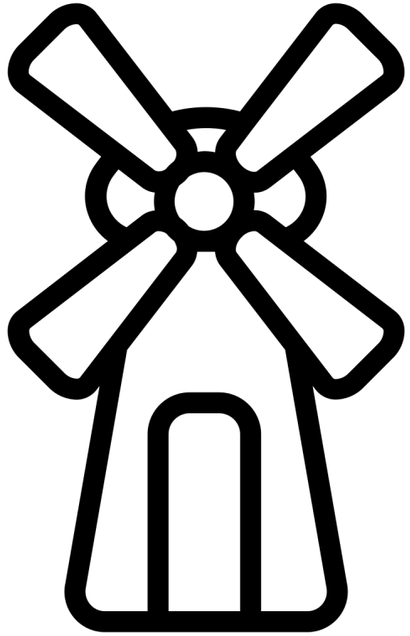
```
// Setup

..
send_sns
send_fir

..
```

```
int
data
  <
  *
  *
  .
  u
  *
  .
  i
  i
  i
  pa
}
```

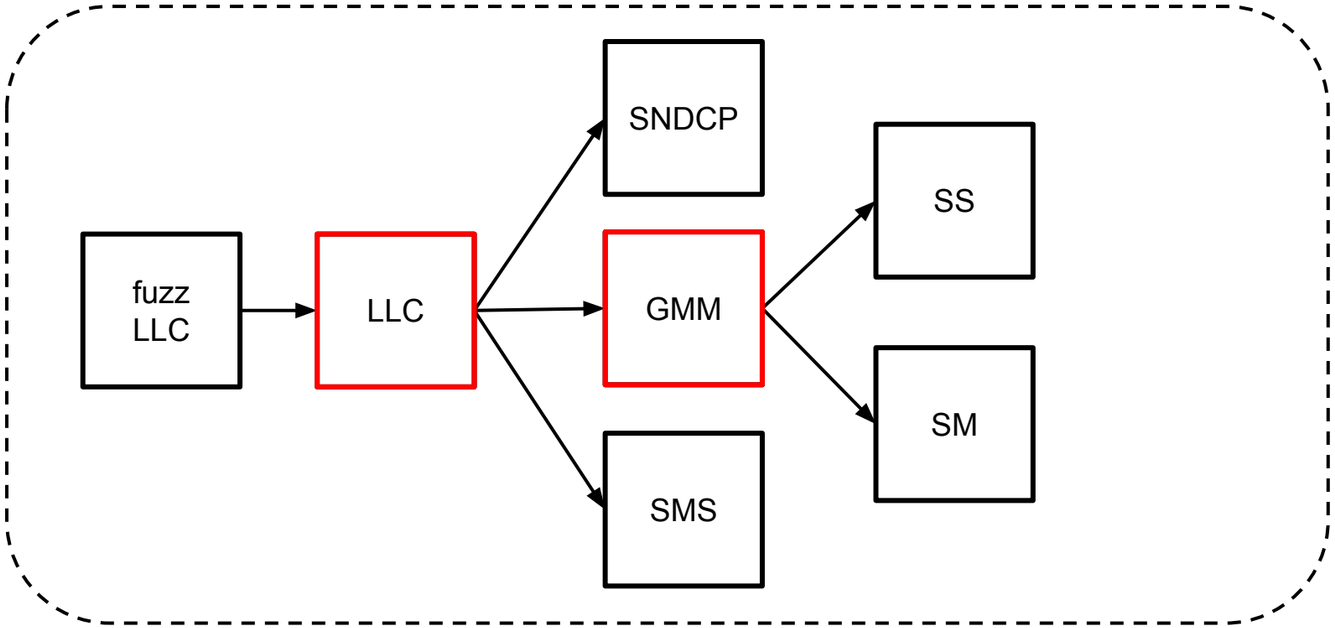
```
int send_first_xid_block(uint8_t isHdr_isData_XID, uint8_t N, uint8_t num_of_xid_params){
  <..> alloc + set item
  uint8_t xid_sapi = 0x3;
  xid->payload[0] = xid_sapi;
  uint8_t *xid_block = pal_MemAlloc(4, 11, __FILE__, __LINE__);
  *(uint32_t*)(xid->payload + 0x8) = xid_block;
  *(uint32_t*)(xid->payload + 0xc) = num_of_xid_params;
  uint8_t algorithm_type = 0x0;
  uint8_t NSAPIs = nsapi;
  uint8_t entity_num = 0x0;
  uint8_t DPCOMP1 = 0x1;
  uint8_t DPCOMP2 = 0x0;
  construct_xid_block(xid_block, algorithm_type, NSAPIs, N - 3, isHdr_isData_XID,
entity_num, DPCOMP1, DPCOMP2);
  pal_MsgSendTo(queuename2id("SNDCP"), xid, 2);
}
```

- All tasks are initialized, we should now be able to start fuzzing ….

let's take the fuzzer for a spin



```
[1.56233][MM] 0x40e6a275 0b101: [../../../HEDGE/NASL3/MM/Code/Src/mm_Main.c] - GS30: ucTransactionID = 0xFF,  ucProtocol
=NON_CALL_RELATED_SS_MESSAGES_PD
[1.56330][MM] 0x40e6a4cf 0b101: [../../../HEDGE/NASL3/MM/Code/Src/mm_Main.c] - GS30: Raw data length =5
[1.56482][MM] 0x40a6bcc5 0b101: [../../../HEDGE/NASL3/MM/Code/Src/mm_LlcManagement.c] - mm_DecodeLlGmmUnitDataIndMsg:
[1.56524][MM] 0x40e69bdd 0b10: [../../../HEDGE/NASL3/MM/Code/Src/mm_Main.c] - mm_GmmState -> GMM_NULL   GmmFunctionalState =
GMM_IDLE     GmmServiceState = GMM_NO_IMSI
[1.56584][MM] 0x40e6e88d 0b101: [../../../HEDGE/NASL3/MM/Code/Src/mm_Main.c] - Protocol Discriminator ->
NON_CALL_RELATED_SS_MESSAGES_PD
..
[1.57438][MM] 0x40a6be17 0b1: [../../../HEDGE/NASL3/MM/Code/Src/mm_LlcManagement.c] - Invalid mm_GmmState in
mm_DecodeLlGmmUnitDataIndMsg
```



Challenge 2: Initialize the State

```
switch(gmm_State) {
default:
  if (DAT_4182f500 < 2) {
    local_28 = &TE::mm_LlcManagement::Invalid_mm
    local_24 = (uint)DAT_4182f500 * 0x40000 + 0x
    dbgTrace_print(&local_28,&DAT_fecdba98);
  }
  gmm_State = 0x6d3;
  break;
case 1:
case 2:
  if (DAT_4182f500 < 2) {
    local_24 = (uint)DAT_4182f500 * 0x40000 + 0x
    local_28 = &TE::mm_LlcManagement::No_action_
    dbgTrace_print(&local_28,&DAT_fecdba98);
  }
  gmm_State = 0x6cb;
  break;
case 3:
case 4:
case 6:
case 7:
```

Do interesting stuff

```
local_2c = param_2;
local_28 = param_3;
local_24 = param_4;
gmm_State = gmm_getState();
if (DAT_4182f500 < 2) {
  local_28 = &TE::mm_LlcManagement::mm_DecodeLlGmmUnitDataIndMsg;
  loca
  dbgT
}
```

```
undefined gmm_getState(void)

{
  return (&DAT_42e22f60)[(uint)DAT_4182f500 * 10];
}
```

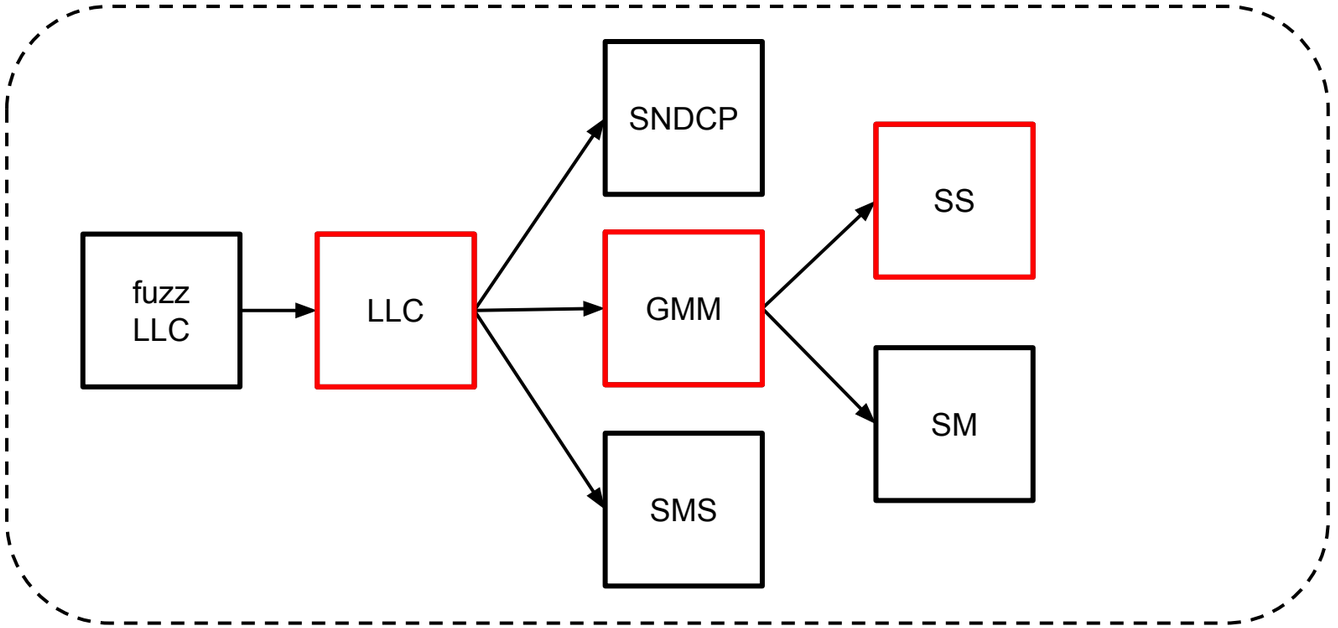
```
[1.57438][MM] 0x40a6be17 0b1: [../../../HEDGE/NASL3/MM/Code/Src/mm_LlcManagement.c] - Invalid mm_GmmState in
mm_DecodeLlGmmUnitDataIndMsg
```

Challenge 2: Initialize the state

```
#define NSAPI 0x25
#define nsapi (NSAPI & 0xf)
#define SAPI_OFFSET (nsapi * 0x63c)
#ifdef SAMSUNG_S10e
#define LLC_INIT_LEN 0x19
uint32_t gmm_state_ptr = 0x42e22f60;
uint32_t num_sndPDUs_addr =     0x42e0a88c + SAPI_OFFSET;
uint32_t sndp_TxMode_addr =     0x42e0a778 + SAPI_OFFSET;
uint32_t sndp_RxState_addr =    0x42e0a76d + SAPI_OFFSET;
uint32_t sndp_LlcSapi_addr =    0x42e0a87d + SAPI_OFFSET;
uint32_t snp_RxUnackSeqNumInq = 0x42e0a862 + SAPI_OFFSET;
uint32_t snpSetSate_check = 0x4182eff4;
uint32_t snp_trick_xid    = 0x42e163dc;
#endif
```

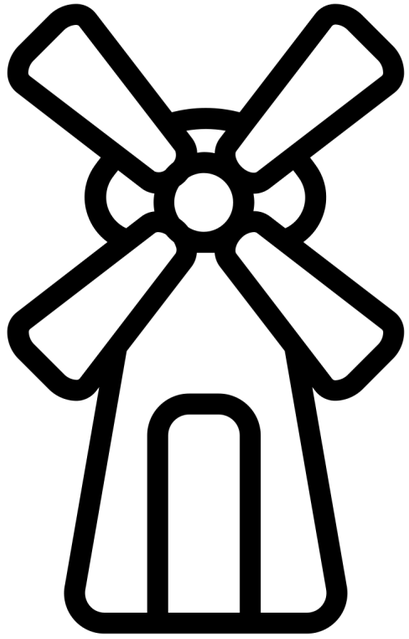
```
*(uint8_t*)gmm_state_ptr = 0x3;
*(uint32_t*)num_sndPDUs_addr = 0x0;
*(uint8_t*)sndp_TxMode_addr = 0x0;
*(uint8_t*)sndp_RxState_addr = 0x1;
*(uint16_t*)snp_RxUnackSeqNumInq = (uint16_t)0x0;
```

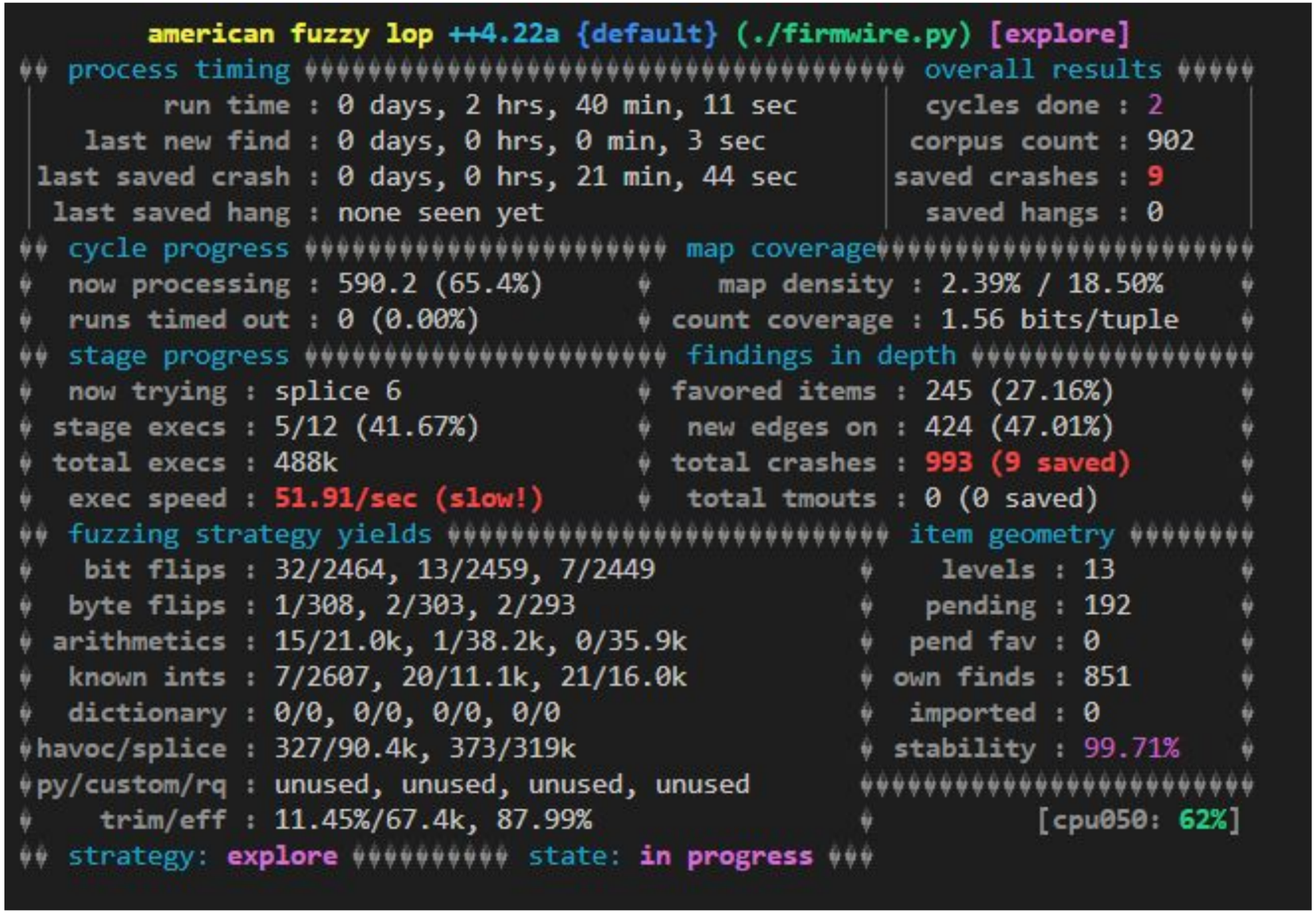
```
[1.22843][SS] 0x4106a97d 0b11: [../../../HEDGE/NASL3/SS/Code/Src/ss_Main.c] - ------------------------ SS TASK
----------------------------
[1.22937][SS] 0x41069ed3 0b10: [../../../HEDGE/NASL3/SS/Code/Src/ss_Main.c] - ss_UpdStackId :SsCurrentStackId: 0
[1.23043][SS] 0x4106a81b 0b100: [../../../HEDGE/NASL3/SS/Code/Src/ss_Main.c] - SS <== <RADIO MSG> REGISTER
[1.23081][SS] 0x4106a835 0b100: [../../../HEDGE/NASL3/SS/Code/Src/ss_Main.c] - Disp RX Msg Contents
[1.23272][SS] 0x416fc217 0b101: [../../../HEDGE/NASL3/SS/Code/Src/ss_PduCodec.c] - Displaying Information
Elements
[1.23371][SS] 0x416fc241 0b10: [../../../HEDGE/NASL3/SS/Code/Src/ss_PduCodec.c] - Received SS_REGISTER From
Network
```



All tasks (and state) are initialized, we should now be able to start fuzzing ….

let's take the fuzzer for a spin




american fuzzy lop ++4.22a {default} (./firmwire.py) [explore]
process timing
run time : 0 days, 2 hrs, 40 min, 11 sec
last new find : 0 days, 0 hrs, 0 min, 3 sec
last saved crash : 0 days, 0 hrs, 21 min, 44 sec
last saved hang : none seen yet
overall results
cycles done : 2
corpus count : 902
saved crashes : 9
saved hangs : 0
cycle progress
now processing : 590.2 (65.4%)
runs timed out : 0 (0.00%)
map coverage
map density : 2.39% / 18.50%
count coverage : 1.56 bits/tuple
stage progress
now trying : splice 6
stage execs : 5/12 (41.67%)
total execs : 488k
exec speed : 51.91/sec (slow!)
findings in depth
favored items : 245 (27.16%)
new edges on : 424 (47.01%)
total crashes : 993 (9 saved)
total tmouts : 0 (0 saved)
fuzzing strategy yields
bit flips : 32/2464, 13/2459, 7/2449
byte flips : 1/308, 2/303, 2/293
arithmetics : 15/21.0k, 1/38.2k, 0/35.9k
known ints : 7/2607, 20/11.1k, 21/16.0k
dictionary : 0/0, 0/0, 0/0, 0/0
havoc/splice : 327/90.4k, 373/319k
py/custom/rq : unused, unused, unused, unused
trim/eff : 11.45%/67.4k, 87.99%
item geometry
levels : 13
pending : 192
pend fav : 0
own finds : 851
imported : 0
stability : 99.71%
[cpu050: 62%]
strategy: explore
state: in progress

Success!

```
[8.67507][MM] 0x40e6a275 0b101: [../../../HEDGE/NASL3/MM/Code/Src/mm_Main.c] - GS30: ucTransactionID =
0x00,  ucProtocol =SM_MESSAGES_PD
[8.67526][MM] 0x40e6a4a5 0b101: [../../../HEDGE/NASL3/MM/Code/Src/mm_Main.c] - GS30: ucMessageType =0x42,
ucChannel =0x0B
[8.67544][MM] 0x40e6a4cf 0b101: [../../../HEDGE/NASL3/MM/Code/Src/mm_Main.c] - GS30: Raw data length =64
[8.67703][MM] 0x40e447e3 0b10: [../../../HEDGE/NASL3/MM/Code/Src/mm_GmmPduCodec.c] -
mm_NsUpdateCommonSignalingInfo MsgType = 2 / PD = 10
[8.67730][MM] 0x40e44897 0b10: [../../../HEDGE/NASL3/MM/Code/Src/mm_GmmPduCodec.c] - PDP ACT ACCEPT from NW
253 AddrInx =2
[8.67750][MM] 0x40e448df 0b10: [../../../HEDGE/NASL3/MM/Code/Src/mm_GmmPduCodec.c] - Data 0xfd
[8.68137][MM] 0x40c84ff7 0b10: [../../../PSS/StackService/CNS/DbgSap/Code/Src/ns_ServiceHandlerDmCommon.c]
- ns_UpdateCommonSignalingInfo length : 64
[8.68157][MM] 0x40c85069 0b10: [../../../PSS/StackService/CNS/DbgSap/Code/Src/ns_ServiceHandlerDmCommon.c]
- ns_UpdateCommonSignalingInfo direction = 6 / P_E: 0
[8.68177][MM] 0x40c85161 0b10: [../../../PSS/StackService/CNS/DbgSap/Code/Src/ns_ServiceHandlerDmCommon.c]
- ignore BT MSG at privacyEnable is false
[ERROR] firmwire.vendor.shannon.hooks: FATAL ERROR (MM): from 0x40589141 [pal_PlatformMisc.c:146 - Fatal
error: PAL_MEM_GUARD_CORRUPTION
../../../VARIANT/PALVar/Platform_EV/PAL/MemoryInterface/src/pal_MemInterface.c]
```

Challenge 3: No support for recent phones

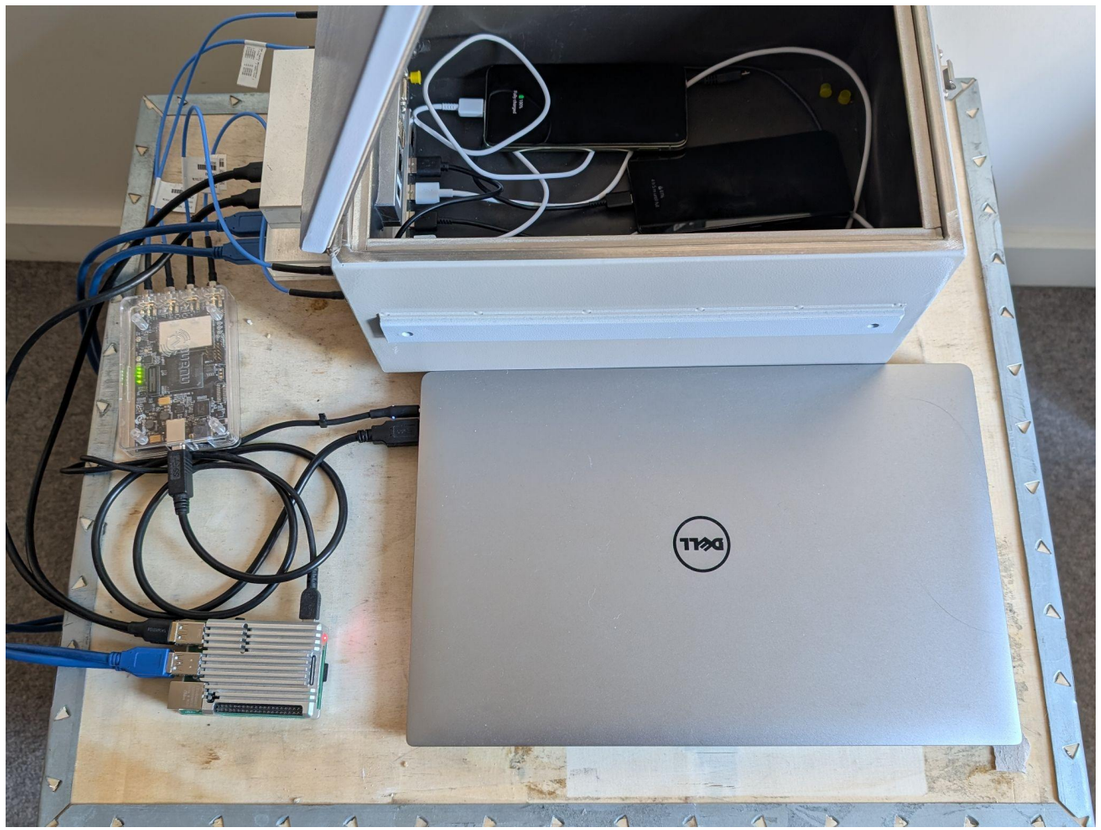
- Our fuzzing targeted a firmware from early 2023
- Confirm vulnerabilities OTA against newer devices
- Collect a bunch of crashing payloads
- Patch open source tooling to allow for automated testing

OTA Setup

- Hardware
  - SDR (BladeRF 2.0 micro xA4)
  - USB hub + cables
  - Laptop
  - Raspberry Pi 4
  - Faraday Cage

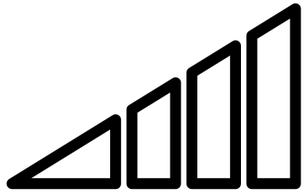
- Software:
  - Open source GPRS Base Station software: Yate v6.2.1 / YateBTS v6.1.1

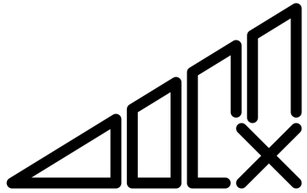
- Tested Phone:
  - Google Pixel 6 and 8
  - Samsung Galaxy S10e, S22, A14



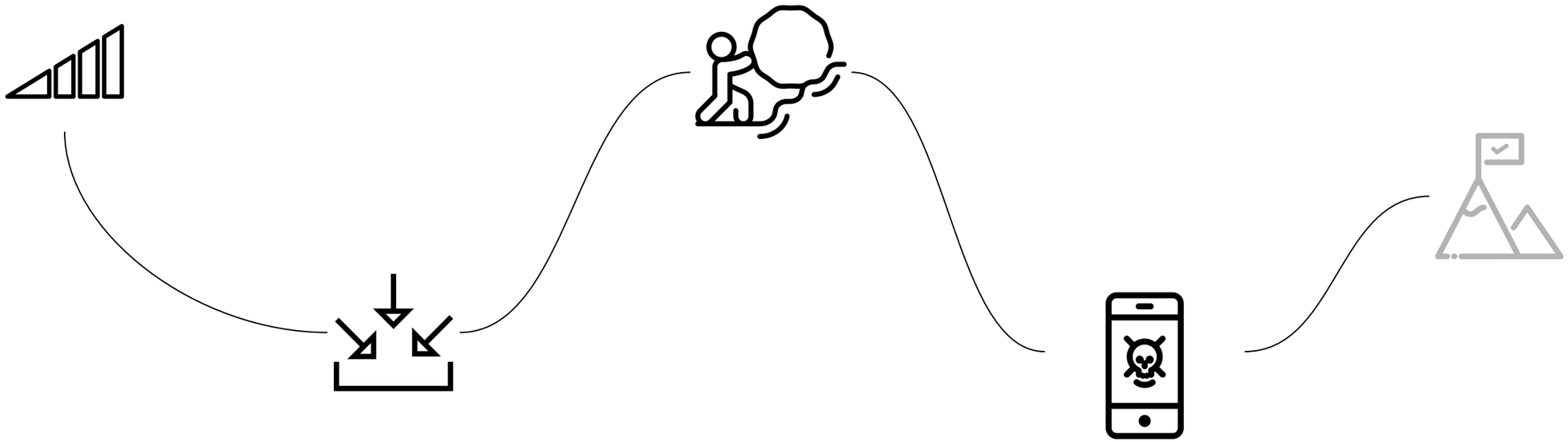
Replaying Crashes Over the Air

```
04-06 16:24:27.678  1063  1165 D DMD     : ModemStateMonitor : Modem  CRASH!!![2]
04-06 16:24:27.678  1063  1165 D DMD     : ModemStateMonitor : Check the state again after
2 seconds later.
04-06 16:24:29.669  1078  1152 D RFSD    : [ModemStateMonitor::OnModemCrashOrReset] Modem
is STATE_CRASH_EXIT or STATE_CRASH_RESET
04-06 16:24:29.679  1063  1165 D DMD     : ModemStateMonitor : Modem  CRASH!!![2]
04-06 16:24:29.679  1063  1165 D DMD     : ModemStateMonitor : Check the state again after
2 seconds later.
04-06 16:24:31.669  1078  1152 D RFSD    : [ModemStateMonitor::OnModemCrashOrReset] Modem
is STATE_CRASH_EXIT or STATE_CRASH_RESET
04-06 16:24:31.680  1063  1165 D DMD     : ModemStateMonitor : Modem  CRASH!!![2]
04-06 16:24:31.681  1063  1165 D DMD     : ModemStateMonitor : Check the state again after
2 seconds later.
04-06 16:24:33.670  1078  1152 D RFSD    : [ModemStateMonitor::OnModemCrashOrReset] Modem
is STATE_CRASH_EXIT or STATE_CRASH_RESET
```





A look at vulnerabilities



CVE 2024-47012(GMM)

- SM Activate PDP context accept
- N : Input payload of 192 bytes

0a 42 fd fd 41 41 41 41 41 41 41 41 ..
41 41 41 41 41 41 41 41 41 41 41 41 ..

IE	Presence	Value	Length
PD	M		1
MsgType	M		1
Negotiated LLC SAPI	M		1
Negotiated QoS	M		13-21
Radio Priority	M		1
PDP Address	O	0x2b	4-24
Protocol Configuration Options	O	0x27	3-253
Packet Flow Identifier	O	0x34	3
SM Cause	O	0x39	3
Connectivity Type	O	0xb-	1
WLAN Offload Indication	O	0xc-	1
NBIFOM Container	O	0x33	3-257
Extended Protocol Configuration Options	O	0x7b	4-65538
Extended QoS	O	0x5c	12

table source: https://www.etsi.org/deliver/etsi_ts/124000_124099/124008/18.07.00_60/ts_124008v180700p.pdf

CVE 2024-47012(GMM)

- Negotiated Quality of Service (QoS)

```
PDPaddrIdx = (uint)(byte)(payload[3] + 5);
        // 0xFD + 5 -> 0x2
```

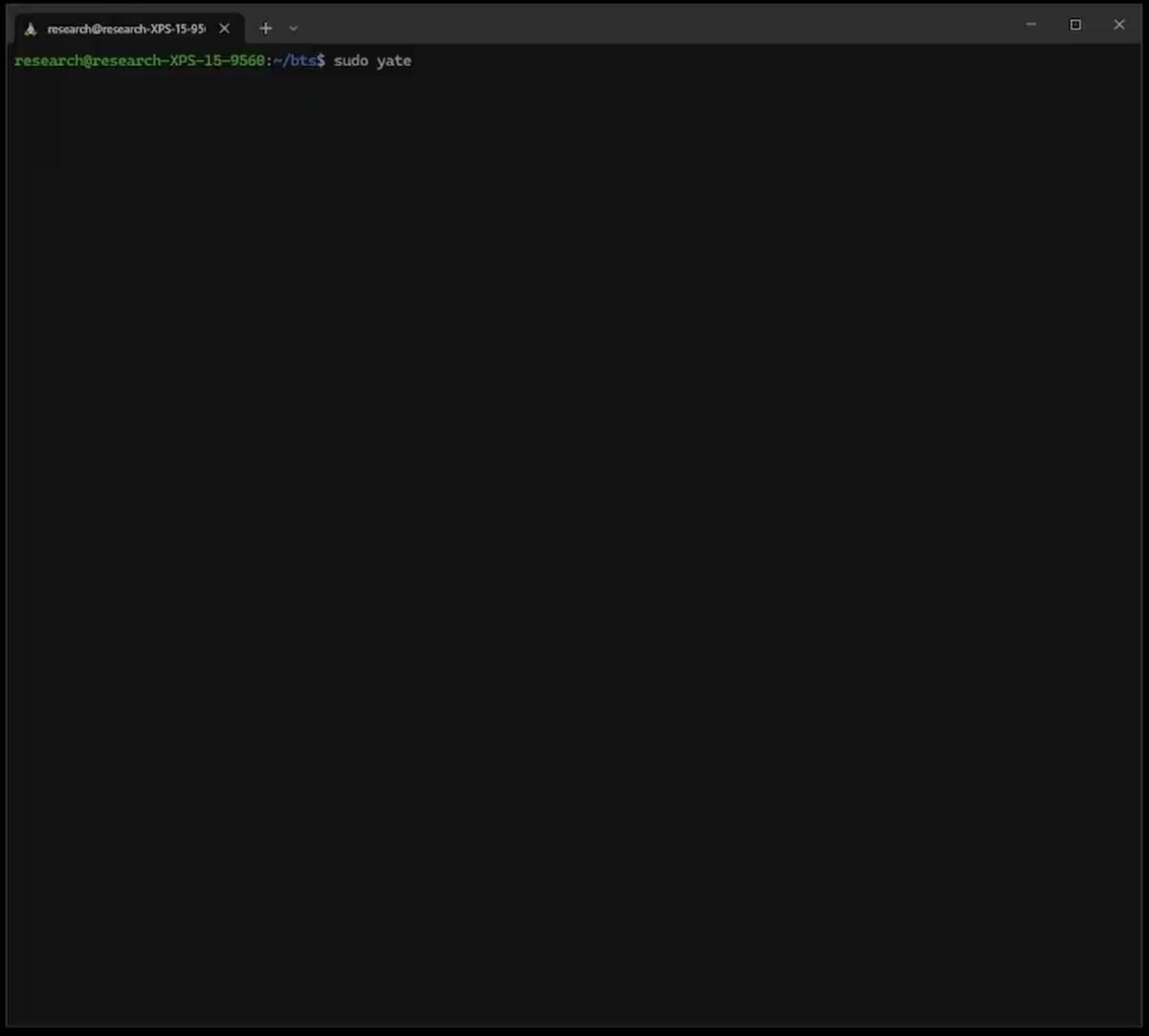
```
PDPaddrIE= (uint)(byte)payload[PDPaddrIdx];
        // uVar3 = payload[2] = 0xFD
```

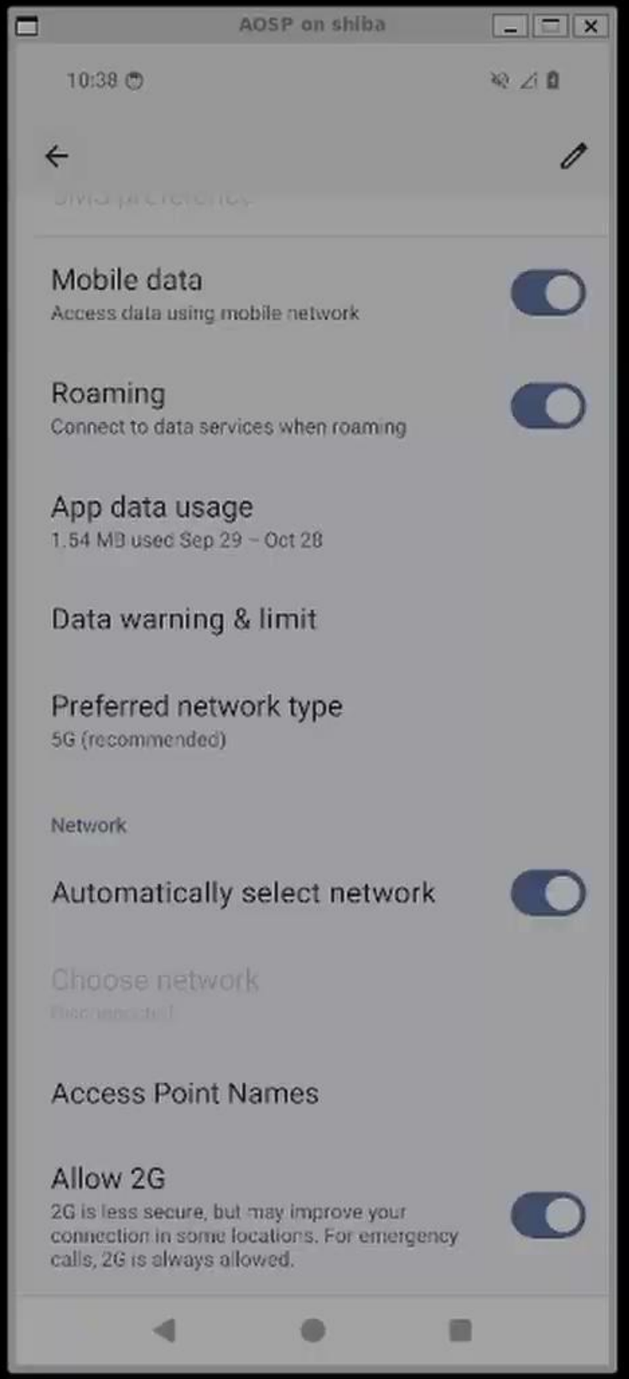
0a 42 fd fd 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41 41 41

IE	Presence	Value	Length
PD	M		1
MsgType	M		1
Negotiated LLC SAPI	M		1
Negotiated QoS	M		13-21
Radio Priority	M		1
PDP Address	O	0x2b	4-24
			3-253
			3
			3
			1
			1
			3-257
			4-65538
			12

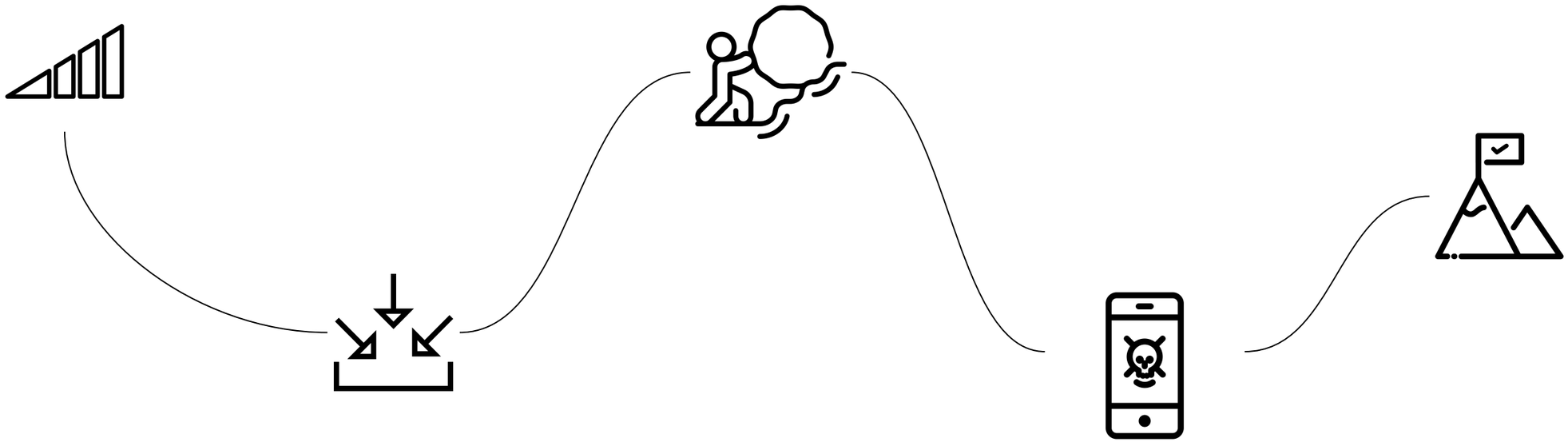
```
if(PDPaddrIdx < N){
  if(PDPaddrIE == 0x2b){
     //PDP Address Field present
      ..
  }
  else if(PDPaddrIdx == 2 && payload[2] > 2){
    local_40 = malloc(N);
    memset(local_40+5,0,PDPaddrIE);
   // OVERFLOW! 192 < 253 + 5
  }
  ..
}
```

table source: https://www.etsi.org/deliver/etsi_ts/124000_124099/124008/18.07.00_60/ts_124008v180700p.pdf


research@research-XPS-15-95
research@research-XPS-15-9560:~/bts$ sudo yate


AOSP on shiba
10:38
Mobile data
Access data using mobile network
Roaming
Connect to data services when roaming
App data usage
1.54 MB used Sep 29 – Oct 28
Data warning & limit
Preferred network type
5G (recommended)
Network
Automatically select network
Choose network
Access Point Names
Allow 2G
2G is less secure, but may improve your connection in some locations. For emergency calls, 2G is always allowed.

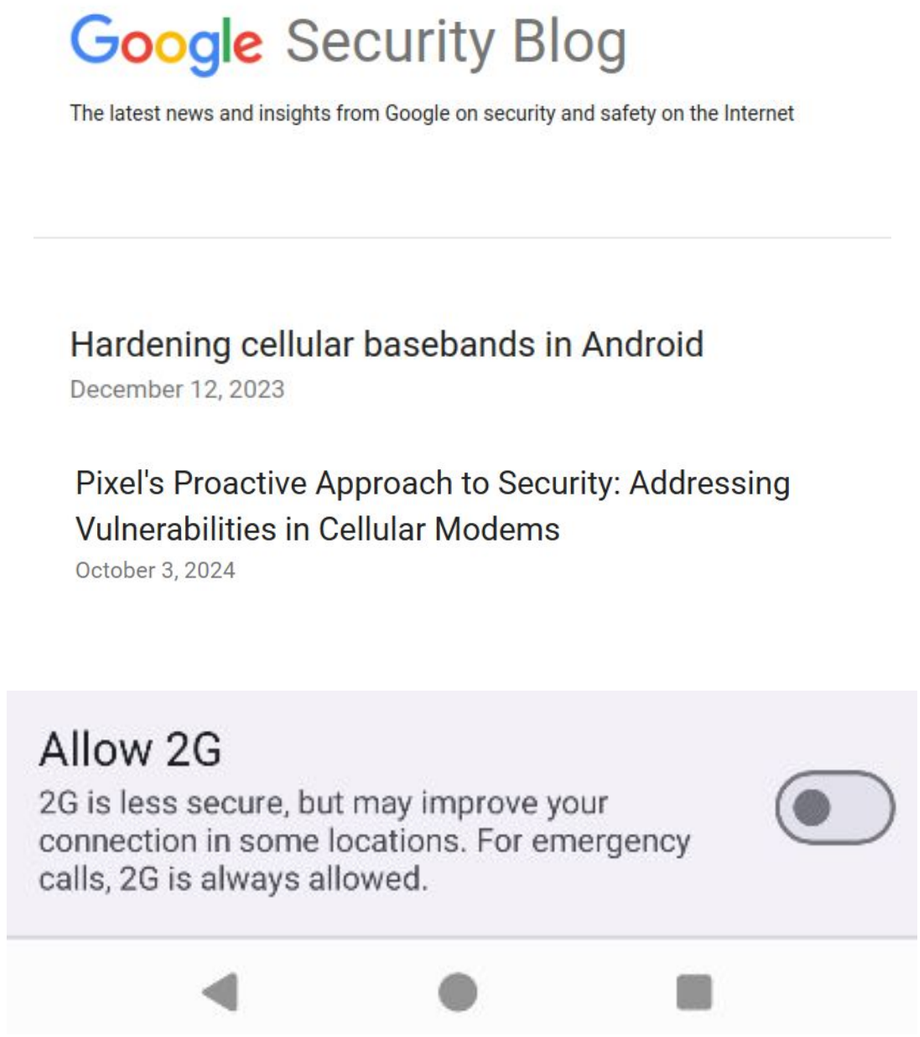
Wrapping Up



Defenses

- Recent shift in vendor's approaches:
  - More hardening for basebands (good)!

- Recently introduced defenses:
  - Integer Overflow Sanitizer / CFI / Auto-Initialize Stack Variables (Pixel 9)
  - Heap Sanitization (Pixel 8)
  - Stack Canaries
  - More consistent use of XN
  - Allow 2G



Conclusion

- Basebands have been extensively analysed in the past
  - Yet, new approaches may still yield plenty undiscovered bugs
- Undiscovered bugs have a long lifetime in baseband firmware
  - And can have cross-vendor implications
- Vendors approaches to baseband security are changing
  - In a good way!

Questions

Or via email:
<dyonwg@gmail.com>
<m.muench@bham.ac.uk>