



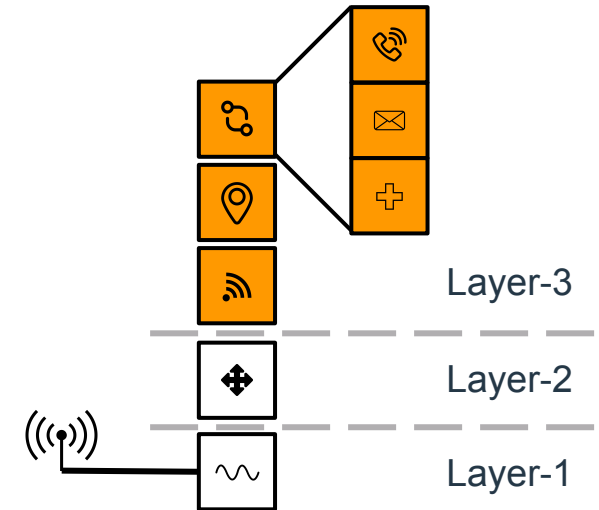
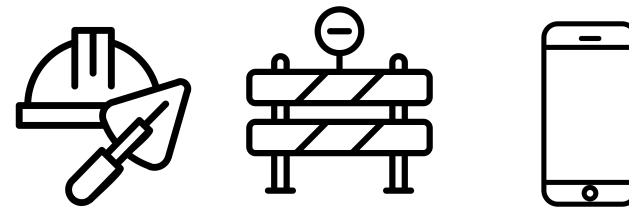
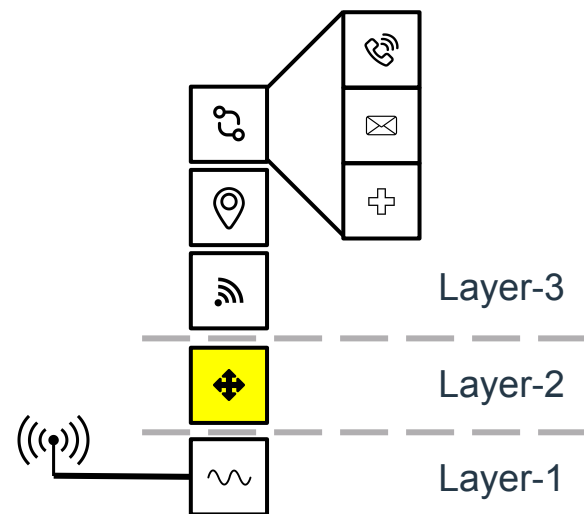
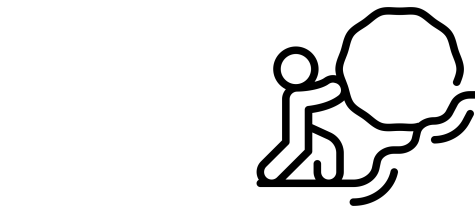
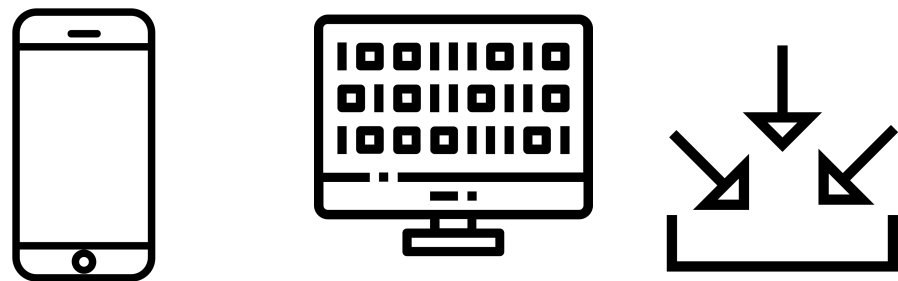
**black hat**<sup>®</sup>  
USA 2024

**AUGUST 7-8, 2024**  
BRIEFINGS

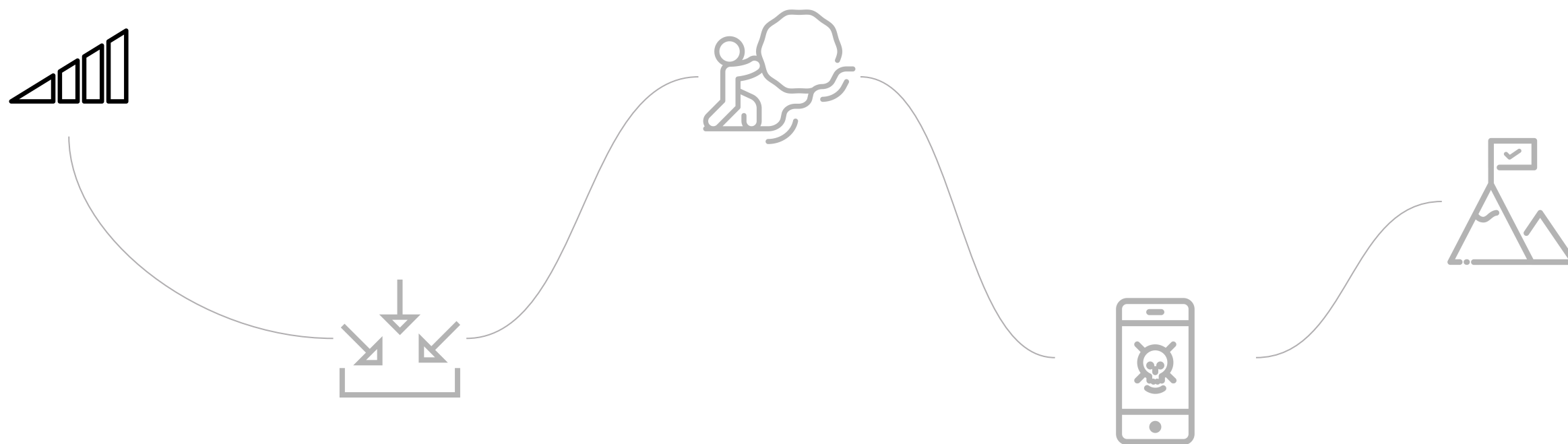
# **Overcoming State: Finding Baseband Vulnerabilities by Fuzzing Layer-2**

Speakers: Dyon Goos & Marius Muench

# This talk

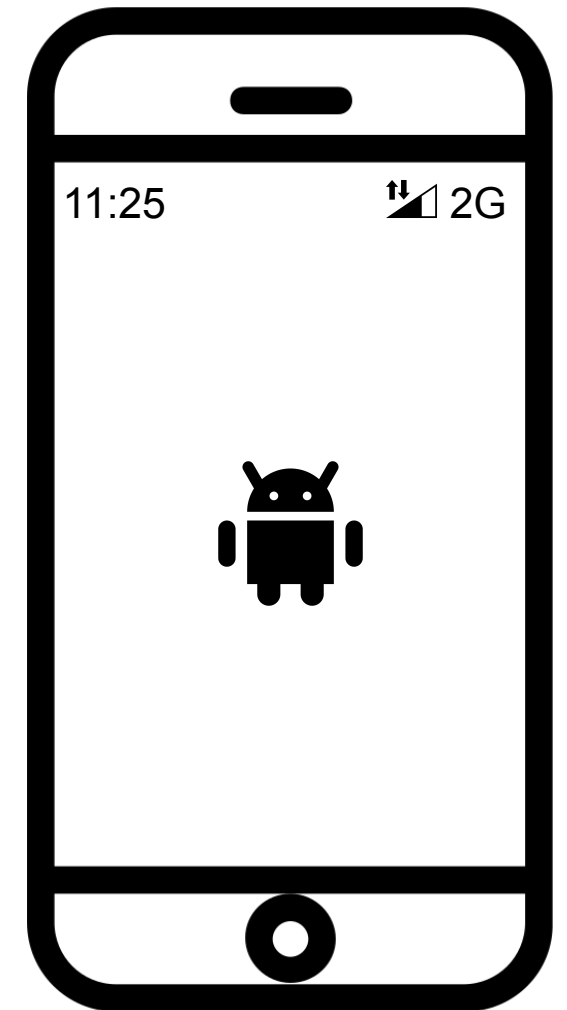


# Basebands



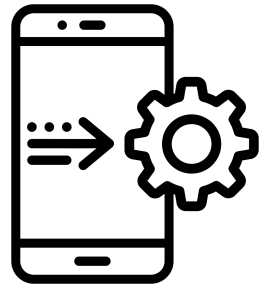
# Basebands

- 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.:

Basebanheimer  
Now I Am Become Death, The Destroyer of Chains

black hat USA 2023  
AUGUST 9-10, 2023  
BRIEFINGS

**Over the Air, Under the Radar**  
Attacking and Securing the Pixel Modem

Google

How to Hack Shannon Baseband  
(from a Phone)

White Paper | Attack Overview | Targets | Impact | Descriptions | 5G Tooling & PoC

**5Ghoul : Unleashing Chaos on 5G Edge Devices**

Matheus E. Garbelini<sup>1</sup>; Zewen Shang<sup>1</sup>; Shijie Luo<sup>1</sup>; Sudipta Chattopadhyay<sup>1</sup>; Sumei Sun<sup>2</sup>; Ernest Kurniawan<sup>2</sup>

<sup>1</sup>Singapore University of Technology and Design;  
<sup>2</sup>I2R, A\*STAR

**There will be Bugs: Exploiting Basebands in Radio Layer Two**

Daniel Komaromy

Baseband exploitation in public originally focused on message decoding bugs in layer 3 (NAS and RRC) and more recently in layer 4 (traffic over IP). In this presentation we uncover a new area of exploration for remote baseband exploitation in layer 2. In the past, this part of cellular specifications has been overlooked due to its function and packet size limitations. However, a deeper dive uncovers possibilities that show up in both old and new standards. Importantly, this is a layer that is below the ciphering applied to cellular communications providing an attack surface reachable not only with fake base stations but with direct MITM-ing of legitimate cell tower communications too. The presentation will describe the chain of vulnerabilities we have found and explain how to exploit them for remote code execution in the baseband of flagship Samsung smartphones. The new class of bugs meant new challenges both in developing and delivering an exploit. I will describe how we have modified radio software to inject a more complex sequence of malicious layer two traffic without the

Analyzing Cellular Basebands with

CCCamp '23  
2023-08-17

nsr & domenukk

**One SMS to Root Them All**

Alexander Kozlov @NOUm3n0n  
Sergey Anufrienko @madprogrammer

Kaspersky ICS CERT

**Cracking the 5G Fortress: Peering Into 5G's Vulnerability Abyss**

Kai Tu | Research Assistant, The Pennsylvania State University  
Yilu Dong | Research Assistant, The Pennsylvania State University  
Abdullah Al Ishtiaq | Research Assistant, The Pennsylvania State University  
Syed Md Mukit Rashid | Research Assistant, The Pennsylvania State University  
Weixuan Wang | Graduate Researcher, The Pennsylvania State University  
Tianwei Wu | Research Assistant, The Pennsylvania State University  
Syed Rafiul Hussain | Assistant Professor, The Pennsylvania State University

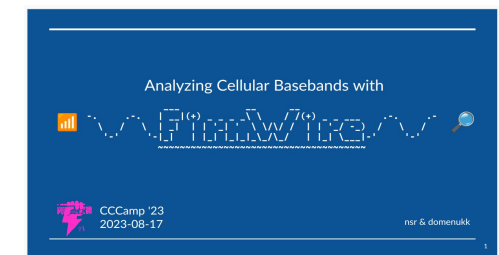
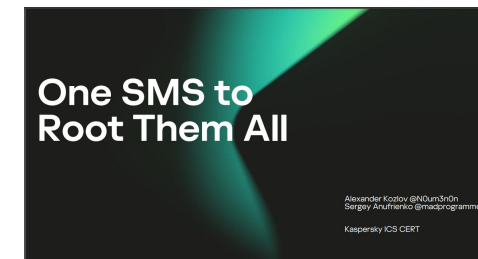
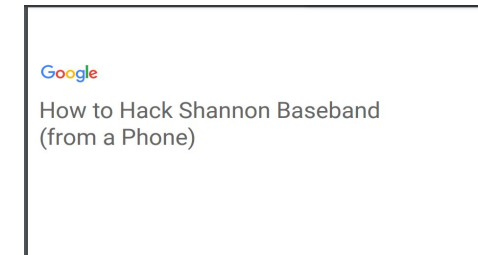
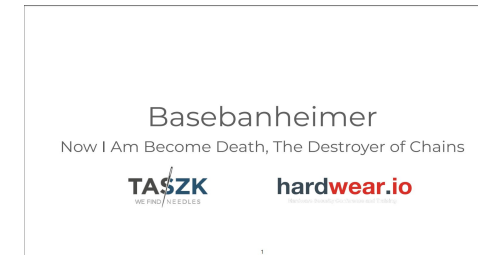
# What about Layer-2?

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

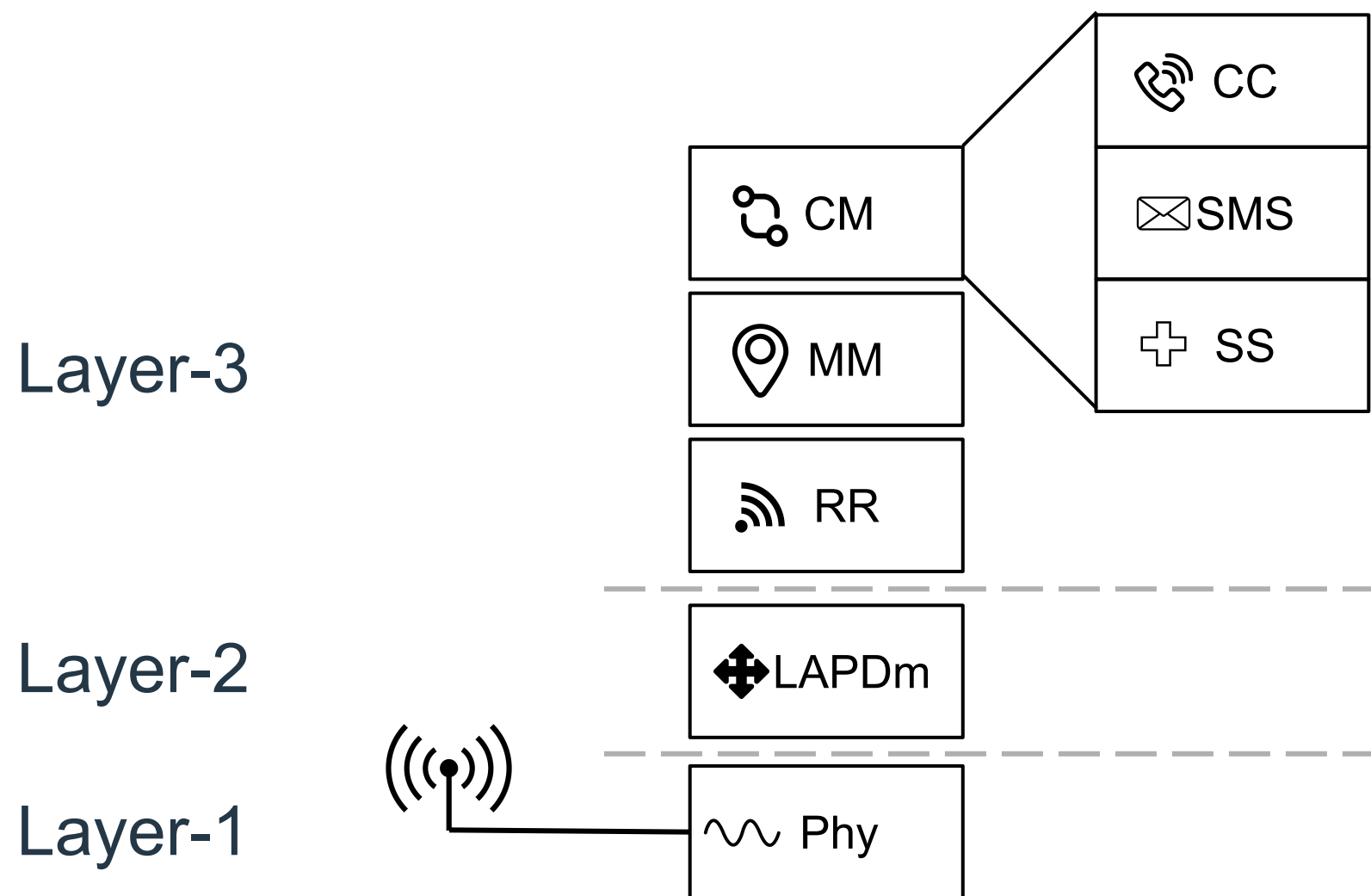
🤔 Let's have a look at layer-2 ourselves!

⇒ Let's start with the lowest hanging fruits:

- GSM Layer-2
- Fuzzing



# GSM Protocol Stack



- RR** : Radio Resource
- MM** : Mobility Management
- CM** : Connection Management
- CC** : Call Control
- SMS** : Short Messaging Service
- SS** : Supplementary Services

**LAPDm** : Link Access Protocol on the Dm Channel (LAPDm)

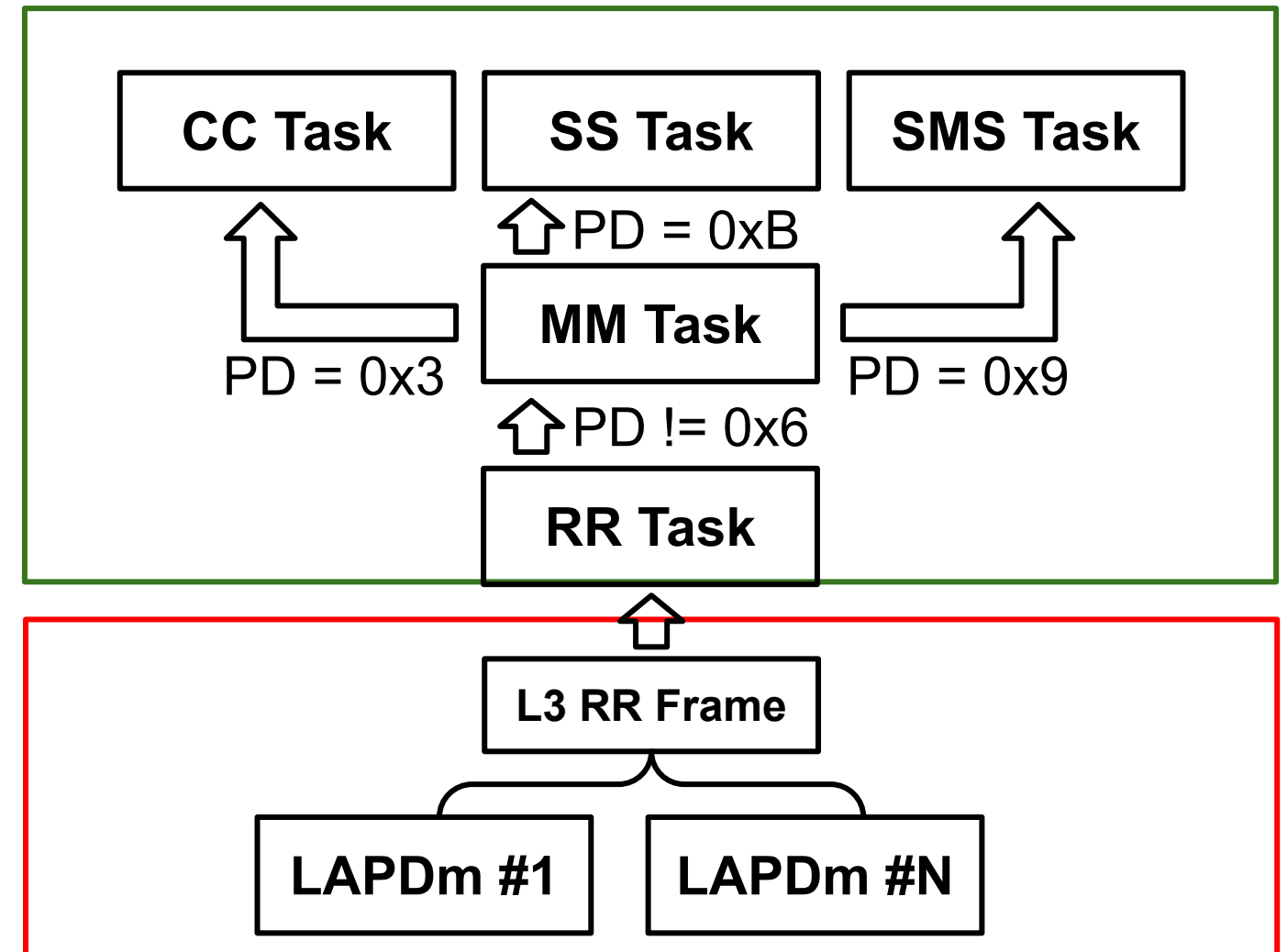
**Phy** : Physical



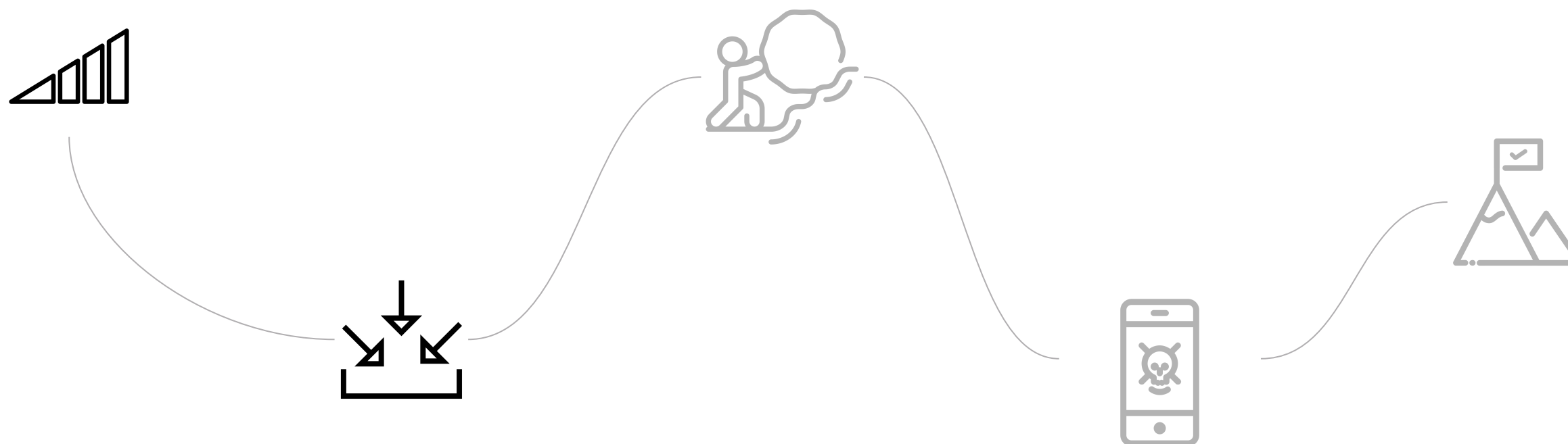
## GSM Layer 2

- Link Access Protocol on the Dm Channel (LAPDm)
- Frame Concatenation
- PD: *information[0]* & 0xF

```
struct LAPDM_frame{  
    uint8_t addr;  
    uint8_t ctrl;  
    uint8_t len;  
    uint8_t information[N];  
} PACKED;
```

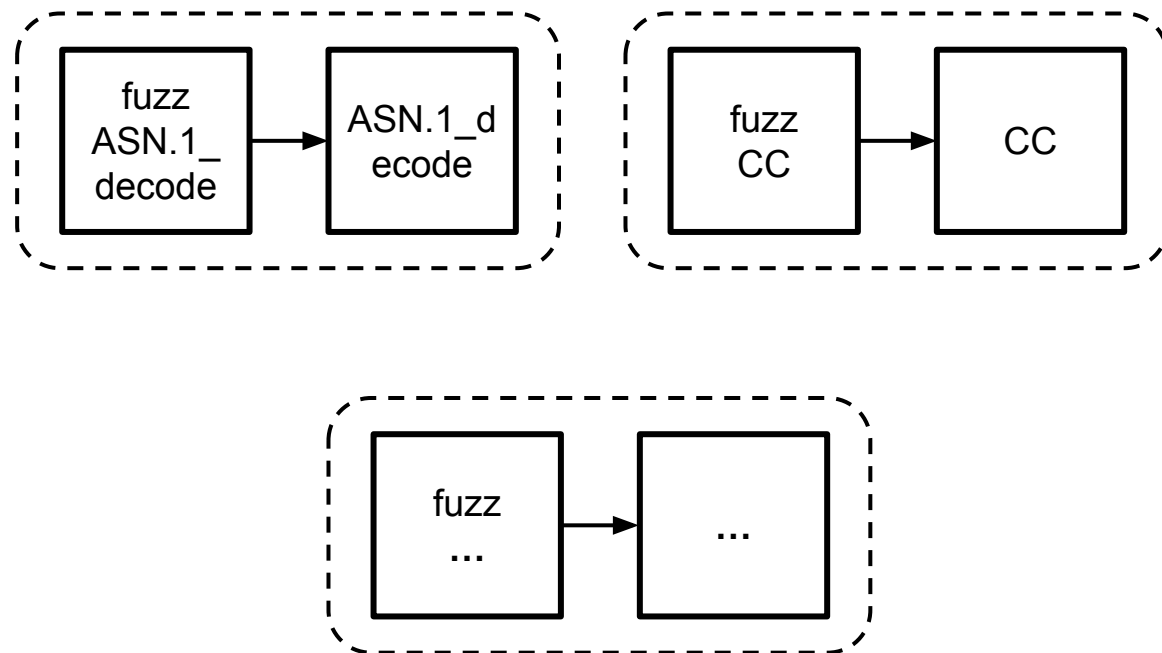


# Our approach to fuzzing

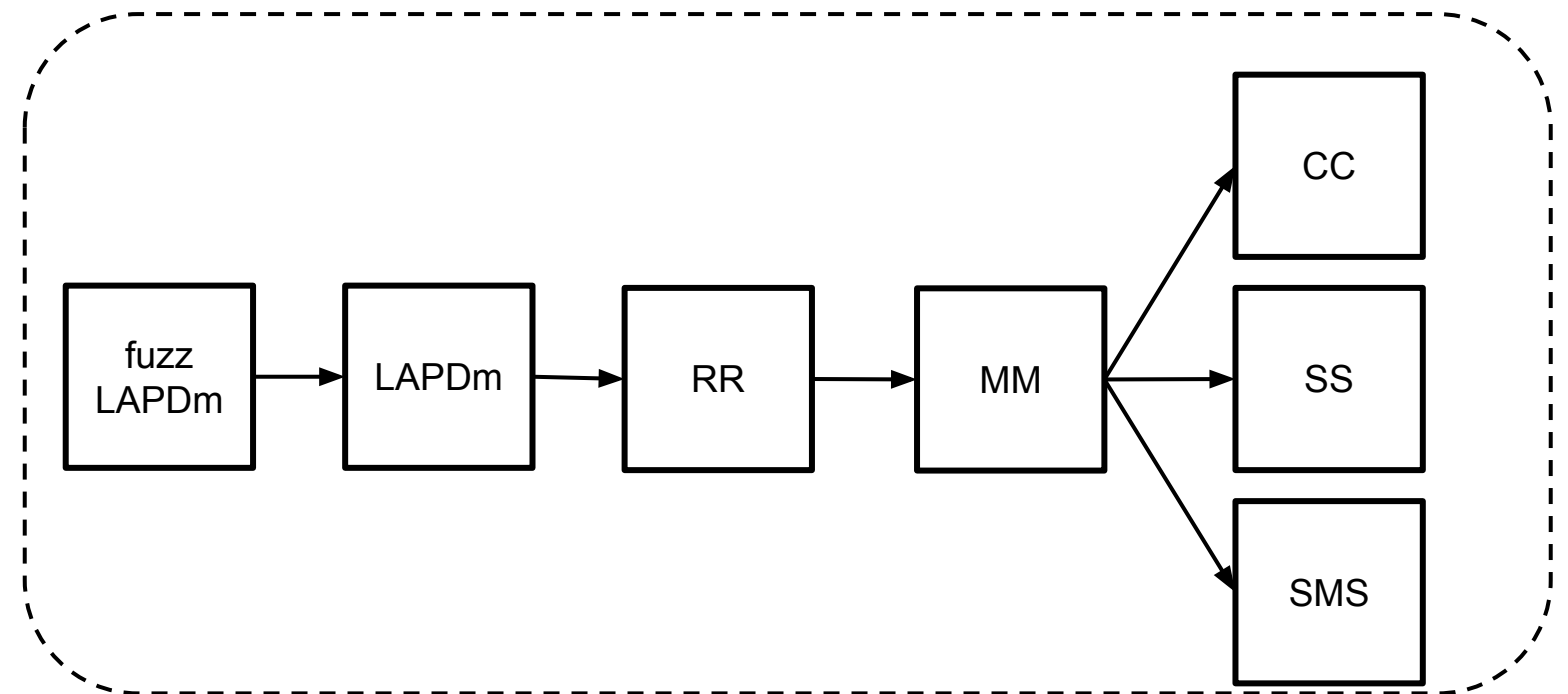


# The Plan: Fuzzing Layer-2

## Existing Fuzzers (non-OTA)

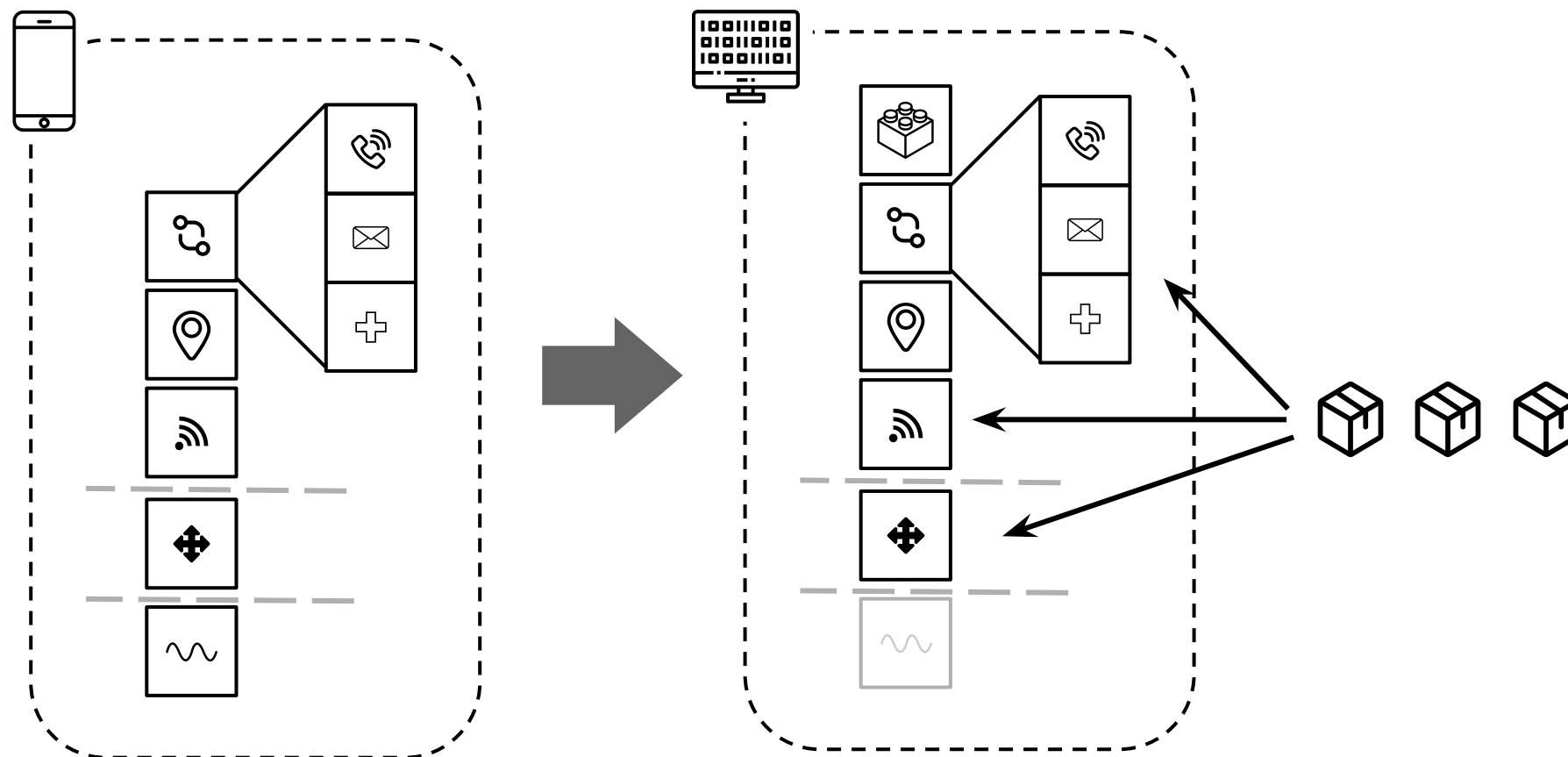


## Our Vision (GSM)



# Our Fuzzing Campaigns: FirmWire

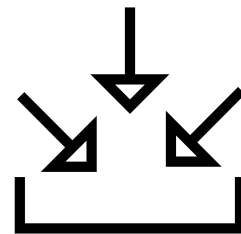
Full-system baseband emulator



- 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



```
void fuzz_single()
{
    [...]
    char * buf = getWork(&input_size);
    size = (uint16_t) input_size;
    //set item values
    [...]
    memcpy(item->payload, buf, size);

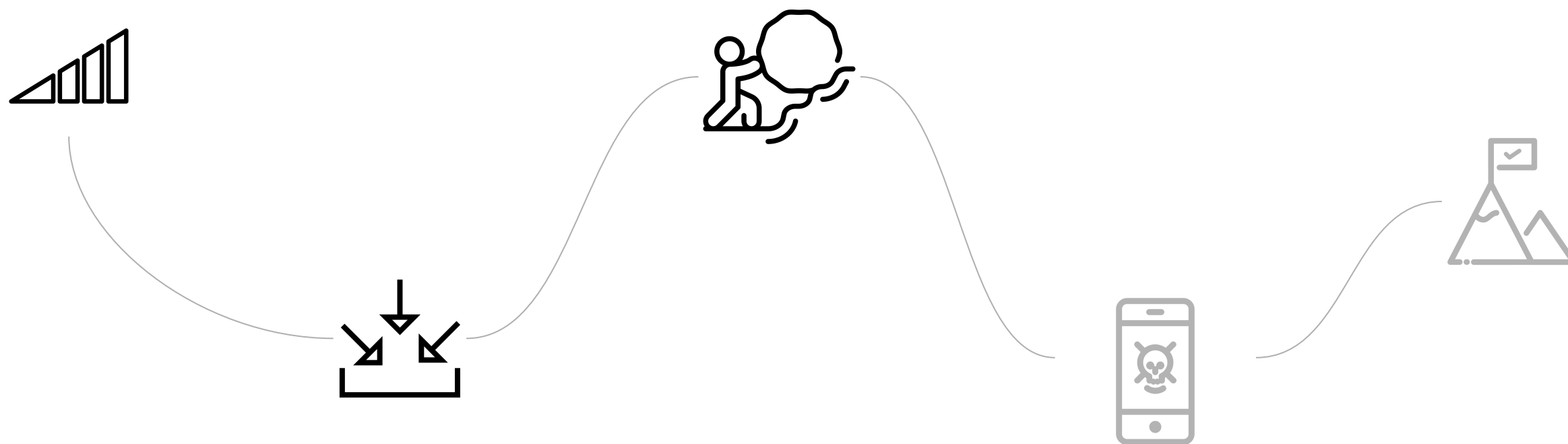
    startWork(0, 0xffffffff);
    pal_MsgSendTo(qid, item, 2);
}
```

# Target Firmwares (Exynos)

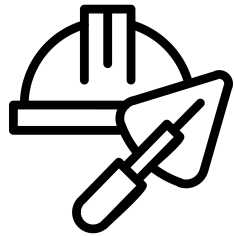
Phone model:	Motorola One Vision	Galaxy S10e
Phone release:	May 2019	March 2019
Firmware date:	July 2022	March 2023
Selection Reason:	Availability, familiarity	Latest model supported by FirmWire



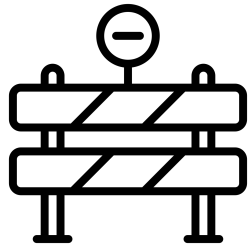
# Challenges



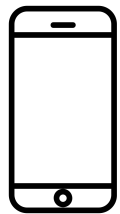
# The Challenges



Need to create fuzzing task



How to deal with complex baseband state

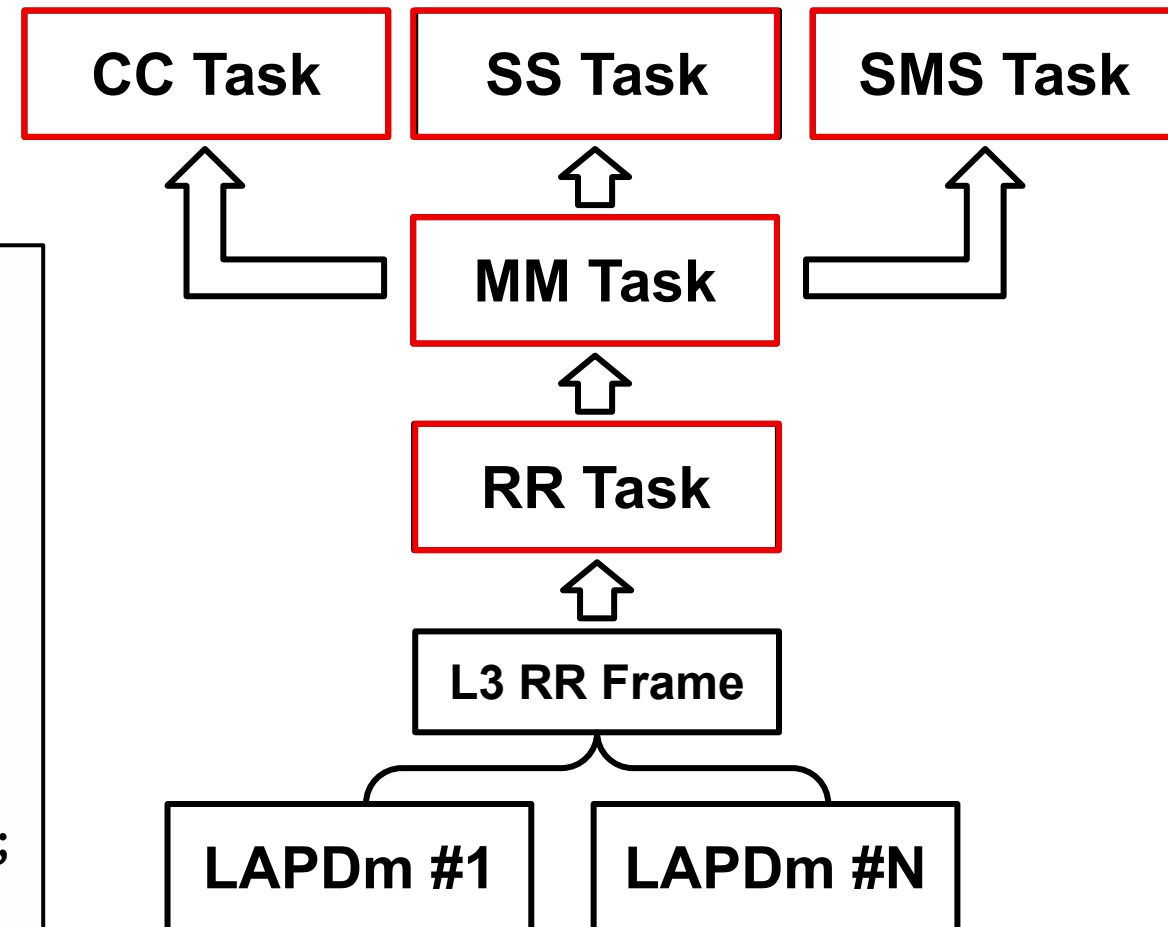


No support for recent phones



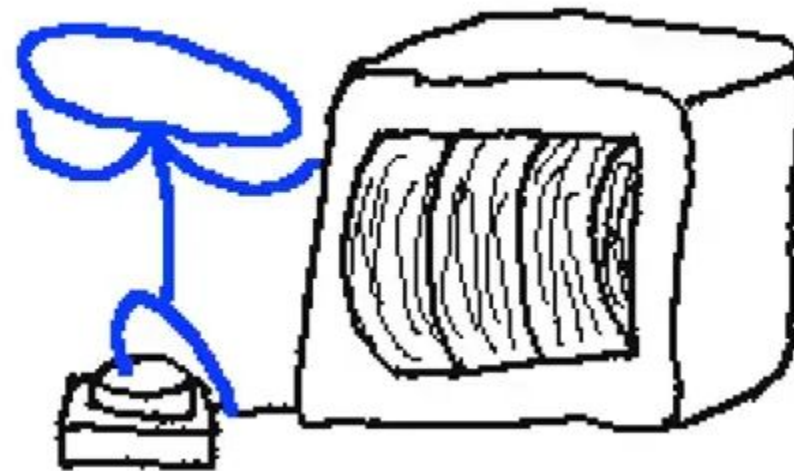
# Challenge 1: Creating fuzzing task

```
/Initialize RR task
s //Initialize MM task
s //Initialize CC task
st //Initialize SS task
st //Initialize SMS task
struct qitem_sms * init_sms =
    pal_MemAlloc(4, sizeof(struct qitem_sms),
    __FILE__, __LINE__);
init_sms->header.op = 0;
init_sms->header.op2 = 0x8a;
init_sms->header.size = sizeof(struct qitem_sms_init_req);
init_sms->header.msgGroup = 0x2d07;
pal_MsgSendTo(queueName2id("SMS"), init_sms, 2);
```



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

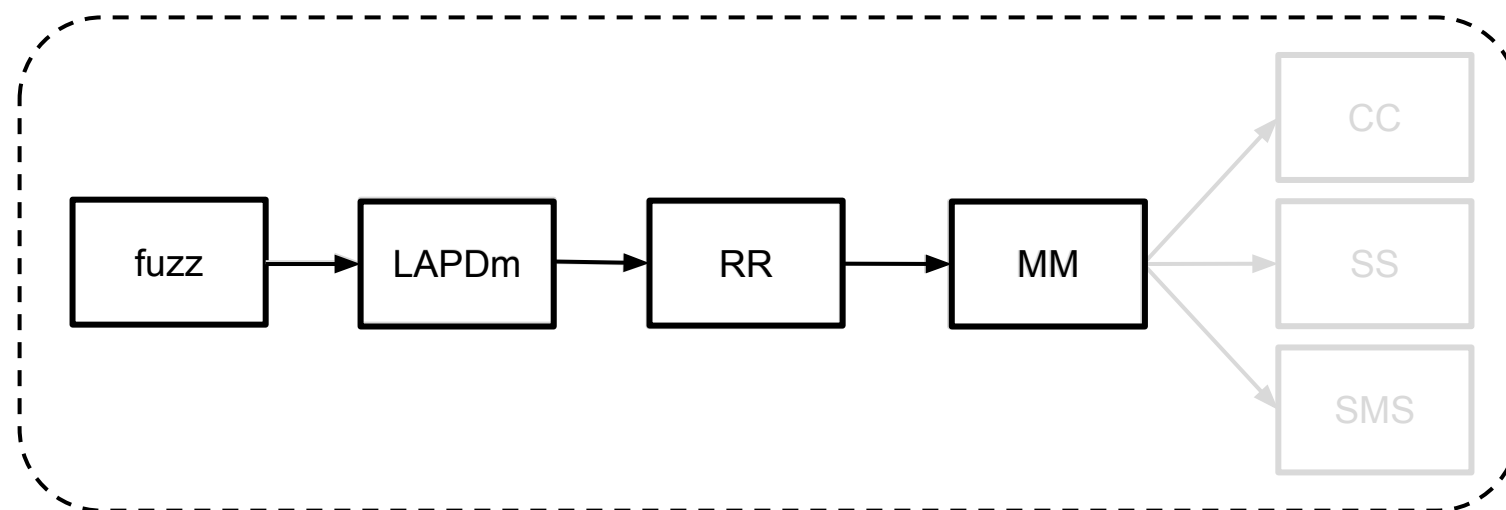
*Let's go fuzzing*



**AW DANG IT**

.... but we initialized all tasks?

```
[GLADD] 0x4105a44b 0b10: [ / / /HEDGE/GSM/GL2/GLADD/Code/Src/dl_util.c] - Add_concat_buf : 4462222c  
[GRR] 0x40f0f8d7 0b10: [../../../../HEDGE/GSM/GL3/GRR/Code/Src/rr_dedi.c] - Receive Data from Network (BTS)  
[GRR] 0x40f0f907 0b10: [ / / /HEDGE/GSM/GL3/GRR/Code/Src/rr_dedi.c] - (CTI)## - In_SemRecBtsData - TTD .  
[MM] 0x40f439af 0b10: [../../../../HEDGE/NASL3/MM/Code/Src/mm_Main.c] - STACK ID - 0  
[MM] 0x40f43561 0b10: [../../../../HEDGE/NASL3/MM/Code/Src/mm_Main.c] - MM Message Count -> 5  
[MM] 0x40f5546d 0b10: [../../../../HEDGE/NASL3/MM/Code/Src/mm_CellIndication.c] - Resetting Sent Reg Status  
[MM] 0x41359a0f 0b1: [../../../../HEDGE/NASL3/MM/Code/Src/mm_Uutilities.c] - TRAP MESSAGE !!! - Invalid  
Request for this State in mm_DecodeRrDataIndMsg !!!
```



## Challenge 2 : Initialize the state

### (1) Identifying crucial state beyond task initialization

```
iVar2 = mm_GetState();  
uVar3 = CONCAT44(iVar2 + -6, iVar2);  
switch(iVar2) {  
case 6:  
case 7:  
case 8:  
case 9:  
case 0xd:  
case 0xf:  
    break;  
default:  
    mm_printTrapMessage("mm_DecodeRrDataIndMsg", iVar2 + -6, local_10, local_c);  
    return;  
}
```

returns **\*(uint8\_t\*) 0x42e22f58**

```
[1.67198][MM] 0x41359a0f 0b1: [../.././HEDGE/NASL3/MM/Code/Src/mm_Utilities.c] - TRAP  
MESSAGE !!! - Invalid Request for this State in mm_DecodeRrDataIndMsg !!!
```

## Challenge 2: Initialize the state

### (2a) Fix up the state variables

- We need to fake a valid connection state

```
#ifdef SAMSUNG_S10e  
[...]  
  
uint32_t mm_state_addr = 0x42e22f58;  
[...]  
#endif
```

- Simple state constants, as well as ..

```
[...]  
  
//make sure the mm state is 9  
*(uint8_t*)mm_state_addr = 0x9;  
  
[...]
```

## Challenge 2: Initialize the state

(2b) .. more advanced state

```
#ifdef SAMSUNG_S10e
[...]
```

`uint32_t rr_serv_cell_addr = 0x4182cdd8;`

```
[...]
```

`#endif`

```
struct rr_servingCell{
    uint16_t arfcn;
    uint16_t rxLvl;
    uint8_t[0x17] unk;
    uint8_t[0x3] mnc_mmc;
    uint16_t lac;
    uint8_t[0xd0] unk2;
} PACKED;
```

Wait, unknown?

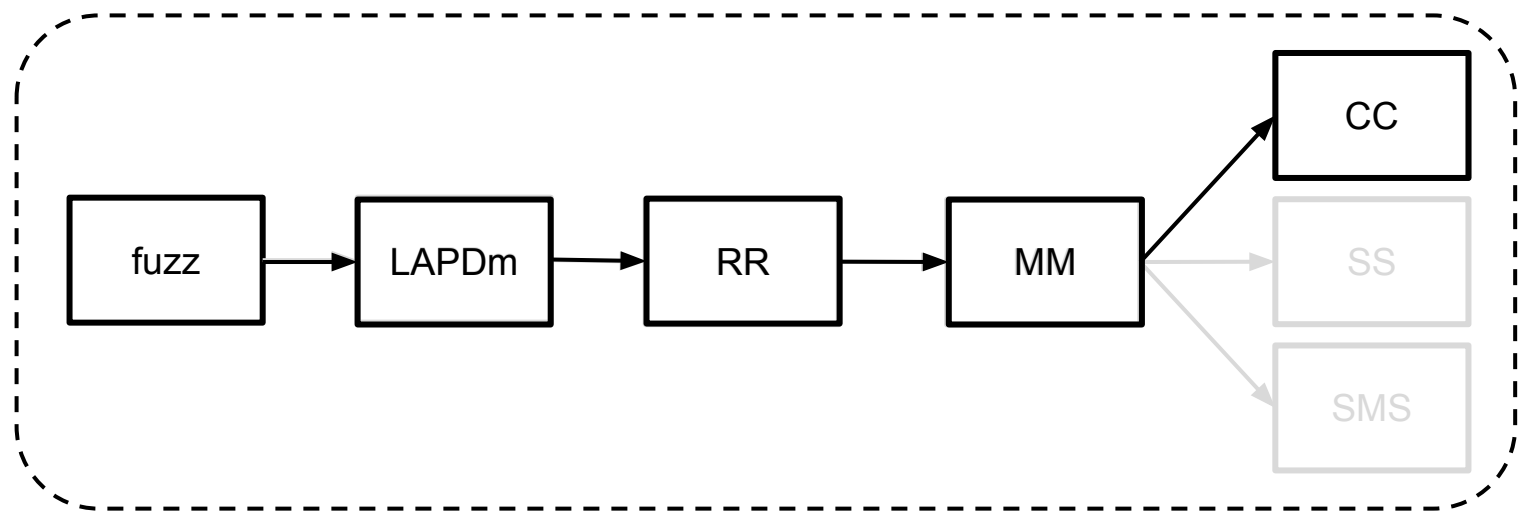
Only restore crucial state

```
[...]
```

`struct rr_servingCell *rr_servCell;``rr_servCell = alloc(0xec);``memset(rr_servCell, 0x0, 0xec);`  
`rr_servCell->arfcn = 0x35d;``rr_servCell->mnc_mmc = 0x1869f;``rr_servCell->lac = 0x3e8;`  
`*rr_serv_cell_addr = rr_servCell;`

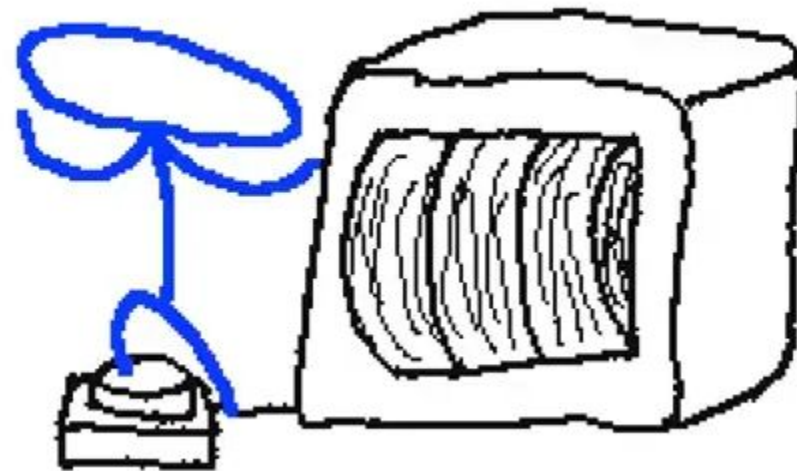
```
[...]
```

```
[GLADD] 0x4105a44b 0b101: [../../../../HEDGE/GSM/GL2/GLADD/Code/Src/dl_util.c] - Add_concat_buf : 4462223c
[GRR] 0x40f0f8d7 0b10: [../../../../HEDGE/GSM/GL3/GRR/Code/Src/rr_dedi.c] - Receive Data from Network (BTS)
[MM] 0x40f439af 0b10: [../../../../HEDGE/NASL3/MM/Code/Src/mm_Main.c] - STACK ID - 0
[CC] 0x40fe8cfd 0b101: [../../../../HEDGE/NASL3/CC/Code/Src/cc_Main.c] - ----- CC TASK
-----
[CC] 0x40fe87e3 0b10: [../../../../HEDGE/NASL3/CC/Code/Src/cc_Main.c] - cc_UpdStackId :CcCurrentStackId: 0
[CC] 0x40ac21b1 0b101: [../../../../HEDGE/NASL3/CC/Code/Src/cc_MsgDescription.c] - cc_MapSubTypeToMessageNum
SubType = 0x5
```



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

*Let's go fuzzing pt 2*



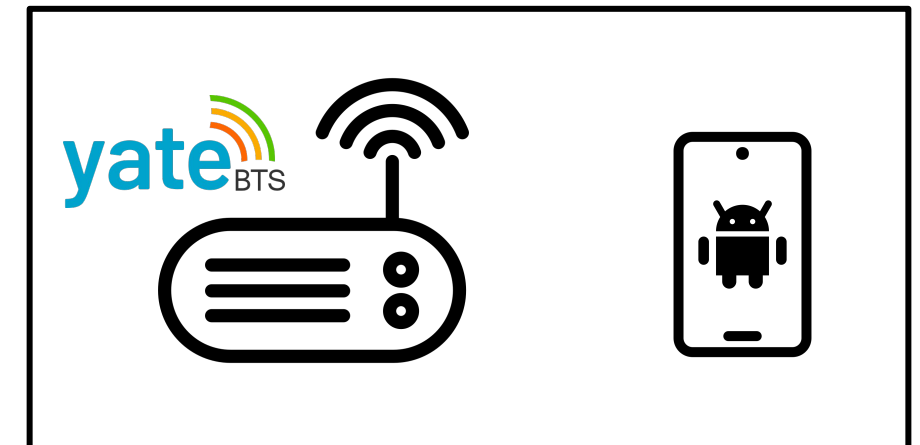
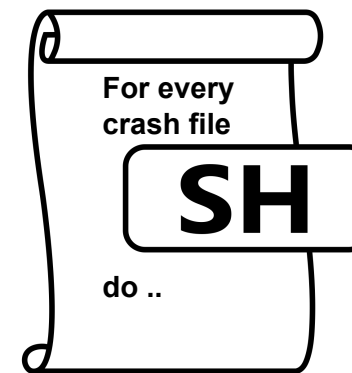
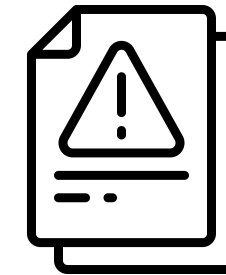


```
american fuzzy lop ++4.22a {default} (./firmwire.py) [explore]
┌─── process timing ────┐ ┌─── overall results ────┐
│   run time   : 0 days, 1 hrs, 27 min, 54 sec      │ cycles done   : 0      │
│  last new find : 0 days, 0 hrs, 0 min, 0 sec      │ corpus count  : 1962  │
│ last saved crash : 0 days, 0 hrs, 2 min, 18 sec  │ saved crashes  : 15   │
│ last saved hang  : none seen yet                 │ saved hangs   : 0     │
├─── cycle progress ───┘ └─── map coverage ────┘
│ now processing  : 1465.0 (74.7%)                  │ map density   : 1.63% / 27.95% │
│ runs timed out : 0 (0.00%)                       │ count coverage : 1.66 bits/tuple │
├─── stage progress ───┘ ┌─── findings in depth ────┘
│ now trying     : havoc                            │ favored items  : 981 (50.00%) │
│ stage execs    : 2275/7680 (29.62%)              │ new edges on  : 1284 (65.44%) │
│ total execs    : 222k                            │ total crashes : 29 (15 saved) │
│ exec speed     : 40.40/sec (slow!)                │ total tmouts  : 0 (0 saved)  │
├─── fuzzing strategy yields ───┘ ┌─── item geometry ────┘
│ bit flips     : 20/7640, 7/7623, 11/7589          │ levels        : 20           │
│ byte flips    : 1/955, 3/938, 6/904              │ pending       : 1716          │
│ arithmetics   : 48/66.5k, 2/128k, 0/122k          │ pend fav      : 783           │
│ known ints    : 5/8480, 26/35.3k, 25/50.3k        │ own finds     : 1896          │
│ dictionary    : 0/0, 0/0, 0/0, 0/0              │ imported      : 0             │
│ havoc/splice  : 1009/122k, 644/69.7k             │ stability     : 99.96%       │
│ py/custom/rq  : unused, unused, unused, unused   │                                         │
│ trim/eff      : 0.37%/4588, 97.17%              │                                         │
├─── strategy: explore ───┘ ┌─── [cpu000: 37%] ────┘
│ state: in progress ───┘
```

```
[7.13763][MM] 0x40aaa399 0b101: [../../../../HEDGE/NASL3/MM/Code/Src/mm_Information.c] - Short Network Name
->
[7.13818][MM] 0x40aaa2f1 0b101: [../../../../HEDGE/NASL3/MM/Code/Src/mm_Information.c] - LSA Identity of
the Serving Cell ->
[...]
[7.17115][MM] 0x40f40ab1 0b11: [../../../../HEDGE/NASL3/MM/Code/Src/mm_Main.c] - 19. MM <== <RADIO MSG>
MM_INFORMATION
[7.17151][MM] 0x40f40ab1 0b11: [../../../../HEDGE/NASL3/MM/Code/Src/mm_Main.c] - 20. MM ==>
MM_NS_INFORMATION_IND
[7.17190][MM] 0x40f40ab1 0b11: [../../../../HEDGE/NASL3/MM/Code/Src/mm_Main.c] - 21. MM ==>
MM_NS_SERVICE_IND
[7.17227][MM] 0x40f40ab1 0b11: [../../../../HEDGE/NASL3/MM/Code/Src/mm_Main.c] - 22. MM ==>
MM_PLMN_UPDATE_MMINFO_IND
[...]
[7.18368][REG_SAP] 0x407d9a1f 0b11: [../../../../PSS/StackService/CNS/Common/Code/Src/ns_MsgHandler.c] -
REG_SAP ==> NS_INFORMATION_IND (Mbx 5)
[7.18412][REG_SAP] pal_MsgSendTo+0x3ff (0x4100460f) pal_MsgSendTo(ATI (5)) - PALMsg<0x760d, REG_SAP (a5)
-> ATI (5), 99 bytes>
[ERROR] firmware.vendor.shannon.hooks: FATAL ERROR (REG_SAP): from 0x40effd05 [pal_PlatformMisc.c:146 -
Fatal error: PAL_MEM_GUARD_CORRUPTION pal_MemInterface.c Line 895]
```

## Challenge 3: No support for recent phones

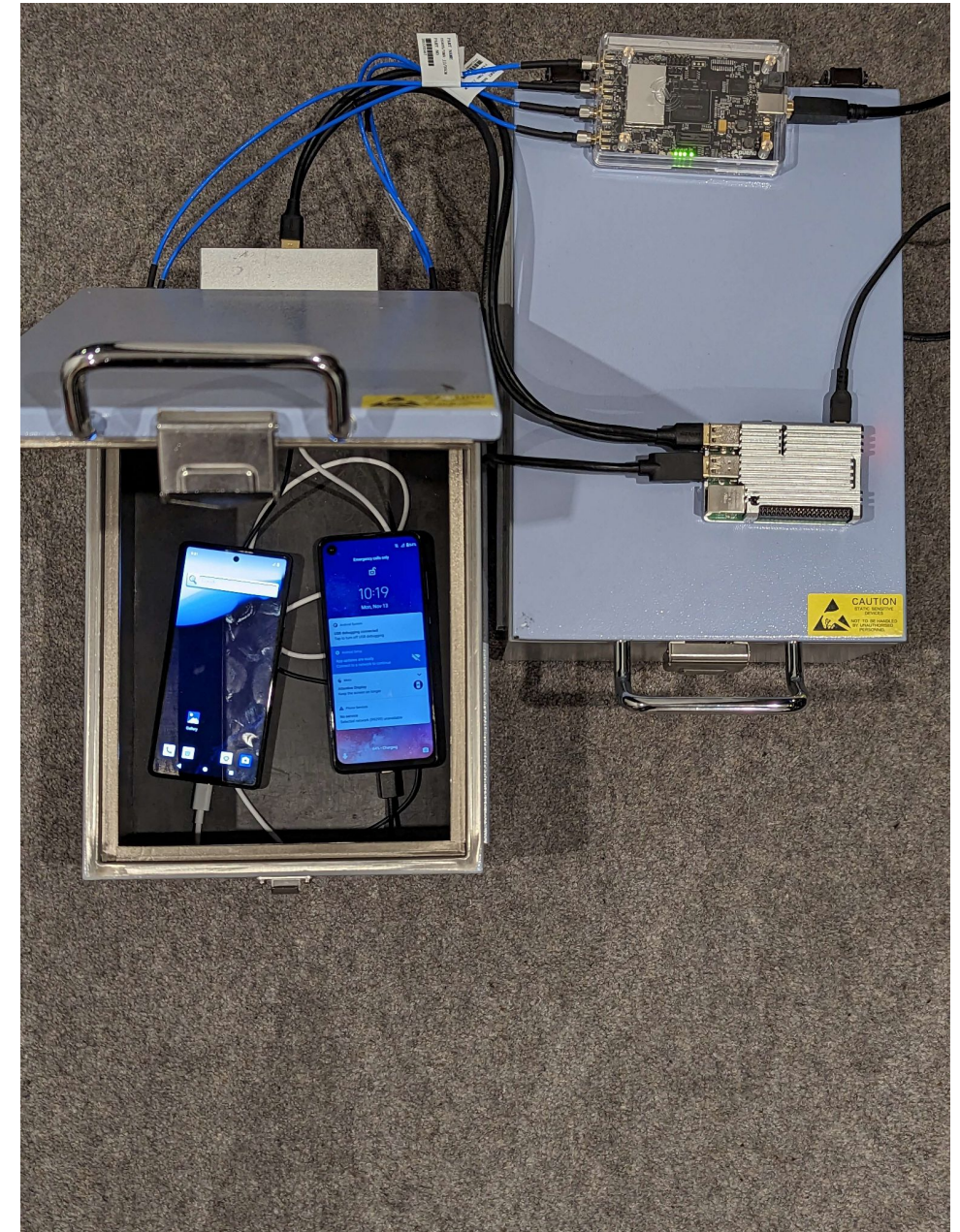
- Our fuzzing targets firmware from early 2023 or before
- Idea: Confirm vulnerabilities OTA against newer devices



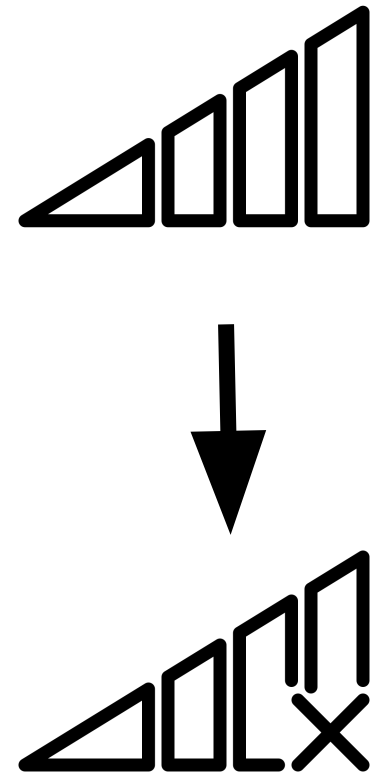
# OTA Setup

- Hardware
  - Faraday Cage
  - SDR (BladeRF 2.0 micro xA4)
  - Raspberry Pi 4
- Software

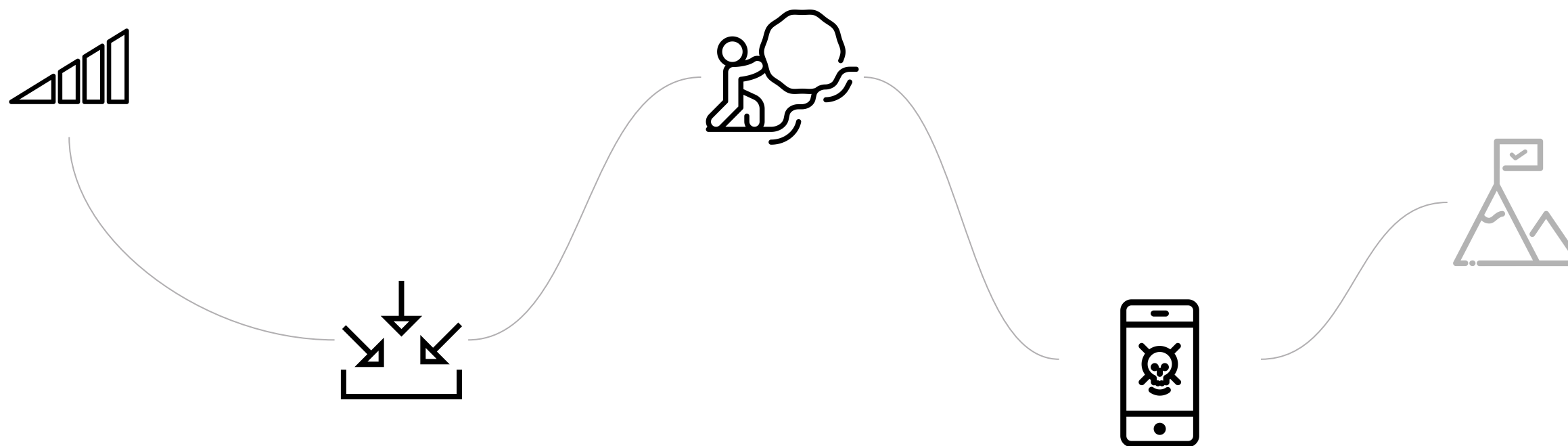
Open source GSM Base Station software :  
Yate v6.2.1 / YateBTS v6.1.1
- Tested Phones:
  - Motorola One Vision, Samsung Galaxy S10e
  - Google Pixel 6 and 8
  - Samsung Galaxy S22, A14



```
02-24 07:11:12.305 1098 1161 D RFSD      : [ModemStateMonitor::OnModemCrashOrReset]
Modem is STATE_CRASH_EXIT or STATE_CRASH_RESET
02-24 07:11:14.286 1088 1219 D DMD      : ModemStateMonitor : Modem CRASH!!![2]
02-24 07:11:14.286 1088 1219 D DMD      : ModemStateMonitor : Check the state
again after 2 seconds later.
02-24 07:11:14.305 1098 1161 D RFSD      : [ModemStateMonitor::OnModemCrashOrReset]
Modem is STATE_CRASH_EXIT or STATE_CRASH_RESET
02-24 07:11:16.286 1088 1219 D DMD      : ModemStateMonitor : Modem CRASH!!![2]
02-24 07:11:16.287 1088 1219 D DMD      : ModemStateMonitor : Check the state
again after 2 seconds later.
02-24 07:11:16.306 1098 1161 D RFSD      : [ModemStateMonitor::OnModemCrashOrReset]
Modem is STATE_CRASH_EXIT or STATE_CRASH_RESET
02-24 07:11:18.287 1088 1219 D DMD      : ModemStateMonitor : Modem CRASH!!![2]
02-24 07:11:18.287 1088 1219 D DMD      : ModemStateMonitor : Check the state
again after 2 seconds later.
```



# Found Vulnerabilities



# Showcase

	ID	Tested Phones	Affected Protocol	What?	Severity	Reported in
Previously Unknown	CVE-2023-50807	Pixel 6	MM	Heap-based buffer overflow	8.1	Oct-23
	CVE-2023-50805	Pixel 6, 8, S22	RR	Heap-based buffer overflow	8.1	Oct-23
	CVE-2024-28068	S22, A14	SS/GPRS	Null-Pointer Deref	5.3	Jan-24
Duplicates	N/A - internal find	Pixel 6, S22, A14	CC/SS	Heap-based buffer overflow	N/A	Jan-24
	N/A - internal find	A14	SMS	Stack-based buffer overflow	N/A	Jan-24
Under Disclosure	CVE-XXXX-YYYYYY CVE-XXXX-YYYYYY CVE-XXXX-YYYYYY	-	-	-	-	-

# Example bug : CVE-2023-50807\* (MM)

- MM Information message
  - Holds information about the network

```
05 32 45 8c 03 03 03 ...  
48 41 41 00 01 04 00 ...
```

IE	Presence	Value
MM PD	M	0x05
MsgType	M	0x32
Full Network Name	O	0x43
Short Network Name	O	0x45
Local Time Zone	O	0x46
Universal time	O	0x47
LSA Identity	O	0x48
Network Daylight Saving Time	O	0x49



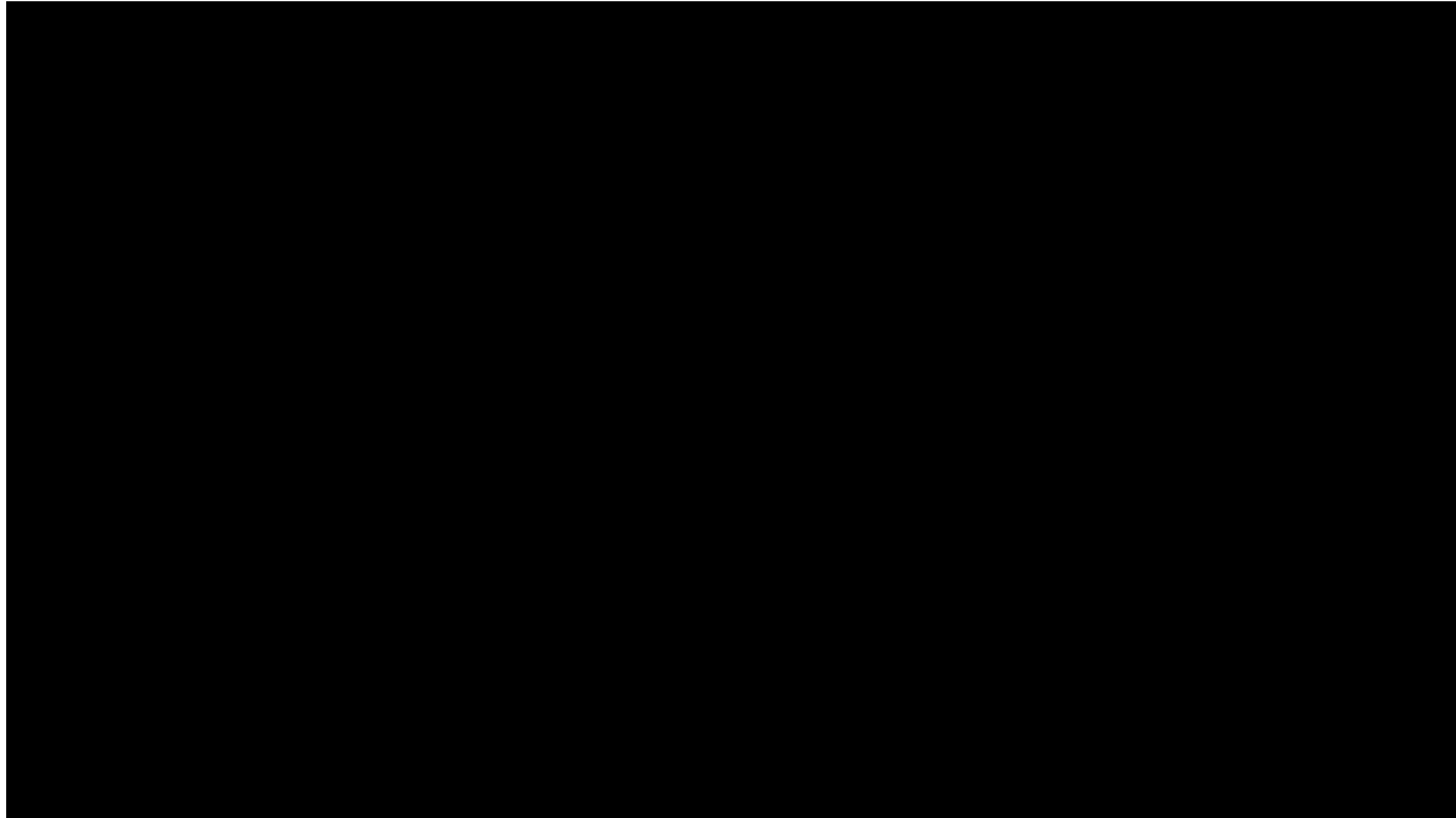
## Example Bug: CVE-2023-50807 (MM)

```
05 32 45 8c 03 03 03 ...  
48 41 41 00 01 04 00 ...
```

```
void mm_getIe(uint8_t *src, int p2){  
    uVar1 = mm_getIeLength(p2);  
    if((p2 == 0x3c) && (3 < uVar1)) uVar1 = 3;  
    memcpy(src, IE_buf, uVar1);  
}
```

```
void ns_ServiceHandlerEmm(..){  
    uint8_t* buf = mem_Alloc(0x6f);  
    memcpy(buf+0x58, src, IE_len);  
}
```

- LSA Identity IE
- LSA Identity IE length
- LSA Identity IE length should be fixed to 3 bytes
- Value from the first check is not propagated!



# Defenses

- Recent shift in vendor's approaches
  - More hardening for basebands!
- Recently introduced defenses:
  - Heap Sanitization (Pixel 8)
  - Stack cookies
  - More consistent use of XN
  - Allow 2G

## Google Security Blog

The latest news and insights from Google on security and safety on the Internet

### Hardening cellular basebands in Android

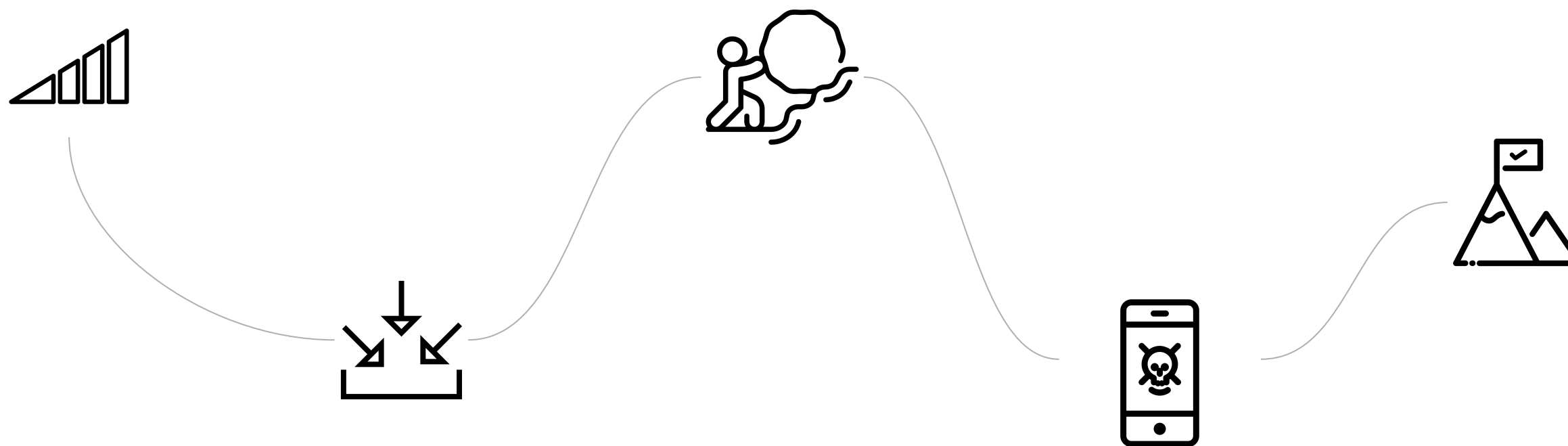
December 12, 2023

#### Allow 2G

2G is less secure, but may improve your connection in some locations. For emergency calls, 2G is always allowed.



# Wrapping Up



## Limitations/Future Work

- State needs to be reversed for every image
  - Potential for automation
- What if we cannot replicate a vulnerability OTA?
  - FirmWire support for firmware targeting recent firmware
- What about 3G/4G/5G?
  - More reversing, more harnessing, more state, more everything

# Key Takeaways

- State is key to overcome fuzzing roadblocks when fuzzing across communication stacks
- Fuzzing older firmware images can lead to discovering vulnerabilities in the newest devices
- Despite years of research, critical vulnerabilities still hide in 2G implementations

# Questions



Or via email:

<dyonwg@gmail.com>

<m.muench@bham.ac.uk>