# A walk with Shannon

## Walkthrough of a pwn2own baseband exploit.

@amatcama

# Introduction

## Amat Cama

- Independant Security Researcher.
- CTF player @ Shellphish.
- Exploitation and Reverse Engineering.
- Currently interested in Hypervisors and Baseband security reseach.

## Previously

- Security Consultant - ***Virtual Security Research***.
- Research Assistant - ***UCSB Seclab***.
- Product Security Engineer - ***Qualcomm Inc***.
- Senior Security Researcher - ***Beijing Chaitin Tech Co., Ltd***.

# Agenda

- Prior Work.
- Motivation.
- Cellular Networks ? Baseband ?
- The Shannon Baseband.
- Hunting for Bugs.
- Demo.
- Conclusions.

# Prior Work

# Prior Work

- **“Breaking Band - reverse engineering and exploiting the shannon baseband”** – *Nico Golde and Daniel Komaromy*.
- Very useful talk if you want to do research on the shannon baseband. Lots of scripts and information that will definitely be of help.

# Motivation

# Motivation

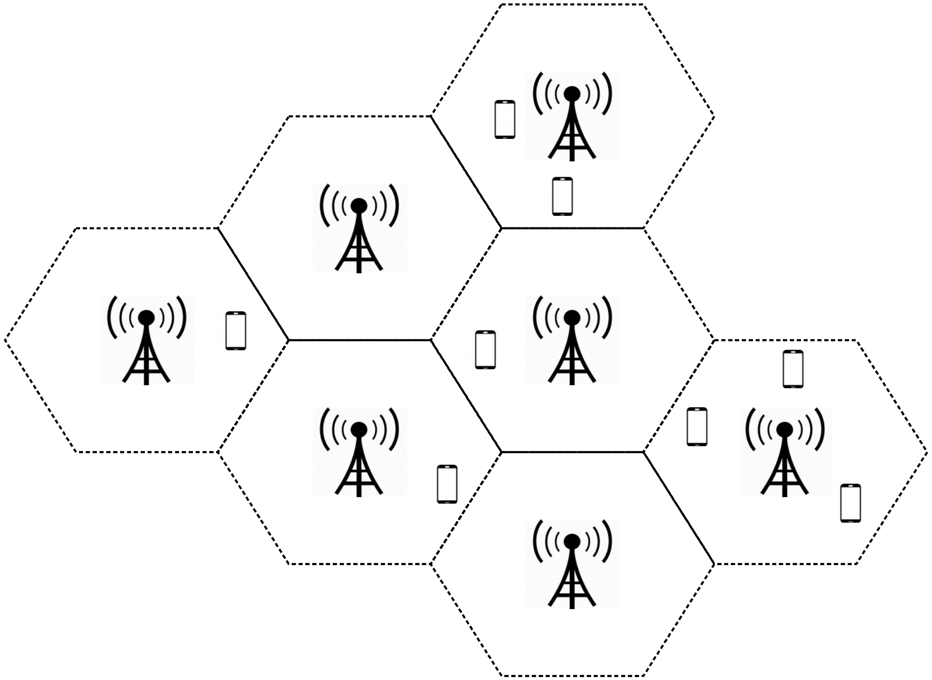
- Because it is fun.
- Unexplored area of research; great opportunity to learn.
- Many bugs.
- Big impact.
- Pwning a phone just by having it connect to a cellular network sounds pretty cool.

**Cellular Networks ?**
**Baseband ?**

# Cellular Networks ? Baseband ?

## What is a Cellular Network ?

- Mobile communication network.
- “Cells” are land areas covered by a *base transciever station (**BTS**)*.
- To cover a large area, the cells are used in junction: *A Cellular Network*.
- Technically could be any kind of network, today mostly *Mobile Phone Network*.



# Cellular Networks ? Baseband ?

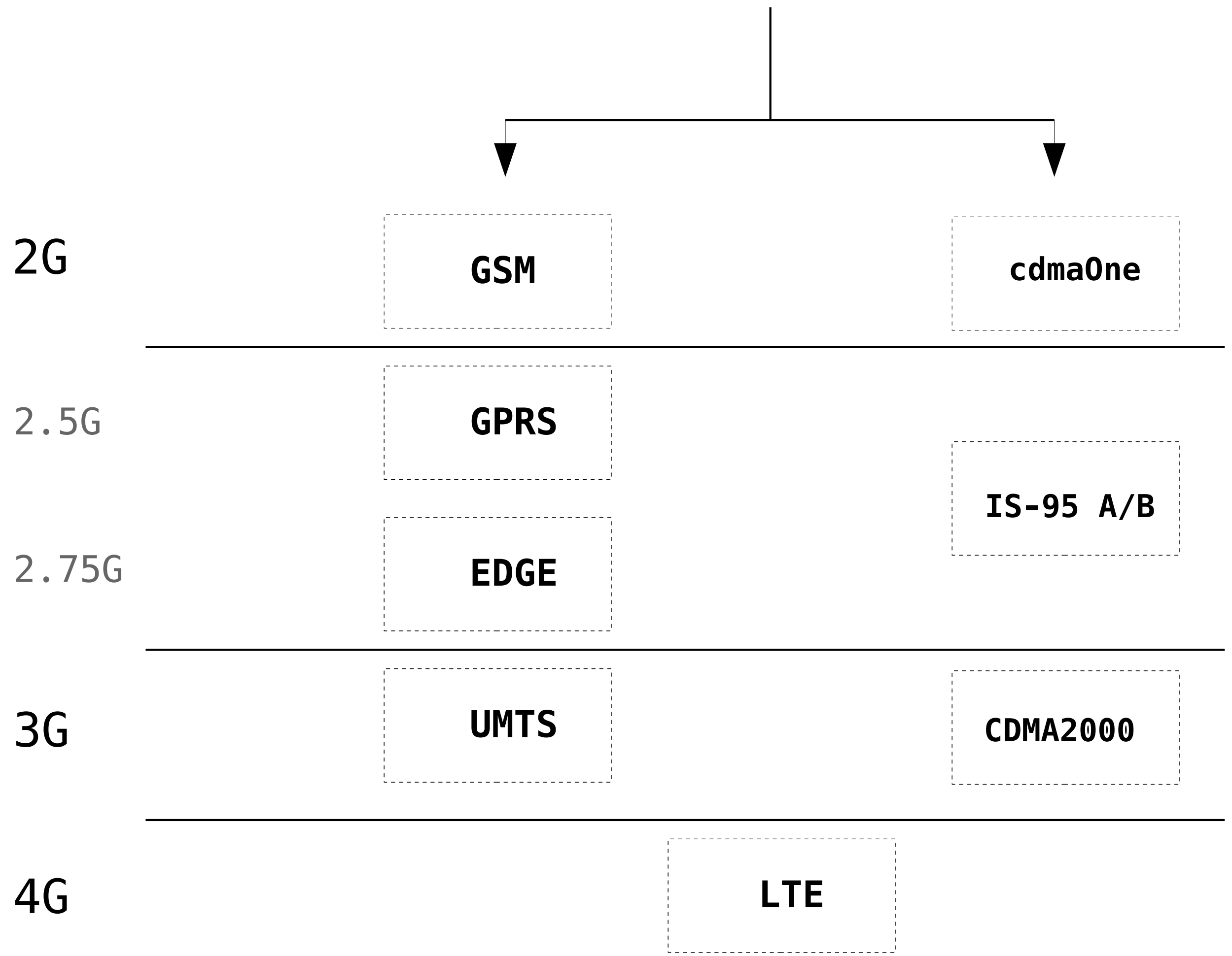
## The technologies and standards (I)

- A number of technologies and standards developped.
- Different generations with improving speeds and capacity.
- Competing technologies for different generations.

# Cellular Networks ? Baseband ?

## The technologies and standards (II)

- Mainly two branches: ***GSM*** branch and ***CDMA*** branch



# Cellular Networks ? Baseband ?

## The technologies and standards (III)

- *3GPP* is a collaboration agreement with a number of telecommunication standard bodies.
- Provides maintenance and development of the GSM *Technical Specifications (TS)*
  - GSM
  - GPRS / EDGE
  - UMTS
  - LTE
- Is Comprised of bodies such as the *European Telecommunications Standards Institute (ETSI).*
- The technical standards provide detailed information on the structure of messages exchanged.

# Cellular Networks ? Baseband ?

## The Protocol Stack

| | GSM | GPRS/EDGE | LTE |
|---|---|---|---|
| Network Layer | **CM** - Connection Management<br>**MM** - Mobility Management<br>**RR** - Radio Resource Management | **SM** - Session Management<br>**GMM** - GPRS Mobility Management<br>**GRR** - GPRS Radio Resource Management | **IP** - Internet Protocol / **NAS** - Non-Access Spectrum<br>**RRC** - Radio Resource Control |
| Data Link Layer | **LAPDm** | **SNDCP** - Subnetwork Dependent Convergence Protcol<br>**RLC/LLC** - Radio/Logical Link Control<br>**MAC** - Media Access Control | **PDCP** - Packet Data Convergence Protcol<br>**RLC/LLC** - Radio/Logical Link Control<br>**MAC** - Media Access Control |
| Physical Layer | | | |

# Cellular Networks ? Baseband ?

## The Baseband (I)

- Component of the phone in charge of handling communication with the mobile network.
- Deals with low level radio signal processing.
- Supports a number of standards (GSM, 3G, 4G, 5G, cdmaOne, CDMA2000, ...).
- Basically the main “interface” to the mobile network.

# Cellular Networks ? Baseband ?

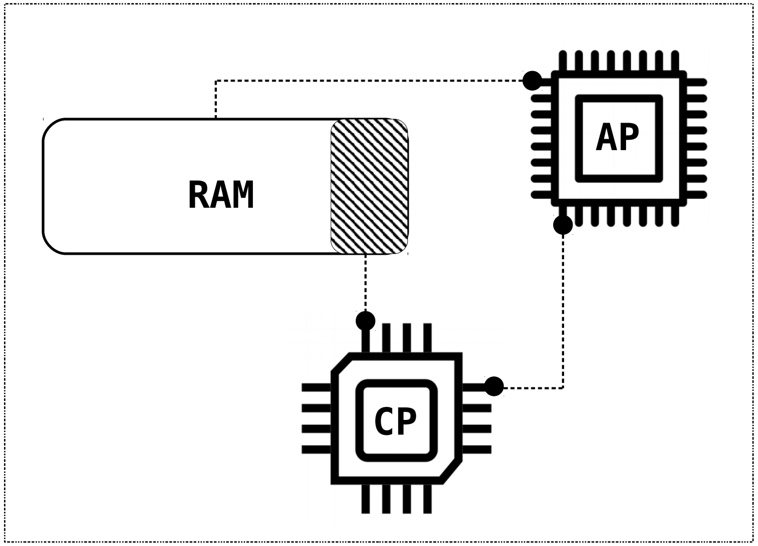
## The Baseband (II)

- A number of different implementations.
- Qualcomm owns most of the market.
- Qualcomm: Galaxy, iPhone, OnePlus, Pixel, Xperia, HTC, LG, ASUS, Motorola, ...
- Huawei: Mate 10, P20, Honor 9, ...
- **Samsung**: Galaxy S6, S7, S8, S9, ...
- Intel: iphone X, iphone 8, ...

# Cellular Networks ? Baseband ?

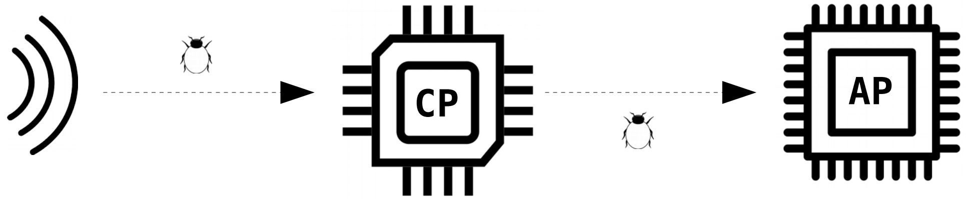
## The Baseband (III)

- The most common architecture today: baseband firmware runs on a dedicated chip; the *cellular processor (**CP**)*.
- This chip is tasked with all of the radio processing.
- The code is generally written in low level languages such as C/C++.
- A communication interface between **CP** and **AP** (Application Processor) such as shared memory, serial or interrupts.



## The Baseband (IV)

- Getting code execution on the *CP* doesn't necessarily result in owning the whole device.
- A number of attacks can be performed:
  - Redirect/Intercept phone calls.
  - Redirect/Intercept SMS.
  - Modify Internet traffic.
  - ...
- A step further; attack the *AP* through the IPC mechanisms and gain full control of the device.



# The Shannon Baseband

# The Shannon Baseband

## About Shannon

- Samsung’s Baseband implementation.
- Typically ships with phones featuring the Exynos SoC.
- e.g: most non-US Galaxy phones.
- A RTOS running on an ARM Cortex R7.

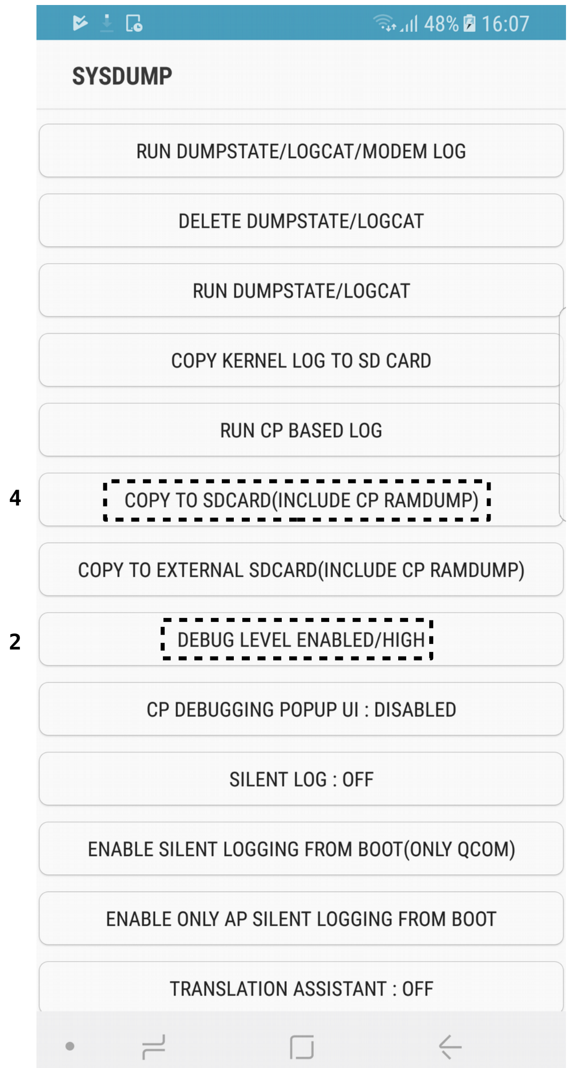
# The Shannon Baseband

## Obtaining the code (I)

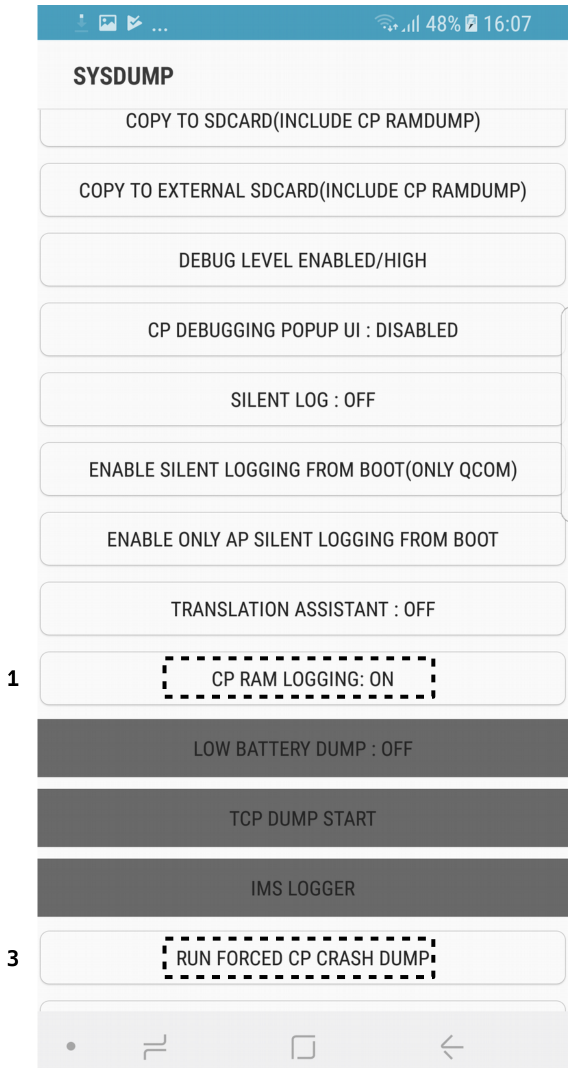
- The modem firmware can be obtained from the phone’s firmware images.
- However it is encrypted and doesn’t seem to be an easy way to decrypt it.
- Luckily it is possible to make the phone generate modem RAM dumps.
- Dialing the code *#9900# brings up the *SYSDUMP* menu.

# The Shannon Baseband

## Obtaining the code (II)




Page 1

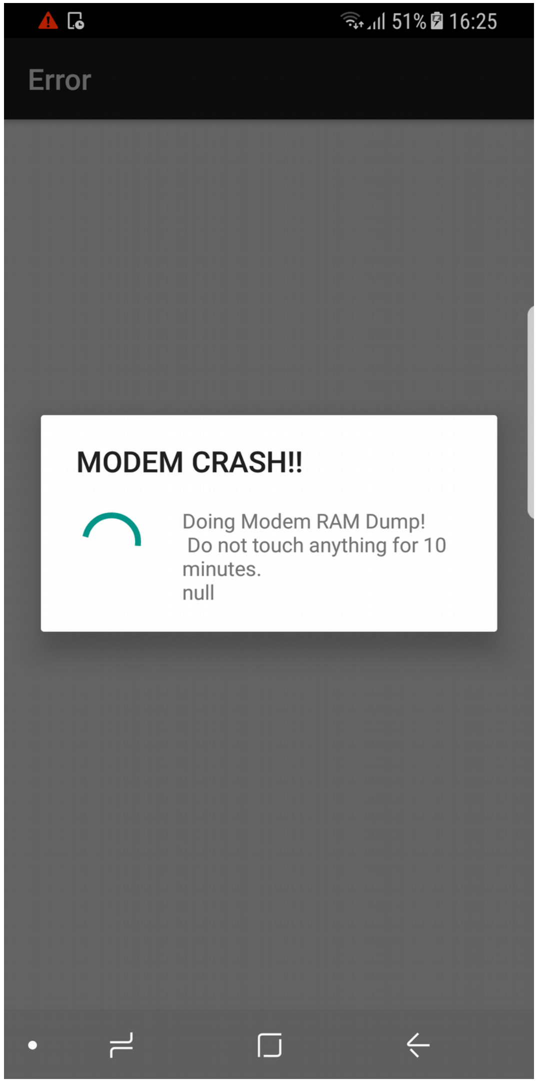


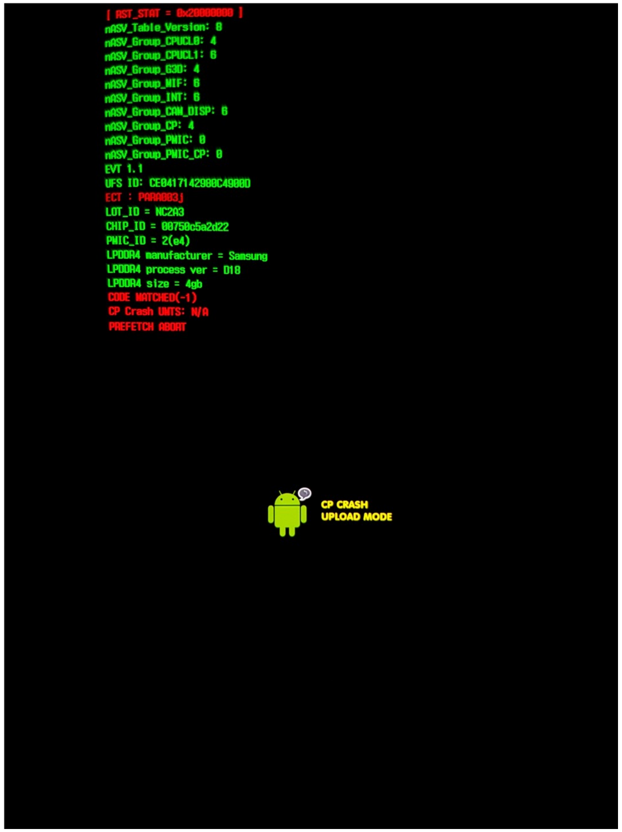

Page 2

## Obtaining the code (III)

- Tap on the `DEBUG LEVEL ENABLED/` option and set it to `High`. The phone will reboot.
- Reopen the SYSDUMP menu, scroll down and tap on the `CP RAM LOGGING` option and set it to `On`. The phone will reboot.
- Reopen the SYSDUMP menu and scroll all the way down, tap the `RUN FORCED CP CRASH DUMP` option. The phone will reboot and go into the ram upload mode. Hold the power and volume down button for 10 seconds to turn the phone off and then power it back on.
- Reopen the SYSMDUMP menu and tap the `COPY TO SDCARD(INCLUDE CP RAMDUMP)` option.
- Now in the folder `/sdcard/log` of the device, we have the log files including the ram dump. Largest file in the folder and has a name of the following format `cpcrash_dump_YYYYMMDD_HHSS.log`

## Obtaining the code (IV)



# The Shannon Baseband

## Loading Code in IDA

- The CP Boot Daemon (/sbin/cbd) handles powering on the modem and processing RAM dumps amongst other things.
- Boot code can be found at the start of the encrypted modem image in the firmware packages.
- By reversing the cbd and boot, we can translate the file offsets of the RAM dump to virtual addresses:

  **0x40000000**

  **0x8000000**

  **0x4000000**

  **0x20000**

  **0x4800000**

  **0x4000**

  **0x3E00**

  **0x200**

# The Shannon Baseband

Identifying Tasks

- We need to identify the different tasks run by the RTOS.
- Start reversing from RESET Exception Vector Handler…
- Look at the start of the different memory regions and you recognize the Exception Vector Table in one of them.
- A linked list contains all the different tasks’ entry points, corresponding stack frames and task names (very useful).
- Traverse the list and identify all the tasks.

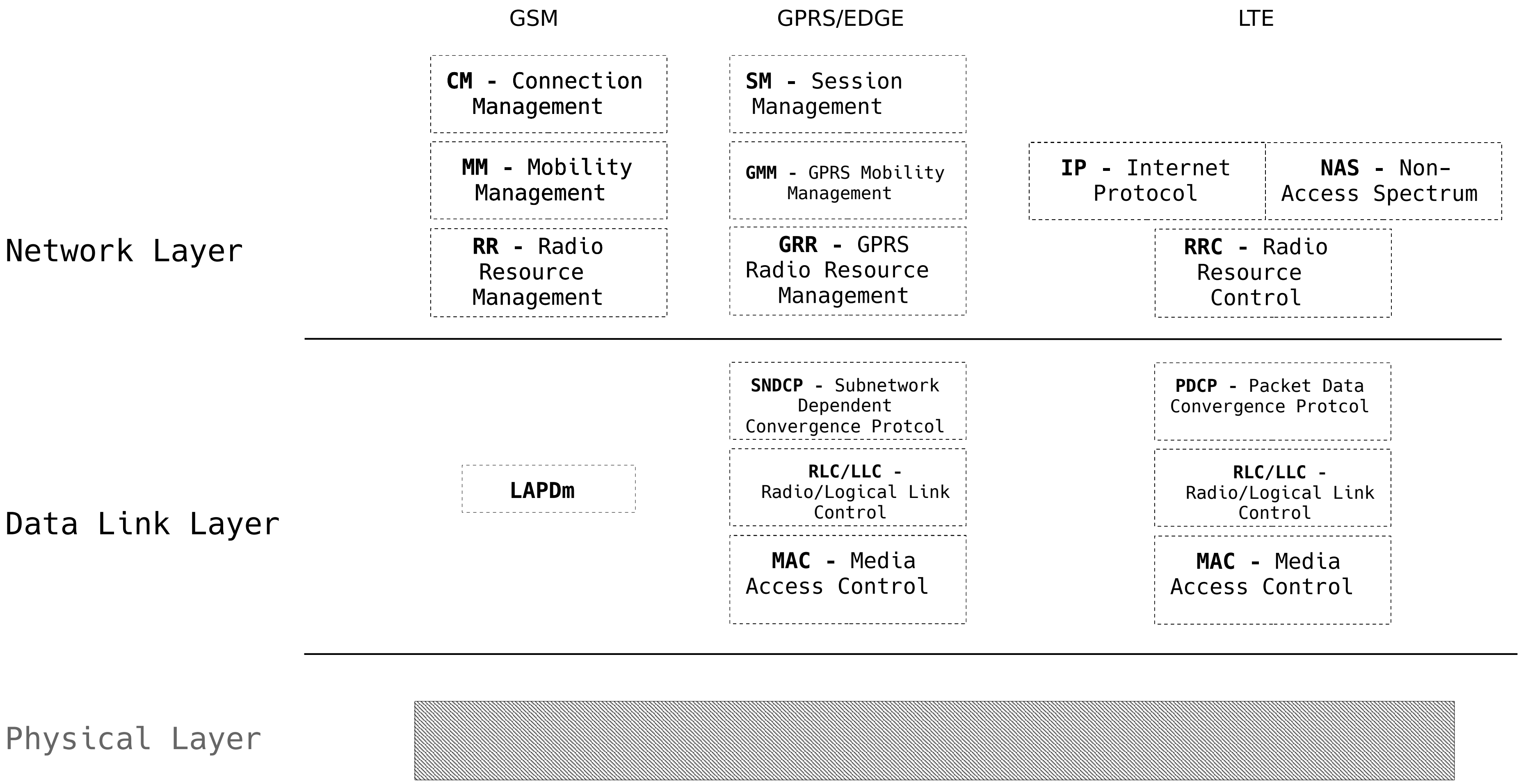
# The Shannon Baseband

## The Tasks (I)

- We end up with a list of tasks with different names, some of them self-explanatory, some of them misleading, some of them hard to understand.
- **MM** (**M**obility **M**anagement ?)
- LLC
- SMS_SAP
- GRR
- SS
- SAEL3
- SNDCP
- **CC** (**C**all **C**ontrol ?)
- **SM** (**S**ession **M**anagement ?)
- LLC
- ...

# Cellular Networks ? Baseband ?

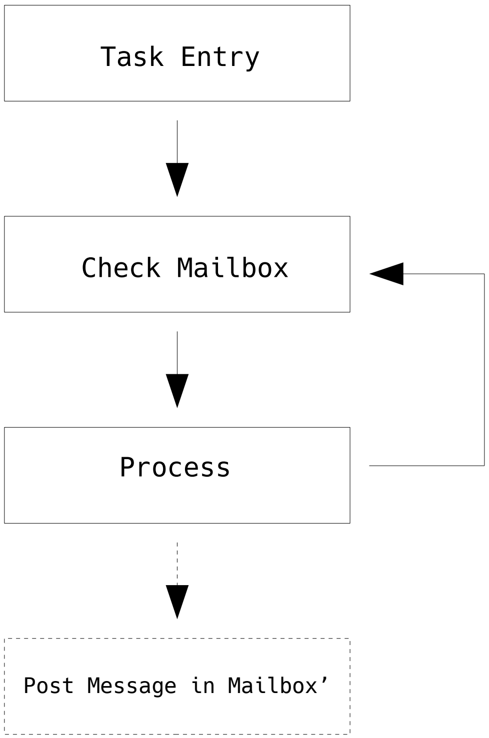
- The Tasks (II)



# The Shannon Baseband

## The Tasks (III)

- Different tasks are used for different components and layers of the protocol stacks.
- Tasks communicate with each other using a *mailbox* system.
- Tasks are pretty much *while* loops waiting to process messages (from other tasks).



# The Shannon Baseband

## The Tasks (IV)

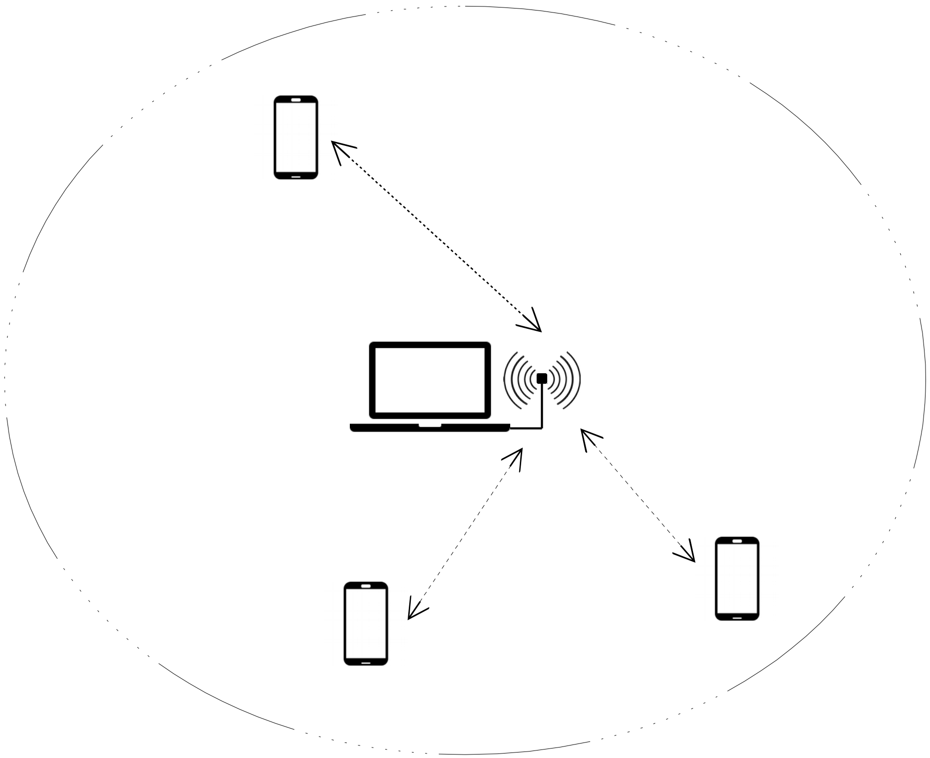
- Pick a task and start reversing.
- The Code is pretty generous in that it contains a **lot** of strings.

# Hunting for Bugs

# Hunting for Bugs

## Setting up an environment (I)

- The goal is to be able to send arbitrary data the the baseband.
- Need to operate our own cellular network.
- Can be achieved with a *Software Defined Radio (**SDR**)*.
- The Mobile Network Stack / Standard is implemented in software that runs on our computers.
- The SDR (device) is a general purpose transciever that supports different frequencies.



# Hunting for Bugs

Setting up an environment (II)

- A number of different options for the SDRs.
  - *BladeRF x40*: $420.00
  - ***BladeRF x115***: $650.00
  - *USRP B200*: $675.00
  - *LimeSDR*: $300.00
  - *UmTRX*: $950.00 – $1300.00

# Hunting for Bugs

## Setting up an environment (III)

- A number of different options for software implementation of the standards.
- ***YateBTS*:**
  - Clean code, easy to modify.
  - Good support for *bladeRF*.
  - GSM and GPRS.
  - Easy to compile and run.
- ***OpenBTS*** *(OpenBTS-UMTS):*
  - Clean code, easy to modify.
  - Good support for *USRP* and *UmTRX*.
  - GSM, GPRS, 3G.
  - Easy to compile and run.

# Hunting for Bugs

## Setting up an environment (IV)

- *OpenBSC* (OsmoNITB, OsmBTS, …):
  - ➔ Good support for *USRP*, *LimeSDR* and *UmTRX*.
  - ➔ Compiling wasn't easy.
  - ➔ Clean code, easy to modify.
  - ➔ GSM + GPRS.
- *OpenAirInterface:*
  - ➔ Hard to compile and run.
  - ➔ Good support for *USRP*.
  - ➔ 4G.
- *OpenLTE:*
  - ➔ Hard to compile and run.
  - ➔ 4G.
  - ➔ Good support for *USRP*.
  - ➔ Clean code, easy to modify.

# Hunting for Bugs

Setting up an environment (V)

- Provisionned or programmable *SIM Cards* because 3G and 4G do not support open authentication.
- *Faraday Cage / RF Enclosure* because in most countries, operating a cell network without permission is **Illegal**!

# Hunting for Bugs

## Debugging The Phone

- Everytime the modem crashes we get a RAM dump.
- Luckily the dump contains the state of the registers at the time of the crash, therefore we have pretty decent post-mortem debugging capabilities.
- Write a script to process the dumps and do useful stuff (registers, peeking at memory).

# Hunting for Bugs

## Digging into the code (I)

- Back to picking a task to have a closer look at.
- An interesting approach is the following:
  - Layer 3 Messages are comprised of *Information Elements (IEs)*.
  - IEs are *V*, *LV*, *TLV*.
  - What are the different messages that can be sent to different components?
  - Cross Reference the Technical Standards to know the different message types sent to different components.
  - Read the description of the different messages and the content of the Information Elements. Are there (T)LVs ? Then reverse the corresponding task and try to find the code processing that particular IE.
  - A number of trivial bugs can be found this way...

# Hunting for Bugs

## Digging into the code (II)

- Let's clarify with an example.
- The CC task most likely stands for Call Control.
- Call control is a part of Connection Management in the GSM protocol stack.
- What are the different CC messages ?

# Hunting for Bugs

## Digging into the code (III)

**Table 9.54/3GPP TS 24.008: Messages for circuit-mode connections call control.**

| **Call establishment messages:** | **Reference** |
|---|---|
| ALERTING | 9.3.1 |
| CALL CONFIRMED (NOTE) | 9.3.2 |
| CALL PROCEEDING | 9.3.3 |
| CONNECT | 9.3.5 |
| CONNECT ACKNOWLEDGE | 9.3.6 |
| EMERGENCY SETUP (NOTE) | 9.3.8 |
| PROGRESS | 9.3.17 |
| CC-ESTABLISHMENT | 9.3.17a |
| CC-ESTABLISHMENT CONFIRMED | 9.3.17b |
| START CC | 9.3.23a |
| SETUP | 9.3.23 |
| **Call information phase messages:** | **Reference** |
| MODIFY (NOTE) | 9.3.13 |
| MODIFY COMPLETE (NOTE) | 9.3.14 |
| MODIFY REJECT (NOTE) | 9.3.15 |
| USER INFORMATION | 9.3.31 |
| **Call clearing messages:** | **Reference** |
| DISCONNECT | 9.3.7 |
| RELEASE | 9.3.18 |
| RELEASE COMPLETE | 9.3.19 |
| **Messages for supplementary service control** | **Reference** |
| FACILITY | 9.3.9 |
| HOLD (NOTE) | 9.3.10 |
| HOLD ACKNOWLEDGE (NOTE) | 9.3.11 |
| HOLD REJECT (NOTE) | 9.3.12 |
| RETRIEVE (NOTE) | 9.3.20 |
| RETRIEVE ACKNOWLEDGE (NOTE) | 9.3.21 |
| RETRIEVE REJECT (NOTE) | 9.3.22 |
| **Miscellaneous messages** | **Reference** |
| CONGESTION CONTROL | 9.3.4 |
| NOTIFY | 9.3.16 |
| START DTMF (NOTE) | 9.3.24 |
| START DTMF ACKNOWLEDGE (NOTE) | 9.3.25 |
| START DTMF REJECT (NOTE) | 9.3.26 |
| STATUS | 9.3.27 |
| STATUS ENQUIRY | 9.3.28 |
| STOP DTMF (NOTE) | 9.3.29 |
| STOP DTMF ACKNOWLEDGE (NOTE) | 9.3.30 |

## Digging into the code (IV)

**Table 9.67/3GPP TS 24.008: PROGRESS message content**

| IEI | Information element | Type/Reference | Presence | Format | Length |
|---|---|---|---|---|---|
| | Call control protocol discriminator | Protocol discriminator 10.2 | M | V | 1/2 |
| | Transaction identifier | Transaction identifier 10.3.2 | M | V | 1/2 |
| | Progress message type | Message type 10.4 | M | V | 1 |
| | Progress indicator | Progress indicator 10.5.4.21 | M | LV | 3 |
| 7E | User-user | User-user 10.5.4.25 | O | TLV | 3-131 |

# Hunting for Bugs

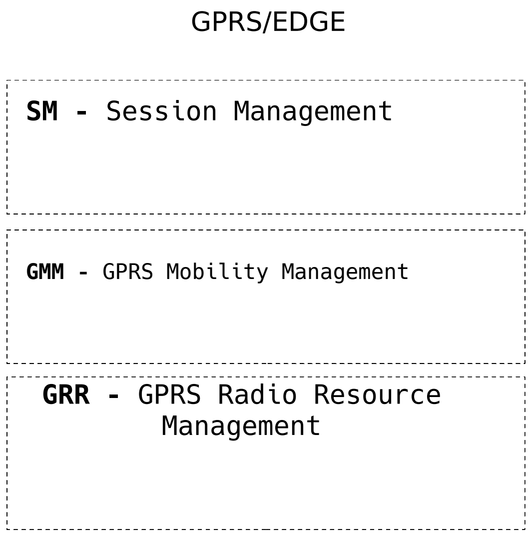
Digging into the code (V)

- Using this approach it is possible to find a number of trivial vulnerabilities.
- A previous bad experience competing at P2O taught me that trivial bugs are bad.
- Dig a bit deeper in order to find something less trivial and reduce the chance for collisions.

# Hunting for Bugs

## The Mobile Pwn2Own Bug (I)

- Decided to look at GPRS since it seems complicated?
- Start by reading the standards and looking at the GPRS Session Management Messages.



## The Mobile Pwn2Own Bug (II)

- The ACTIVATE PDP CONTEXT ACCEPT message looks good.

**Table 9.5.2/3GPP TS 24.008: ACTIVATE PDP CONTEXT ACCEPT message content**

| IEI | Information Element | Type/Reference | Presence | Format | Length |
|---|---|---|---|---|---|
| | Protocol discriminator | Protocol discriminator 10.2 | M | V | 1/2 |
| | Transaction identifier | Transaction identifier 10.3.2 | M | V | 1/2– 3/2 |
| | Activate PDP context accept message identity | Message type 10.4 | M | V | 1 |
| | Negotiated LLC SAPI | LLC service access point identifier 10.5.6.9 | M | V | 1 |
| | Negotiated QoS | Quality of service 10.5.6.5 | M | LV | 13-21 |
| | Radio priority | Radio priority 10.5.7.2 | M | V | 1/2 |
| | Spare half octet | Spare half octet 10.5.1.8 | M | V | 1/2 |
| 2B | PDP address | Packet data protocol address 10.5.6.4 | O | TLV | 4-24 |
| 27 | Protocol configuration options | Protocol configuration options 10.5.6.3 | O | TLV | 3-253 |
| 34 | Packet Flow Identifier | Packet Flow Identifier 10.5.6.11 | O | TLV | 3 |
| 39 | SM cause | SM cause 2 10.5.6.6a | O | TLV | 3 |
| B- | Connectivity type | Connectivity type 10.5.6.19 | O | TV | 1 |
| C- | WLAN offload indication | WLAN offload acceptability 10.5.6.20 | O | TV | 1 |
| 33 | NBIFOM container | NBIFOM container 10.5.6.21 | O | TLV | 3 – 257 |

# Hunting for Bugs

## The Mobile Pwn2Own Bug (III)

- By reversing the SM task, we find the handlers for the different messages.
- One of these messages is the ACTIVATE PDP CONTEXT ACCEPT message.
- One part of it that seems to be interesting is the Protocol Configuration Options, the function processing that IE seems complicated.

# Hunting for Bugs

## The Mobile Pwn2Own Bug (IV)

```
signed int __fastcall sm_ProcessProtConfigOpts(unsigned __int8 *buf, unsigned int bufsize, unsigned __int8 *a3)
{
  ...
  unsigned __int8 v62[16]; // [sp+8h] [bp-58h]
  ...
  while ( idx < bufsize_ )
  {
    cursord = &buf__[idx];
    v10 = idx + 2;
    v11 = buf__[v10];
    idx = v10 + 1;
    proto = (unsigned __int16)(cursord[1] + (*cursord << 8));
    ...
    v61 = v13;
    ...
    if ( v11 )
    {
      if ( proto == 0x8021 )
      {
        subprot = buf__[idx];
        nidx = idx + 2;
        if ( subprot == 2 || subprot == 3 || subprot == 4 || subprot == 1 )
        {
          nptr = &buf__[nidx];
          idx = nidx + 2;
          v18 = (unsigned __int16)(nptr[1] + (*nptr << 8));
          if ( v18 >= 4 )
          {
            v19 = v18 - 4;
            while ( 2 )
            {
              v28 = v19;
              while ( 1 )
              {
                if ( !v28 )
                  goto LABEL_217;
                v20 = buf__[idx];
                v21 = idx + 1;
                if ( v20 != 3 )
                  break;
                v23 = buf__[v21];
                idx = v21 + 1;
                v24 = v23;
                ...
                if ( !len || len > 0x10 )
                {
                  if ( byte_41695FA4 )
                  {
                    v60 = 1108168344;
                    v61 = ((unsigned __int8)byte_41695FA4 << 18) + 0x40521;
                    v25 = len;
                  }
                  else
                  {
                    v25 = len;
                    v60 = 1108168372;
                    v61 = 0x40521;
                  }
                  DbgRelatedFcn_0(&v60, v25, 0xFECDBA98);
                }
                for ( i = 0; i < (signed int)(v24 - 2); i = (i + 1) & 0xFF )
                {
                  c = buf__[idx++];
                  v62[i] = c;
                }
                ...
          }
        }
    ...
}
```

# Hunting for Bugs

The Mobile Pwn2Own Bug (V)

- Processes Protocol Configuration Options which are sent by the Network in a \`ACTIVATE PDP CONTEXT ACCEPT\`.
- PDP stands for Packet Data Protocol
- The purpose of the protocol configuration options information element is to:
  - transfer external network protocol options associated with a PDP context activation, and
  - transfer additional (protocol) data (e.g. configuration parameters, error codes or messages/events) associated with an external protocol or an application.

## The Mobile Pwn2Own Bug (VI)

- One of the supported protocols is **IPCP** (Internet Protocol Control Protocol).

**IPCP header:**

| *Offsets* | Octet | 0 | | | | | | | | 1 | | | | | | | | 2 | | | | | | | | 3 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Octet** | **Bit** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** | **16** | **17** | **18** | **19** | **20** | **21** | **22** | **23** | **24** | **25** | **26** | **27** | **28** | **29** | **30** | **31** |
| **0** | **0** | Code | | | | | | | | Identifier | | | | | | | | Length | | | | | | | | | | | | | | | |

**Code.**

8 bits.
Specifies the function to be performed.

| Code | Description | References |
|---|---|---|
| **0** | Vendor Specific. | RFC 2153 |
| **1** | Configure-Request. | |
| **2** | Configure-Ack. | |
| **3** | Configure-Nak. | |
| **4** | Configure-Reject. | |
| **5** | Terminate-Request. | |
| **6** | Terminate-Ack. | |
| **7** | Code-Reject. | |

# Hunting for Bugs

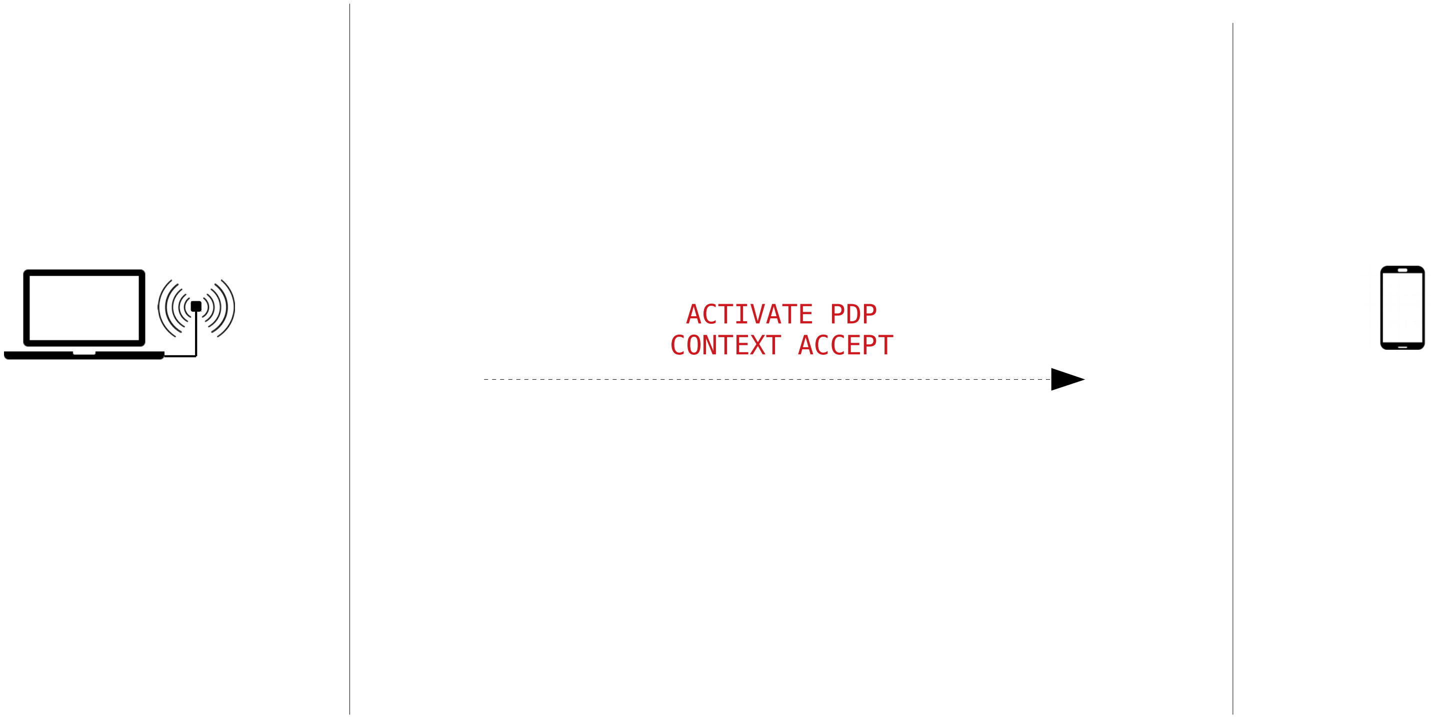
## The Mobile Pwn2Own Bug (VII)

```
int sm_ProcessProtConfigOpts(unsigned __int8 *buf, unsigned int bufsize, unsigned __int8 *a3)
{
    unsigned __int8 tbuf[16];
    int idx;
    int len;
    unsigned __int8 c;


    ...

    len = buf[idx++];
    ...
    if ( !len || len > 0x10 )
    {
        // Do nothing about it
    }
    for ( i = 0; i < len - 2; i = (i + 1) & 0xFF )
    {
      c = buf[idx++];
      tbuf[i] = c;
    }


    ...
}
```

# Hunting for Bugs

## The Mobile Pwn2Own Bug (VIII)

- The plan looks like this:



# Hunting for Bugs

## The Mobile Pwn2Own Bug (IX)

- Not so easy...
- Problem is the phone will only process this message if it is in the correct state.
- This happens when the phone sends a `ACTIVATE PDP CONTEXT REQUEST` message.
- Which in turn happens if the phone is manually configured to include an APN in the connection settings.
- However this is a problem for the P20...

# Hunting for Bugs

The Mobile Pwn2Own Bug (X)

- Read more of the technical standards...
- We can force the MS get in the correct state (i.e perform PDP activation procedure) by sending a `REQUEST PDP CONTEXT ACTIVATION`.

## 9.5.7 Request PDP context activation

This message is sent by the network to the MS to initiate activation of a PDP context. See table 9.5.7/3GPP TS 24.008.

Message type: REQUEST PDP CONTEXT ACTIVATION

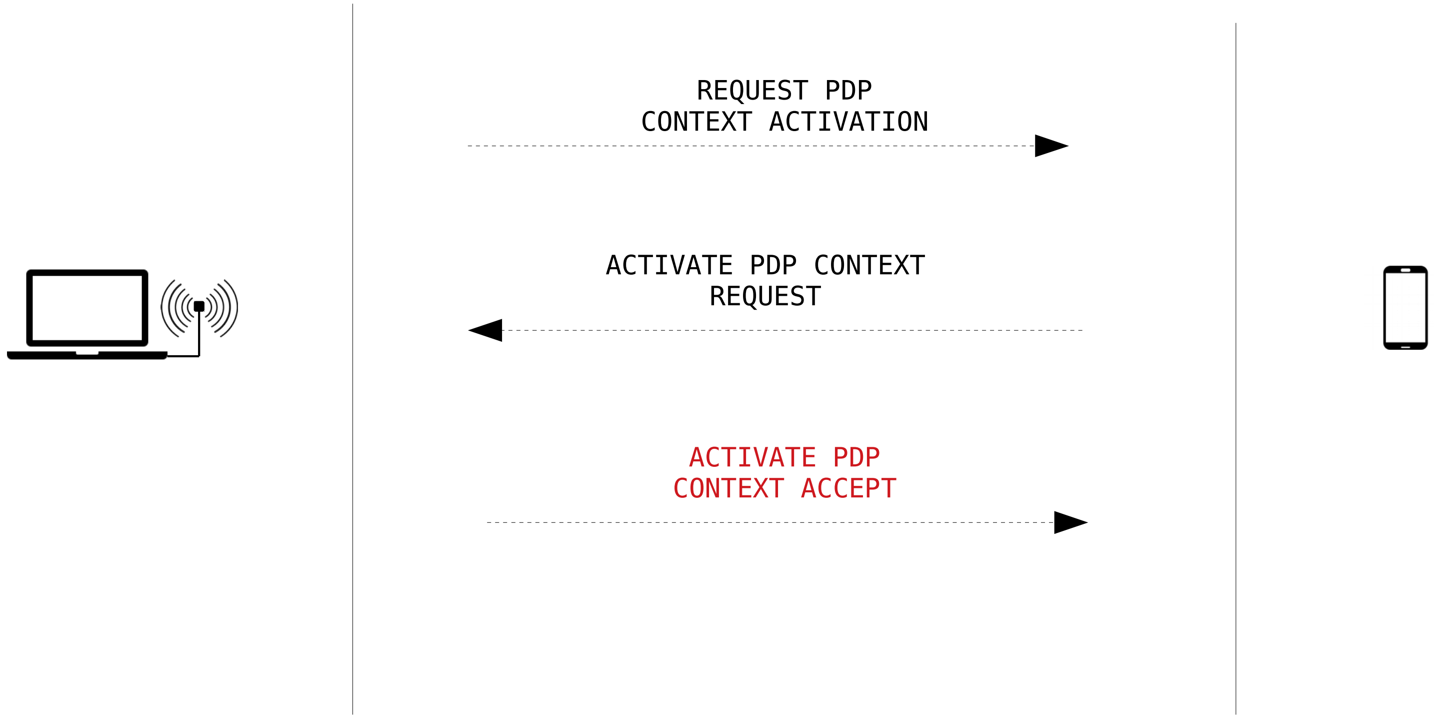
Significance: global

Direction: network to MS

# Hunting for Bugs

The Mobile Pwn2Own Bug (XI)

- The actual plan looks like this:



# Hunting for Bugs

## The Mobile Pwn2Own Bug (XII)

- In order to actually implement the attack we need to modify the source code of *YateBTS*.
- Add code to send the `REQUEST PDP CONTEXT ACTIVATION` message to the phone.
- Modify the `ACTIVATE PDP CONTEXT ACCEPT` messages to trigger the bug.
- As said earlier the code is pretty clean and actually reading it will help you better understand the GSM protocol stack.
- For this attack, the file to modify is: mbts/SGSNGGSN/Ggsn.cpp

# Hunting for Bugs

## The Mobile Pwn2Own Bug (XIII)

- ROP is needed for the first stage of your payload due to ARM cache-fu.
- Copy shellcode to some arbitrary RWX address and invalidate/flush the i-cache/d-cache.
- Jump to win.
- Payload can do any number of things, for P2O I chose to write to the Android filesystem by leveraging the **RFS** (Remote? File System), a mechanism which allows the baseband to store data such as NV Items to the android filesystem.
- Payload can even be a custom “debugger” that can be used to find other bugs and write more involved exploits (e.g heap memory corruption).

**Demo**

# Conclusions

# Conclusions

- Baseband exploitation isn’t as hard as it is percieved to be.
- You don’t need to know much about cellular networks in order to exploit them.
- When will we see the first full remote compromise through baseband ?
- Many targets out there, Huawei, Intel, Qualcomm...

?’s