Modmobtools

Internals, updates and more

By Sébastien Dudek

Troopers - Telco Sec Day

March 19th 2019
About me

- Sébastien Dudek (@FlUxIuS)
- Working at Synacktiv: pentests, red team, audits, vuln researches
- Likes radio and hardware
- And to confront theory vs. practice
Introduction

- Pentesting mobile devices (phones, intercoms, connected cars, ...) → right tools
- Data exchanged: (IoT) devices ↔ server are generally trusted
- Spawn a fake station → OpenBTS/OsmoBTS, OpenBTS-UMTS, srsLTE, Amarisoft...
- But we need also to attract the device to this station
- Also sometimes it’s needed to perform cell monitoring on 2G/3G/4G and soon in 5G.
  → we developed some cool & cheap tools to do that!
Our tools

- Modmobmap: monitoring 2G/3G/4G cells and more
- Modmobjam: smart/targeted jamming tools
Where can I use this tool?

**Cell towers discovery**
- have a list and description of surrounding towers
- spot rogue base stations (mature list required!)

**Jamming**
Where can I use this tool?

Cell towers discovery

Jamming

- replace the noisy chineese jammer
- avoid commercial jamming device reworking (bands disabling)
Remember: monitoring with holy relics

Old Nokia phone have a net monitor mode that could be enabled via FBus or MBUS access.

Tools

- Gnokii, Gammu and others: activate monitor mode, interact with the phone, and capture trace logs.
- DCT3-GSMTAP: evolution of Gammu, capture of GSM Um and SIM-ME via GSMTAP pseudo-header format.
Thing that exists

OpenCellID example

Very few information... could be used as a database for spotting rogue base stations. But useless for jamming attacks
Thing we wanna do for 3G, 4G and more

<table>
<thead>
<tr>
<th>ARFCN</th>
<th>MCC</th>
<th>MNC</th>
<th>LAC</th>
<th>cell ID</th>
<th>forb.LA</th>
<th>prio</th>
<th>min-db</th>
<th>max-pwr</th>
<th>rx-lev</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>208</td>
<td>01</td>
<td>0x</td>
<td>0xe</td>
<td>n/a</td>
<td>n/a</td>
<td>-110</td>
<td>5</td>
<td>-71</td>
</tr>
<tr>
<td>3</td>
<td>208</td>
<td>01</td>
<td>0x</td>
<td>0xb</td>
<td>n/a</td>
<td>n/a</td>
<td>-110</td>
<td>5</td>
<td>-76</td>
</tr>
<tr>
<td>7</td>
<td>208</td>
<td>01</td>
<td>0x</td>
<td>0xa</td>
<td>n/a</td>
<td>n/a</td>
<td>-110</td>
<td>5</td>
<td>-74</td>
</tr>
<tr>
<td>11</td>
<td>208</td>
<td>01</td>
<td>0x</td>
<td>0xe</td>
<td>n/a</td>
<td>n/a</td>
<td>-110</td>
<td>5</td>
<td>-75</td>
</tr>
<tr>
<td>77</td>
<td>208</td>
<td>10</td>
<td>0x</td>
<td>0x9</td>
<td>no</td>
<td>normal</td>
<td>-105</td>
<td>5</td>
<td>-84</td>
</tr>
<tr>
<td>513DCS</td>
<td>208</td>
<td>01</td>
<td>0x</td>
<td>0xd</td>
<td>n/a</td>
<td>n/a</td>
<td>-95</td>
<td>0</td>
<td>-82</td>
</tr>
<tr>
<td>518DCS</td>
<td>208</td>
<td>01</td>
<td>0x</td>
<td>0x5</td>
<td>n/a</td>
<td>n/a</td>
<td>-95</td>
<td>0</td>
<td>-79</td>
</tr>
<tr>
<td>609DCS</td>
<td>208</td>
<td>01</td>
<td>0x</td>
<td>0xf</td>
<td>n/a</td>
<td>n/a</td>
<td>-95</td>
<td>0</td>
<td>-70</td>
</tr>
<tr>
<td>744DCS</td>
<td>208</td>
<td>10</td>
<td>0x</td>
<td>0xe</td>
<td>n/a</td>
<td>n/a</td>
<td>-95</td>
<td>0</td>
<td>-91</td>
</tr>
<tr>
<td>976</td>
<td>208</td>
<td>20</td>
<td>0x</td>
<td>0xc</td>
<td>n/a</td>
<td>n/a</td>
<td>-104</td>
<td>5</td>
<td>-81</td>
</tr>
<tr>
<td>978</td>
<td>208</td>
<td>20</td>
<td>0x</td>
<td>0xc</td>
<td>n/a</td>
<td>n/a</td>
<td>-104</td>
<td>5</td>
<td>-79</td>
</tr>
<tr>
<td>979</td>
<td>208</td>
<td>20</td>
<td>0x</td>
<td>0x0</td>
<td>n/a</td>
<td>n/a</td>
<td>-104</td>
<td>5</td>
<td>-84</td>
</tr>
<tr>
<td>982</td>
<td>208</td>
<td>20</td>
<td>0x</td>
<td>0xc</td>
<td>n/a</td>
<td>n/a</td>
<td>-104</td>
<td>5</td>
<td>-74</td>
</tr>
<tr>
<td>984</td>
<td>208</td>
<td>20</td>
<td>0x</td>
<td>0xc</td>
<td>n/a</td>
<td>n/a</td>
<td>-104</td>
<td>5</td>
<td>-57</td>
</tr>
<tr>
<td>986</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>1011</td>
<td>208</td>
<td>20</td>
<td>0x</td>
<td>0x9</td>
<td>n/a</td>
<td>n/a</td>
<td>-104</td>
<td>5</td>
<td>-87</td>
</tr>
<tr>
<td>1012</td>
<td>208</td>
<td>20</td>
<td>0x</td>
<td>0xb</td>
<td>n/a</td>
<td>n/a</td>
<td>-104</td>
<td>5</td>
<td>-84</td>
</tr>
</tbody>
</table>

OsmocomBB cell monitor
Public tools

Recorded mobile towers

- OpenCellid: Open Database of Cell Towers
- Gsmmap.org
- and so on.

Live scanning tools
Public tools

Recorded mobile towers
- OpenCellid: Open Database of Cell Towers
- Gsmmap.org
- and so on.

Problem!
But these solutions don’t map in live and do not give precise information about cell towers.

Live scanning tools
Public tools

Recorded mobile towers

Live scanning tools

- for 2G cells:
  - Gammu/Wammu, DCT3-GSMTAP, and others
  - OsmocomBB via `cell_log` application

- for 3G, 4G and more:
  - only tricks: use of exposed DIAG interface → decoding → GSMTAP pseudo-header format
  - SnoopSnitch: could be reworked for our purposes ;)


Methods to capture cells information

Possible methods are:

- Software-Defined Radio
- Exposed diagnostic interfaces
- Use of Android RIL
Software-Defined Radio

Existing tools:
- Airprobe or GR-GSM
- OpenLTE: *LTE_fdd_dl_scan*
- srsLTE with srsUE
Software-Defined Radio

Existing tools:

- Airprobe or GR-GSM
- OpenLTE: \textit{LTE\_fdd\_dl\_scan}
- srsLTE with srsUE

\textbf{No 3G}

No 3G tools to capture cell information.
Exposed diagnostic interface

- Diagnostic interface enabled:
  - On old phones and 3G sticks like the Icon 255\(^1\) that expose it by default
  - enabling DIAG ourselves: e.g for some LG devices via 
    /sys/devices/platform/lg_diag_cmd/diag_enable
  - Chips used for development
  - Interfaces kept enabled in production by error (e.g via custom bootmodes → CVE-2016-8467)

- Existing tools:
  - xgoldmon for X-Gold Infineon Basebands
  - diag-parser for exposed Qualcomm DIAG interfaces

\(^1\)https://events.ccc.de/congress/2011/Fahrplan/attachments/2022_11ccc-qcommbdbbg.pdf
Making a development environment

- Good alternative
- Could work with almost all bands we want
- A little expensive: almost 300€

Requirements:

- EC20 LTE modem
- PCengines APU2
Supertramp’s version

- U/EC20 3G/LTE modem
- mPCI-E adapter
(Funny story about EC20)

- Seen at 33c3 by Harald Welte\(^2\) → the modem runs an OE base Linux distribution

- It’s also possible to have a shell via the AT command

\[ \text{AT+QLINUXCMD} : \]

```bash
# echo -e 'AT+QLINUXCMD="/sbin/getty -L ttyGS0 115200 console"\r\n' > /dev/ttyUSB2
# microcom /dev/ttyUSB1

OpenEmbedded Linux 9615—cdp ttyGS0

msm 20160923 9615—cdp ttyGS0

9615—cdp login: root
Password: oelinux123
root@9615—cdp:~#
```

\(^2\)http://git.gnumonks.org/laforg.slides/plain/2016/cellular_modems_33c3/33c3modems.html
RIL on Android

- Daemon forwards commands/messages: application ⇔ Vendor RIL
- Vendor library is proprietary and vendor specific
- Vendor library knows how to talk to modem:
  - classic AT
  - QMI for Qualcomm
  - (old?) Samsung IPC Protocol
  - and so on.
# ServiceMode on Android

- Usually activated by typing a secret code
- Gives interesting details of current cell:
  - implicit network type
  - used band
  - reception (RX/DL) or/and transmission (TX/UP) (E/U)ARFCN (Absolute Radio Frequency Channel Number)
  - PLMN (Public Land Mobile Network) number
  - and so on.

<table>
<thead>
<tr>
<th>ServiceMode</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>RRC:IDLE, Band:1</td>
<td></td>
</tr>
<tr>
<td>PLMN:208-11</td>
<td></td>
</tr>
<tr>
<td>RX:10762 RI:-84 CID:a21c5</td>
<td></td>
</tr>
<tr>
<td>TX:9812 Eclo:-2 RSCP:-86</td>
<td></td>
</tr>
<tr>
<td>L1:PCH_Sleep PSC:507 DRX:128</td>
<td></td>
</tr>
<tr>
<td>SERVICE : LIMITED</td>
<td></td>
</tr>
<tr>
<td>Speech VER : FR FR FR</td>
<td></td>
</tr>
<tr>
<td>therm: 111 LNA: 0</td>
<td></td>
</tr>
<tr>
<td>SIB19 None</td>
<td></td>
</tr>
<tr>
<td>PA STATE : 0 (APT), HDET : 0</td>
<td></td>
</tr>
<tr>
<td>NETWORK : UNBLOCK</td>
<td></td>
</tr>
<tr>
<td>IMEI Certi: PASS, 1</td>
<td></td>
</tr>
<tr>
<td>Unknown</td>
<td></td>
</tr>
</tbody>
</table>

ServiceMode in Samsung
**Samsung ServiceMode in brief**

1. *#0011#* secret code handled by `ServiceModeApp_RIL ServiceModeApp` activity

2. `ServiceModeApp` → IPC connection → `SecFactoryPhoneTest SecPhoneService`

3. `ServiceModeApp` starts the service mode → `invokeOemRilRequestRaw()` through `SecPhoneService` (send RIL command `RIL_REQUEST_OEM_HOOK_RAW`)

4. `ServiceModeApp` process in higher level ServiceMode messages coming from RIL.

**Best place to listen ServiceMode**

Two good places exist: RIL library independent of Vendor RIL library implementation, or use `invokeOemRilRequestRaw()`
Getting SM messages: the lazy way

Ask to our best friend → logcat

```
shell@lte:/ $ logcat
[...]
I /ServiceModeApp_RIL( 1542): in QUERT_SERVM_DONE
I /ServiceModeApp_RIL( 1542): size of result : 1700
I /ServiceModeApp_RIL( 1542): Line 0 : RRC:IDLE, Band:1_
I /ServiceModeApp_RIL( 1542): Line 1 : PLMN:208−20_
I /ServiceModeApp_RIL( 1542): Line 2 : RX:10639 RI:−70 CID:1fc09bd_
I /ServiceModeApp_RIL( 1542): Line 3 : TX:9689 EvoC:−4 RSCP:−74_
I /ServiceModeApp_RIL( 1542): Line 4 : L1:PCH_Sleep PSC:83 DRX:64_
I /ServiceModeApp_RIL( 1542): Line 5 : SERVICE : LIMITED_
I /ServiceModeApp_RIL( 1542): Line 7 : therm: 111 LNA: 0_
I /ServiceModeApp_RIL( 1542): Line 8 : SIB19 Received_
I /ServiceModeApp_RIL( 1542): Line 9 : PA STATE : 0 (APT), HDET : 0_
I /ServiceModeApp_RIL( 1542): Line 10 : NETWORK : UNBLOCK_
I /ServiceModeApp_RIL( 1542): Line 11 : IMEI Certi: PASS, 1_
```

Those messages could be then processed to get our current cell information.
Getting data from DIAG with Xgoldmon

We have reworked Xgoldmon project for that:

- [https://github.com/FlUxIuS/xgoldmon](https://github.com/FlUxIuS/xgoldmon)

```
$ cat ./celllog.fifo
[...]
[CellInfo]:PLMN=208−15;RAC=0x1;LAC=0x4e71;CID=0x1f****;DL_UARFCN=10737;UL_ARFCN=9787
[CellInfo]:PLMN=208−20;RAC=0x1;LAC=0x4e71;CID=0x1f****;DL_UARFCN=2950;UL_ARFCN=2725
[...]
[CellInfo]:PLMN=208−20;RAC=0x1;LAC=0xb5aa;CID=0x97****;DL_UARFCN=10639;UL_ARFCN=9689
[CellInfo]:PLMN=208−10;RAC=0x1;LAC=0xb5aa;CID=0x97****;DL_UARFCN=65535;UL_ARFCN=2850
[...]
```
What do I need?

At least a phone supporting ServiceMode!

- At least supports following tested phones:
  - Samsung Galaxy S3 via xgoldmon (Modmobmap’s edition);
  - Samsung Galaxy S4;
  - Samsung Galaxy S5;
  - Samsung Galaxy Note 2 with LTE;
  - Samsung Galaxy S4 GT-I9500
  - Samsung Galaxy Nexus GT-I9250
  - Samsung Galaxy S2 GT-I9100
  - Samsung Galaxy Note 2 GT-N7100
  - Samsung Galaxy S6 Exynos SoC
  - Samsung Galaxy S7 Exynos SoC
  - Samsung Galaxy A3 Exynos SoC
  - ...

“KTHX! But there are 2 questions”:

1. how to support other operators than your own SIM card?
2. how to enumerate cells a MS (Mobile Station) is supposed to see?
Few contraints to resolve

“KTHX! But there are 2 questions”:

1. how to support other operators than your own SIM card?
2. how to enumerate cells a MS (Mobile Station) is supposed to see?

Answer
The DFR technique!
DFR technique

D.F.R: “D” for Dirty, “F” for Fuzzy, “R” for Registration
The camping concept in brief

Let’s remember 3GPP TS 43.022, ETSI TS 125 304...

- When selecting a PLMN → MS looks for cells satisfying few conditions (cell of the selected PLMN, not barred, pathloss between MS and BTS below a threshold, and so on.)
- Cells are checked in a descending order of the signal strength
- If a suitable is found → MS camps on it and tries to register
The camping concept in brief

Let’s remember 3GPP TS 43.022, ETSI TS 125 304...

- When selecting a PLMN → MS looks for cells satisfying few conditions (cell of the selected PLMN, not barred, pathloss between MS and BTS below a threshold, and so on.)
- Cells are checked in a descending order of the signal strength
- If a suitable is found → MS camps on it and tries to register

**Verified through DIAG and ServiceMode**

If registration fails → MS camps to another cell until it can register → verified via DIAG and ServiceMode
Automate the DFR technique with AT commands

Android phones often expose a modem interface (e.g. /dev/smd0)

```
127|shell@klte:/ $ getprop rild.libargs
   - d /dev/smd0
```

It is possible to:

- set network type: AT^SYSCONFIG
- list PLNM and select a PLMN: AT+COPS
  
requires root privileges
We mix all techniques together
Don’t forget...

*the magic cure powder
Here is the frankenstein: Modmobmap
In brief

- Uses Modmobmap results to jam mobile cells in a DIY way!
- Cheapest and efficient tricks to jam
Before

With a portable/chinese device

- cheap
- jam the whole 2G/3G/(4G?) bands but requires some modifications
- poor signal

Desktop jammers
Before

With a portable/chinese device

Desktop jammers
- heavy, cumbersome but powerful
- also needs a disabling to conserve rogue cells
Software-Defined Radio way

- With Software-Defined Radio
- Many devices could be used even the cheapest:
  - bladeRF;
  - HackRF;
  - ADALM-PLUTO;
  - and so on.
Software-Defined Radio way

- With Software-Defined Radio
- Many devices could be used even the cheapest:
  - bladeRF;
  - HackRF;
  - ADALM-PLUTO;
  - and so on.

The bandwidth
KTHX! But how do you cover all frequencies with your toys bro?
## SDR specs

<table>
<thead>
<tr>
<th>Feature</th>
<th>HackRF</th>
<th>bladeRF</th>
<th>USRP B100 Starter</th>
<th>USRP B200</th>
<th>USRP B210</th>
</tr>
</thead>
<tbody>
<tr>
<td>Radio Spectrum</td>
<td>30 MHz – 6 GHz</td>
<td>300 MHz – 3.8 GHz</td>
<td>50 MHz – 2.2 GHz [1]</td>
<td>50 MHz – 6 GHz</td>
<td>61.44 MHz [3]</td>
</tr>
<tr>
<td>Bandwidth</td>
<td>20 MHz</td>
<td>28 MHz</td>
<td>16 MHz [2]</td>
<td>61.44 MHz [3]</td>
<td></td>
</tr>
<tr>
<td>Duplex</td>
<td>Half</td>
<td>Full</td>
<td>Full</td>
<td>Full</td>
<td>2x2 MIMO</td>
</tr>
<tr>
<td>Sample Size (ADC/DAC)</td>
<td>8 bit</td>
<td>12 bit</td>
<td>12 bit / 14 bit</td>
<td>12 bit</td>
<td></td>
</tr>
<tr>
<td>Sample Rate (ADC/DAC)</td>
<td>20 Msp</td>
<td>40 Msp</td>
<td>64 Msp / 128 Msp</td>
<td>61.44 Msp</td>
<td></td>
</tr>
<tr>
<td>Interface (Speed)</td>
<td>USB 2 HS (480 megabit)</td>
<td>USB 3 (5 gigabit)</td>
<td>USB 2 HS (480 megabit)</td>
<td>USB 3 (5 gigabit)</td>
<td></td>
</tr>
<tr>
<td>FPGA Logic Elements</td>
<td>[4]</td>
<td>40k</td>
<td>25k</td>
<td>75k</td>
<td>150k</td>
</tr>
<tr>
<td>Microcontroller</td>
<td>LPC43XX</td>
<td>Cypress FX3</td>
<td>Cypress FX2</td>
<td>Cypress FX3</td>
<td></td>
</tr>
<tr>
<td>Availability</td>
<td>January 2014</td>
<td>Now</td>
<td>Now</td>
<td>Now</td>
<td>Now</td>
</tr>
<tr>
<td>Cost</td>
<td>$300 [6]</td>
<td>$420</td>
<td>$650</td>
<td>$675</td>
<td>$1100</td>
</tr>
</tbody>
</table>

Solution: “Smart” jamming

In 3 steps:

1. scan cells with Modmobmap;
2. target an operator;
3. and jam only targeted channels;
Scanning with Modmobmap

Modmobmap recovers 2G/3G/4G and more cells pretty much like OsmocomBB monitor mode for 2G only.

```
--$ sudo python modmobmap.py -m servicemode
=> Requesting a list of MCC/MNC. Please wait, it may take a while...
[+] New cell detected [CellID/PCI-DL_freq (83-6400)]
  Network type=4G
  PLMN=151515-1515
  Band=20
  Downlink EARFCN=6400
Found 5 operator(s)
{u'20810': u'F SFR', u'20820': u'F-Bouygues Telecom', u'20815': u'Free', u'20801': u'Orange F', u'20811': u'SFR Home 3G'}
[+] Unregistered from current PLMN
[+] New cell detected [CellID/PCI-DL_freq (f0e02-10787)]
  Network type=3G
  PLMN=208-1
  Band=1
  Downlink UARFCN=10787
  Uplink UARFCN=9837
=> Changing MCC/MNC for: 20810
[+] New cell detected [CellID/PCI-DL_freq (298-6400)]
  Network type=4G
  PLMN=208-10
  Band=20
  Downlink EARFCN=6400
[+] New cell detected [CellID/PCI-DL_freq (298-6300)]
  Network type=4G
  PLMN=208-10
  Band=20
  Downlink EARFCN=6300
[+] New cell detected [CellID/PCI-DL_freq (298-6200)]
  Network type=4G
  PLMN=208-10
  Band=20
  Downlink EARFCN=6200
```
Results

JSON file → needed cells information to be reused with other tools, like Modmobjam!

```json
{
   "4b***−76": {
      "PLMN": "208−10",
      "arfcn": 76,
      "cid": "4b**",
      "type": "2G"
   },
   "60****−2950": {
      "PLMN": "208−20",
      "RX": 2950,
      "TX": 2725,
      "cid": 60***,
      "band": 8,
      "type": "3G"
   }
}
```
GnuRadio: playing with blocks

GnuRadio companion is really nice → can add, make, and remove blocks → generates Python code

Perfect to build the bases of our jammer. But we still need an idea of how to design the schema.
After many years of research...

Lot of experiments with blocks != #blockchains... blablabla
The formula

We have finally found THE formula!
Experimentation with GnuRadio

So we’ve started with a simple schema:

But still needed some work...
Final product: Modmobjam
Results with a simple HackRF

Works pretty well when downgrading a call from 3G to 2G

But the number of cells to jam could raise the number of needed SDR devices.
Could also be cheaper using *OsmoFL2k*

**TODO**

Some work is required target specific frequencies → right sample rate, carrier frequency and harmonics + better ant & amp
1. Modmobmap
2. Modmobjam
3. Updates
4. Conclusion
Next updates

- Add RSSI when possible
- Add support of mPCI-E modems with exposed DIAG
- Add more mobile phone supports → based on SCAT tool
- And more! → add also your contribution
Getting data from exposed DIAG on mPCI-E modems

- Just use *diag-parser* tool from Moiji Mobile

The rest could be parser with *pycrate_mobile* library of Benoit Michau \(\rightarrow\) ASN.1 and CSN.1 compilers included for our purposes (RRC, and so on)!
tshark with Wireshark dissectors

But in the train for Troopers, I got lazy:

Launch `diag-parser` and output result in a FIFO file:

```
$ sudo ./diag_parser -g 127.0.0.1 -p /tmp/fifo in -i /dev/ttyUSB0 -vvv
```

and dissect all LTE and UTRA_FDD carrier list:

```
cat /tmp/fifo in| tshark -i -l -n -T json -e gsmtap.arfcn -e lte_rrc
-e lte-rrc.trackingAreaCode -e lte-rrc.cellIdentity -e lte-rrc.q_RxLevMin
-e lte-rrc.freqBandIndicator -e lte-rrc.MCC_MNC_Digit
-e lte-rrc.carrierFreqListUTRA_FDD
-e lte-rrc.carrierFreq -e lte-rrc.interFreqCarrierFreqList -e lte-rrc.dl_CarrierFreq
-e lte-rrc.q_RxLevMin -e lte-rrc.physCellId -Y 'gsmtap.arfcn!=0' > /tmp/fifo out
```
tshark result

tshark gives us a nice JSON render:

```json
{
    "layers": {
        "gsmtap.arfcn": ["6200"],
        "lte_rrc": ["lte_rrc"],
        "lte_rrc.trackingAreaCode": ["75:c2"],
        "lte_rrc.cellIdentity": ["7a:2a:20:80"],
        "lte_rrc.freqBandIndicator": ["20"],
        "lte_rrc.MCC_MNC_Digit": ["2","0","8","2","0"],
        "lte_rrc.q_RxLevMin": ["-61"]
    }
}

[...]

{
    "layers": {
        "gsmtap.arfcn": ["6200"],
        "lte_rrc": ["lte_rrc"],
        "lte_rrc.interFreqCarrierFreqList": ["3"],
        "lte_rrc.dl_CarrierFreq": ["1850","3175","251"],
        "lte_rrc.q_RxLevMin": ["-63","-62","-63"],
        "lte_rrc.physCellId": ["158"]
    }
}

[...]

"lte_rrc.carrierFreq": ["10639","10688","10664","2950"]

[...]
```
DIAG for the rock!

- Less abstracted data
- Carrier lists → catch a bunch of 3G and LTE DL freqs in the same time
- More optimized for mobile monitoring and attacks...
- Support with the tshark JSON output will be committed soon
- Another support with `pycrate_mobile` to parse RRC messages → in the TODO stack!
1 Modmobmap
2 Modmobjam
3 Updates
4 Conclusion
Conclusion

- **Modmobmap:**
  - is a cheap way to scan mobile cells
  - supports 2 useful interfaces:
    - ServiceMode;
    - host DIAG (could be easily extended for guest DIAG);
    - srsLTE and OpenLTE captures soon...

- **Modmobjam:**
  - is a cheap way to jam mobile cells with only a phone and a HackRF
  - but if cells to jam are important more SDR devices are needed
Downloads

- Modmobmap:
  - https://github.com/Synacktiv/Modmobmap
- Modmobjam:
  - https://github.com/Synacktiv/Modmobjam
Thanks =)

- Joffrey Czarny (@_Sn0rkY)
- Priya Chalakkal (@priyachalakkal)
- Troopers staff (@WEareTROOPERS)
- And of course → You all ;)}
ANY QUESTIONS?

THANK YOU FOR YOUR ATTENTION,

SYNACKTIV
DIGITAL SECURITY