Modmob tools and tricks

Using cheap tools and tricks to attack mobile devices in practice

By Sébastien Dudek

Troopers - NGI

March 18th 2019
(update: 19/04/2019)
About me

- Sébastien Dudek (@FIUxluS)
- Working at Synacktiv: pentests, red team, audits, vuln researches
- Likes radio and hardware
- And to confront theory vs. practice
- First time at Troopers =)!
This presentation

- Few reminders:
  - talk about interception techniques in practice
  - existing tools
- Our contribution:
  - feedbacks of our tests (mobile phones, intercoms, cars...)
  - tools we made (Modmobmap and Modmobjam);
  - some cheap tricks;
  - some hardware attacks.

+ meet us tomorrow at Telco Security day → Modmob tools internals, updates, and more! ;)
Introduction

- Mobile network → more than 30 years
  - 1G: analogic, bandwidth depending on the system (30 kHz for AMPS, 25 kHz for TACS, etc.);
  - 2G: FDMA (25 MHz) in combination with TDMA (in Europe);
  - 3G: WCDMA fixed to 5 MHz, 10-20 MHz with carrier aggregation
  - 4G: OFDMA (downlink) and SC-FDMA (uplink), min. 1.4 MHz bandwidth (most common 5 MHz), CA up to 640 MHz (3GPP release 13)

- Evolution of modulation techniques and encoding → better capacity, growth services...

- Current use of the mobile network:
  - intercoms, delivery pick-up stations;
  - electric counters;
  - cameras, cars...
Use of mobile network with intercoms

Pretty the same with connected cars!
5G is coming...

- LTE-A(vanced)++ → 10 Gbps - 100 Gbps theoretically, broader spectrum
- Targets IoT ecosystem
- C-V2X (Vehicle-to-Everything):
  - infrastructures (V2I);
  - networks (V2N);
  - vehicle (V2V);
  - pedestrians (V2P);
  - babies (V2B)?...

source: blog.co-star.co.uk
Security of communications

- 2G, 3G and 4G technologies are more accessible → OpenBTS/OsmoBTS/YateBTS, OpenBTS-UMTS, srsLTE, Amarisoft LTE, ...

- Publications exist on A5/1 about weaknesses

- GPRS, 3G and 4G use stronger ciphering algorithms:
  - KASUMI (UEA-1 algorithm);
  - Snow-3G (UE-2), second algorithm for UMTS and used for LTE (128-EEA1);
  - AES 128 bits (128-EEA2) in addition to Snow-3G for LTE.
## Security of communications (2)

<table>
<thead>
<tr>
<th></th>
<th>GSM</th>
<th>3G</th>
<th>4G</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Client authentication</strong></td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td><strong>Network authentication</strong></td>
<td>NO</td>
<td>Only if USIM is used (not SIM)</td>
<td>YES</td>
</tr>
<tr>
<td><strong>Signaling integrity</strong></td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td><strong>Encryption</strong></td>
<td>A5/1</td>
<td>KASUMI</td>
<td>SNOW-3G</td>
</tr>
</tbody>
</table>

→ Exception exist depending on baseband implementation
Targets in GPRS, UMTS and LTE exchanged data

IP $\rightarrow$ handled by Packet Data Convergence Protocol...

source: what-when-how.com
1. Requirements

2. Attracting mobile devices

3. Capturing mobile data of a famous intercom in France

4. Hard way

5. Other interesting targets

6. Other interesting targets

7. The future

8. Conclusion
Software-Defined radio

To interface to devices using the mobile network:

<table>
<thead>
<tr>
<th>Peripheral</th>
<th>Frequency</th>
<th>Max. Sampling CAN/CNA (rate, width)</th>
<th>Supported software</th>
<th>Frequency stability</th>
<th>TX/RX Channels</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>USRP B2x0</td>
<td>70 MHz - 6 GHz</td>
<td>61.44 Msps, 12 bits</td>
<td>2G: OpenBTS and OsmoTRX</td>
<td>±2 ppm without GPSDO</td>
<td>- B200: 1 Tx + 1 Rx</td>
<td>~800€ min.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3G: OpenBTS-UMTS</td>
<td></td>
<td>- B210: 2 Tx + 2 Rx</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4G: srsLTE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5G: OpenAirInterface</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BladeRF 1.x</td>
<td>300 MHz – 3.8 GHz</td>
<td>40 Msps, 12 bits</td>
<td>2G: YateBTS</td>
<td>±1 ppm</td>
<td>1 Tx + 1 Rx</td>
<td>~400€ min.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4G: srsLTE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5G: OpenAirInterface</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LimeSDR</td>
<td>100 kHz-3.8 GHz</td>
<td>61.44 Msps, 12 bits</td>
<td>2G: OpenBTS with OsmoTRX</td>
<td>±2.5 ppm</td>
<td>2 Tx + 2 Rx</td>
<td>~300€ min.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4G: srsLTE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5G: OpenAirInterface</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XTRX</td>
<td>30 MHz - 3.7 GHz</td>
<td>120 Msps SISO / 90 Mss MIMO, 12 bits</td>
<td>2G: OpenBTS with OsmoTRX</td>
<td>± 0.5 ppm with GPS / ± 0.01 ppm with GPS lock</td>
<td>2 Tx + 2 Rx</td>
<td>~260€ min.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4G: srsLTE</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Alternatives

- sysmoBTS for GSM and GPRS
- sysmoNITB for 3G/LTE → requires a custom/vulnerable femtocell
- LTE LabKit by Yate for LTE;
- Amarisoft LTE → relevant and, as a great core network implementation and includes Cat-NB1/NB2 and others...
- commercial version of srsLTE including Cat-NB1
- specialised equipments like CMU200 → helped some researchers to find vulns in CDMA baseband stacks ;)

SYNACKTIV DIGITAL SECURITY
Set-up: architecture example with bladeRF

Alternative: a limeSDR mini + osmoBTS (and other osmo* components) for almost 100€ min.
Enabling GPRS on YateBTS

As explained on YateBTS Wiki: edit the *ybts.conf* file

```plaintext
[gprs]
Enable=yes
```

for NGI invitation and information And configure the Gateway GPRS Support Node section to handle exchange: GPRS ↔ Internet

```plaintext
[ggsn]
DNS=8.8.8.8 8.8.4.4 ; its preferable to use your own servers for client side attacks
IP.MaxPacketSize=1520
IP.ReuseTimeout=180
IP.TossDuplicatePackets=no
LogFile.Name=/tmp/sgsn.log
MS.IP.Base=192.168.99.1
MS.IP.MaxCount=254
TunName=sgsntun
```

Testing it

Don’t forget to forward traffic from the internal network:

```
# echo 1 > /proc/sys/net/ipv4/ip_forward
# iptables -A POSTROUTING -t nat -s 192.168.99.0/24 ! -d 192.168.99.0/24 -j MASQUERADE
```

And we are connected in GPRS (using a Nexus 5X phone):

Dites "Ok Google".
1 Requirements

2 Attracting mobile devices

3 Capturing mobile data of a famous intercom in France

4 Hard way

5 Other interesting targets

6 Other interesting targets

7 The future

8 Conclusion
Possible ways

- Mobile devices always look for better signal reception
- Generally there is > 1 mobile stack
- Few tricks to consider:
  - use of custom (U)SIM card;
  - Faraday shield isolation;
  - downgrade attacks;

We’ll see how to revisit it with cheap equipments + some style ;)

SYNACKTIV
Method 1: Custom SIM/USIM cards

- Prepaid SIM/USIM card in some cases
- Or custom SIM/USIM card from sysmocom for example

→ Make the fake BTS/(e)NodeB act as a legit BTS
Method 1: Custom SIM/USIM cards

- Prepaid SIM/USIM card in some cases
- Or custom SIM/USIM card from sysmocom for example

→ Make the fake BTS/(e)NodeB act as a legit BTS

Caution

Beware with PIN auto-typing → use a SIMtrace tool to get the typed PIN
Program sysmoUSIM cards

- Could be entirely configured → PySIM and sysmo-usim-utils
- Configure secrets:
  - Ki (subscriber key);
  - OP/c (Operator Variant Algorithm Configuration field);
  - and MCC/MNC to avoid roaming forcing on the User Equipment (UE).

```
$ sudo python pySim-prog.py -p0 -t sysmoUSIM-SJS1 -a 9017000000 ***** -x 001 -y 01 -i 89882110000002 ***** ...
> Ki : 6abb9ae663f9889eddaae298cdcb4ec6
> OPC : 074a3a73ed3c54e1960e9e5732ff35b1
> ACC : None
```
SIMtrace for the rescue

Sniff auto-typed PINs with the Osmocom SIMtrace:

Intercom USIM card
Method 2: Faraday cage

- Mostly cumbersome and expensive
- But could be improvised considering several elements:
  - Frequency;
  - Wavelength;
  - Power of reception or transmission;
  - Distance between the receiver and the transmitter.
- Cage with meshes $\rightarrow$ optimised windows against reflection of the electric field
- Shielding boxes attenuate the signal quietly good!
Practical shielding box for us:  
1 Kg M&Ms box

Can feat small devices as well as a bladeRF, or limeSDR
Space optimisation

We can use antenna extenders to avoid to put entire devices...
Final set-up

And fill holes with an aluminum foil tape...
Method 3: Downgrade attacks

- Use a cheap 2G/3G/4G jammer and rework it
- Or perform smart-jamming:
  1. monitor and collect cells data
  2. jam precise frequencies from collected cells → choose few target operators
Monitoring: State of the Art

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

Live scanning tools
Monitoring: State of the Art

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
Monitoring: State of the Art

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: not reflexible, but could be reworked for our purposes ;)

SnoopSnitch: not reflexible, but could be reworked for our purposes ;)

SnoopSnitch: not reflexible, but could be reworked for our purposes ;)

SnoopSnitch: not reflexible, but could be reworked for our purposes ;)

SnoopSnitch: not reflexible, but could be reworked for our purposes ;)

SnoopSnitch: not reflexible, but could be reworked for our purposes ;)

SnoopSnitch: not reflexible, but could be reworked for our purposes ;)

SnoopSnitch: not reflexible, but could be reworked for our purposes ;)

SnoopSnitch: not reflexible, but could be reworked for our purposes ;)

SnoopSnitch: not reflexible, but could be reworked for our purposes ;)

SnoopSnitch: not reflexible, but could be reworked for our purposes ;)

SnoopSnitch: not reflexible, but could be reworked for our purposes ;)

SnoopSnitch: not reflexible, but could be reworked for our purposes ;)

SnoopSnitch: not reflexible, but could be reworked for our purposes ;)

SnoopSnitch: not reflexible, but could be reworked for our purposes ;)

SnoopSnitch: not reflexible, but could be reworked for our purposes ;)

SnoopSnitch: not reflexible, but 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: LTE_fdd_dl_scan
- srsLTE with srsUE

No 3G

No 3G tools to capture cell information.
Exposed DIAG interfaces

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

requirements:

U/EC20 3G/LTE modem
PCengines APU2
Cheaper way

- U/EC20 3G/LTE modem
- And an adapter with (U)SIM slot
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
  - 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 transmission (TX/UP) (E/U) ARFCN (Absolute Radio Frequency Channel Number)
  - PLMN (Public Land Mobile Network) number
  - and so on.

ServiceMode in Samsung

- RRC: IDLE, Band: 1
- PLMN: 208-11
- RX: 10762 RI: -84 CID: a21c5
- TX: 9812 EcIo: -2 RSCP: -86
- L1: PCH_Sleep PSC: 507 DRX: 128
- SERVICE: LIMITED
- Speech VER: FR FR FR
- therm: 111 LNA: 0
- SIB19 None
- PA STATE: 0 (APT), HDET: 0
- NETWORK: UNBLOCK
- IMEI Certi: PASS, 1
- Unknown
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()
Few constraints to resolve

1. How to support other operators than your own SIM card?
2. How to enumerate cells a MS (Mobile Station) is supposed to see?
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 cell changes with AT commands

Android phones often expose a modem interface (e.g. /dev/smd0), but could also be exposed in the host with few configurations

```
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 if it is performed in the phone
Modmobmap: the monster we have created

We implemented interesting techniques in a tool we called "Modmobmap" (reminds some tasty korean dish)
Monitoring 2G/3G/4G cells

Using Modmobmap:

```
$ sudo python modmobmap.py -m servicemode -s <Android SDK path>
=> Requesting a list of MCC/MNC. Please wait, it may take a while...
[+] New cell detected [CellID/PCI–DL_freq (XXXXXXXX)]
  Network type=2G
  PLMN=208–20
  ARFCN=1014
  Found 3 operator(s)
  {u'20810': u'F SFR', u'20820': u'F–Bouygues Telecom', u'20801': u'Orange F'}
[+] Unregistered from current PLMN
=> Changing MCC/MNC for: 20810
[+] New cell detected [CellID/PCI–DL_freq (XXXXXXXX)]
  Network type=2G
  PLMN=208–20
  ARFCN=76
  [...]  
[+] New cell detected [CellID/PCI–DL_freq (XXXXXXXX)]
  Network type=3G
  PLMN=208–1
  Band=8
  Downlink UARFCN=3011
  Uplink UARFCN=2786
  [...]  
[+] Cells save as cells_1536076848.json # with an CTRL+C interrupt
```
Results of Modmobmap

The script produces a JSON file you can use with your own tools:

```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"
    }
}...
```

→ but we’ll see how it could be used for Jamming purposes!
Jamming in general

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

Desktop jammers
Jamming in general

With a portable/chinese device

Desktop jammers

- heavy, cumbersome but powerful
- also needs a disabling to conserve rogue cells’ band
"Smart" jamming

- Jam only targeted cells
- Stealth against monitors
- In 3 steps:
  1. scan cells with Modmobmap;
  2. target an operator;
  3. and jam only targeted channels;

We have also made a tool for that! → Modmobjam → use Software-Defined radio
"Smart" jamming

- Jam only targeted cells
- Stealth against monitors
- In 3 steps:
  1. scan cells with Modmobmap;
  2. target an operator;
  3. and jam only targeted channels;

We have also made a tool for that! → Modmobjam → use Software-Defined radio

Forbidden

Do it at your own risks and adjust settings to the targeted parameter only. The same should also be done with you fake BTS.
# Jamming with Modmobjam

Jamming with Modmobjam

---

<table>
<thead>
<tr>
<th>Sample rate</th>
<th>1.048</th>
<th>1.048</th>
</tr>
</thead>
<tbody>
<tr>
<td>RF gain</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>RF gain</td>
<td>35</td>
<td>35</td>
</tr>
<tr>
<td>Freq</td>
<td>1.815G</td>
<td>1.815G</td>
</tr>
<tr>
<td>Bandwidth</td>
<td>2OM</td>
<td>2OM</td>
</tr>
</tbody>
</table>

---

```
sudo python smartjam rpc.py -f cells 3528307980.jsm
```

---

```
127.0.0.1 - [7/7/2018 16:11:22] POST /RPC2 HTTP/1.1 200
```

---

![Graphical representation of Modmobjam setup](image.png)

---

```
sudo python smartjam rpc.py -f cells 3528307980.jsm
```

---

```
127.0.0.1 - [7/7/2018 16:11:19] POST /RPC2 HTTP/1.1 200
```

---

**Important:**

- **Sample Rate:** 1.048 MHz
- **RF Gain:** 15 dB
- **Frequency:** 1.815 GHz
- **Bandwidth:** 2 MHz

---

**Note:**

- This setup uses a Python script `smartjam` to jam a signal.
- The script takes a `.jsm` file as input.
- The output is a `.py` file for execution.

---

**Interactive GUI:**

- **Options:**
  - **Variable:** RF gain, RF power
  - **Value:** 15, 20

---

**Command Line:**

```
sudo python smartjam rpc.py -f cells 3528307980.jsm
```

---

**Output:**

```
127.0.0.1 - [7/7/2018 16:11:22] POST /RPC2 HTTP/1.1 200
```

---

**Graphical Interface:**

- **GUI Components:**
  - **Kernel:** Boot
  - **Frequency:** 1.815 GHz
  - **Bandwidth:** 2 MHz

---

**Technical Details:**

- **Central Frequency:** 1.815 MHz
- **Bandwidth:** 10 MHz

---

**Security Implications:**

- **Potential Uses:**
  - **Military:** Jamming enemy communications
  - **Civil:** Security and privacy concerns

---

**Further Reading:**

- [Modmobjam Project Website](http://modmobjam.com)
- [Python Script Documentation](http://python-docs.com/smartjam)

---

**Conclusion:**

Jamming with Modmobjam offers a powerful tool for both military and civil applications, highlighting the importance of secure communication systems.
1. Requirements

2. Attracting mobile devices

3. Capturing mobile data of a famous intercom in France

4. Hard way

5. Other interesting targets

6. Other interesting targets

7. The futur

8. conclusion
Analyzing GPRS data

Once we have trapped a device, its IMSI (International Mobile Subscriber Identity) is listed:

```
nipc list registered
IMSI     MSISDN
20801XXXXXXXXXXXX  69691320681
```

Status displayed in SGSN Mobile list:

```
mbts sgsn list
GMM Context: imsi=20801XXXXXXXXXXXX ptmsi=0xd3001 tlli=0xc00d3001 state=
GmmRegisteredNormal age=5 idle=1 MS#1,TLLI=c00d3001,8d402e2e IPs=192.168.99.1
```
Spotting used APNs

Using the GSMTAP interface

<table>
<thead>
<tr>
<th>No.</th>
<th>Source</th>
<th>Destination</th>
<th>Protocol</th>
<th>Length</th>
<th>Time</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>30097</td>
<td>127.0.0.1</td>
<td>127.0.0.1</td>
<td>GSM RLP</td>
<td>111</td>
<td>81.04</td>
<td>410.920355427 GPRS DL:GPRS DL:PACKET_DOWNLINK_DUMMY_CONTROL_B</td>
</tr>
<tr>
<td>30101</td>
<td>127.0.0.1</td>
<td>127.0.0.1</td>
<td>GSM RLP</td>
<td>81.04</td>
<td>81.418</td>
<td>410.935871177 GPRS DL:GPRS DL:PACKET_DOWNLINK_DUMMY_CONTROL_B</td>
</tr>
</tbody>
</table>

Data (50 bytes)

```
Data: 01c09d6a4105030e00000000000000000000000000000000000000000000201...
[Length: 59]
```

Could be interesting to intrude a virtual mobile network with a provided M2M SIM card
Capture exchanges

On the tun interface dedicated to SGSN:

<table>
<thead>
<tr>
<th>Source</th>
<th>Destination</th>
<th>Protocol</th>
<th>Length</th>
<th>Time</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 192.168.99.1</td>
<td>8.8.8.8</td>
<td>DNS</td>
<td>64</td>
<td>0.000000600</td>
<td>Standard query 0x11d8 A gsm..info</td>
</tr>
<tr>
<td>2 8.8.8.8</td>
<td>192.168.99.1</td>
<td>DNS</td>
<td>80</td>
<td>0.037753523</td>
<td>Standard query response 0x11d8 A gsm..info A 91.</td>
</tr>
<tr>
<td>3 192.168.99.1</td>
<td>91.121</td>
<td>TCP</td>
<td>48</td>
<td>0.419114786</td>
<td>80 -&gt; 60001 [SYN] Seq=0 Win=16384 Len=0 MSS=1450 WS=1</td>
</tr>
<tr>
<td>4 91.121</td>
<td>192.168.99.1</td>
<td>TCP</td>
<td>48</td>
<td>0.425593820</td>
<td>60001 -&gt; 80 [SYN, ACK] Seq=6 Ack=1 Win=29200 Len=0 MSS=146</td>
</tr>
<tr>
<td>5 192.168.99.1</td>
<td>91.121</td>
<td>TCP</td>
<td>40</td>
<td>0.855774638</td>
<td>80 -&gt; 60001 [ACK] Seq=1 Ack=1 Win=16384 Len=0</td>
</tr>
<tr>
<td>6 192.168.99.1</td>
<td>91.121</td>
<td>TCP</td>
<td>117</td>
<td>1.120101836</td>
<td>80 -&gt; 60001 [PSH, ACK] Seq=1 Ack=1 Win=16384 Len=77</td>
</tr>
<tr>
<td>7 91.121</td>
<td>192.168.99.1</td>
<td>TCP</td>
<td>40</td>
<td>1.126491129</td>
<td>60001 -&gt; 80 [ACK] Seq=1 Ack=78 Win=29312 Len=0</td>
</tr>
<tr>
<td>8 91.121</td>
<td>192.168.99.1</td>
<td>TCP</td>
<td>60</td>
<td>1.129285661</td>
<td>60001 -&gt; 80 [PSH, ACK] Seq=1 Ack=78 Win=29312 Len=20</td>
</tr>
<tr>
<td>9 91.121</td>
<td>192.168.99.1</td>
<td>TCP</td>
<td>40</td>
<td>1.120573597</td>
<td>60001 -&gt; 80 [FIN, ACK] Seq=21 Ack=78 Win=29312 Len=9</td>
</tr>
<tr>
<td>10 192.168.99.1</td>
<td>91.121</td>
<td>TCP</td>
<td>40</td>
<td>1.637377595</td>
<td>80 -&gt; 60001 [ACK] Seq=70 Ack=21 Win=16364 Len=9</td>
</tr>
<tr>
<td>11 192.168.99.1</td>
<td>91.121</td>
<td>TCP</td>
<td>40</td>
<td>1.608825585</td>
<td>80 -&gt; 60001 [ACK] Seq=78 Ack=22 Win=16384 Len=9</td>
</tr>
<tr>
<td>12 192.168.99.1</td>
<td>91.121</td>
<td>TCP</td>
<td>40</td>
<td>1.722705944</td>
<td>80 -&gt; 60001 [FIN, ACK] Seq=78 Ack=22 Win=16384 Len=9</td>
</tr>
<tr>
<td>13 91.121</td>
<td>192.168.99.1</td>
<td>TCP</td>
<td>40</td>
<td>1.728877651</td>
<td>60001 -&gt; 80 [ACK] Seq=22 Ack=79 Win=29312 Len=9</td>
</tr>
</tbody>
</table>

In that case: two server ports identified → 60001/tcp and 55556/tcp
Talk with one service

We could talk with a sort of synchronisation service on port 6001/tcp:

In [1]: import socket
In [2]: import binascii
In [3]: ip = '91.121.XXX.XXX'
In [4]: port = 60001
In [6]: s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
In [7]: s.connect((ip, port))
In [8]:
s.send(binascii.hexlify("011e4d256360140066000000000000090000011e1540XX[...]"))
Out[8]: 320
In [9]: data = s.recv(1024)
In [10]: data
'

In that case: two server ports identified → 60001/tcp and 55556/tcp
Identification

And could noticed that messages where only identified:
Strange messages

When updating the device: some unknown messages are exchanged on port 55556/tcp
Strange messages (1)

By a naive approach it looked to be encrypted:

```bash
$ ent payload.hex
Entropy = 7.371044 bits per byte.
[...]
```

We have to ook at the firmware to try to decode this message
UMTS interception

- OpenBTS-UMTS could be used
- But doesn’t support authentication and ciphering → SIM mode only can be used

Disabling USIM mode with a sysmoUSIM card:

```
$ sudo python sysmo-usim-tool.sjs1.py -a 772***** -c
[...]
==> USIM application disabled
```

Other alternatives: CMU2000, vulnerable/custom femtocells...
LTE interception

Use of srsLTE → free and stable

Secrets of the SIM should be configured (ex. sysmoUSIM):

- RAND: generated challenge by the HSS (Home Subscriber Server) in the HLR/AuC → generates next authentication vectors
- XRES: result of the challenge/response by the UE
- AUTN: authentication token
- KASME: derivation key of the ciphering and integrity keys
Secrets could be setup in the `user_db.csv` DB of LTE EPC network:

```
# vi /root/.srs/user_db.csv
[...]
ue3,9017000000***** ,b5997ac4a912e9c6216e13951029c674 ,opc,83e5d3f22da411072508f675d2e9e9d9,9001,000000000062,7
```

A good configuration should result as follows:

```
[...]
UE Authentication Accepted.
[...]
SPGW Allocated IP 172.16.0.2 to ISMI 9017000000*****
```
srsLTE setup

Secrets could be setup in the *user_db.csv* DB of LTE EPC network:

```
# vi /root/.srs/user_db.csv
[...] 
ue3,9017000000*****,b5997ac4a912e9c6216e13951029c674,opc,83e5d3f22da411
072508f675d2e9e9d9,9001,000000000062,7
```

A good configuration should result as follows:

```
[...] 
UE Authentication Accepted.
[...] 
SPGW Allocated IP 172.16.0.2 to IMSI 9017000000****
```

Problems with IoT modems

IoT modems use Cat M1 and NB-IoT → only implemented in commercial/private version of srsLTE and Amarisoft
Go further in 5G

- Use of OpenAirInterface5G
- EPC part requires a licence
- NextEPC or *pycrate_mobile* could be used and readapted for the EPC part
Issues during tests

Generally, data are trusted and sent in clear-text, but there are some exceptions:

- whitelist of connections to the backend;
- use of client side certificates;

Moreover, USIM card could be embeeded → potentially accessible via SPI interface → try a kind of relay attack
1. Requirements
2. Attracting mobile devices
3. Capturing mobile data of a famous intercom in France
4. Hard way
5. Other interesting targets
6. Other interesting targets
7. The futur
8. conclusion
Identifying components

The 3G intercom

- SIM/USIM slot (yellow)
- 3G modem (blue)
- MCU (Microcontroller Unit) (green)
- A strange interface (red)
Use schematics to identify PINs via continuity tests:

**Identified PINs**
- PGC1 (pin 25);
- PGD1 (pin 16);
- Vdd (pin 38);
- /MCLR (pin 7);
- AVss (pin 19).
Interfacing and dumping the firmware

Dumping it with MPLAB-X software
Firmware analysis: strings

Firmware dumped in Intel Hex format and contains AT commands: AT+COPS; AT+CREG

<table>
<thead>
<tr>
<th>Address</th>
<th>Data</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001ab00</td>
<td>02 00 78 00 00 80 fa 00 00 00 06 00 41 54 00 00</td>
<td>. . x ............ AT . .</td>
</tr>
<tr>
<td>0001ab10</td>
<td>2b 4e 00 00 45 54 00 00 43 4c 00 00 4f 53 00 00</td>
<td>+N . . ET . . CL . . OS . .</td>
</tr>
<tr>
<td>0001ab20</td>
<td>45 0d 00 00 00 2b 00 00 43 4c 00 00 49 50 00 00</td>
<td>E . . . + . . CL . . IP . .</td>
</tr>
<tr>
<td>0001ab30</td>
<td>3a 20 00 00 22 1b 00 00 df 22 00 00 2c 1b 00 00</td>
<td>: . . . . . . . . . . . . . .</td>
</tr>
<tr>
<td>0001ab40</td>
<td>ef 00 00 00 45 52 00 00 52 4f 00 00 52 00 00 00</td>
<td>. . . . . . . . . . . . . .</td>
</tr>
<tr>
<td>0001ab50</td>
<td>41 54 00 00 2b 43 00 00 4f 50 00 00 53 3d 00 00</td>
<td>AT . . + C . . OP . . S = . .</td>
</tr>
<tr>
<td>0001ab60</td>
<td>33 2c 00 00 32 0d 00 00 00 41 00 00 54 2b 00 00</td>
<td>3 , . . 2 . . A . . T + . .</td>
</tr>
<tr>
<td>0001ab70</td>
<td>43 4f 00 00 50 53 00 00 3f 0d 00 00 00 2b 00 00</td>
<td>C O . . P S . . ? . . . . + . .</td>
</tr>
<tr>
<td>0001ab80</td>
<td>43 4f 00 00 50 53 00 00 3a 20 00 00 1b ef 00 00</td>
<td>C O . . P S . . . . . . . . .</td>
</tr>
<tr>
<td>0001ab90</td>
<td>2c 1b 00 00 ef 2c 00 00 22 1b 00 00 df 22 00 00</td>
<td>: . . . . . . . . . . . . . .</td>
</tr>
<tr>
<td>0001aba0</td>
<td>2c 1b 00 00 ef 00 00 2b 43 00 00 4f 50 00 00</td>
<td>. . . . . . . . . . . . . .</td>
</tr>
<tr>
<td>0001abb0</td>
<td>53 3a 00 00 20 30 00 00 00 41 00 00 54 2b 00 00</td>
<td>S : . . 0 . . A . . T + . .</td>
</tr>
<tr>
<td>0001abc0</td>
<td>43 4f 00 00 50 53 00 00 3d 34 00 00 2c 32 00 00</td>
<td>C O . . P S . . = 4 . . 2 . .</td>
</tr>
<tr>
<td>0001abd0</td>
<td>2c 1b 00 00 eb 2c 00 00 32 0d 00 00 00 41 00 00</td>
<td>: . . . . . . . . . . . . . .</td>
</tr>
<tr>
<td>0001abe0</td>
<td>54 2b 00 00 43 53 00 00 51 0d 00 00 00 2b 00 00</td>
<td>T + . . C S . . Q . . + . .</td>
</tr>
<tr>
<td>0001abf0</td>
<td>43 53 00 00 51 3a 00 00 20 1b 00 00 ef 2c 00 00</td>
<td>C S . . Q . . . . . . . . .</td>
</tr>
<tr>
<td>0001ac00</td>
<td>1b ef 00 00 00 41 00 00 54 2b 00 00 43 52 00 00</td>
<td>. . . . . . . . . . . . . .</td>
</tr>
<tr>
<td>0001ac10</td>
<td>45 47 00 00 3f 0d 00 00 00 2b 00 00 43 52 00 00</td>
<td>E G . . ? . . . . + . . C R . .</td>
</tr>
<tr>
<td>0001ac20</td>
<td>45 47 00 00 3a 20 00 00 1b ef 00 00 2c 1b 00 00</td>
<td>E G . . . . . . . . . . . . .</td>
</tr>
</tbody>
</table>
Firmware analysis: strings (2)

Looking for strings, it was possible to quickly find AT commands used to connect to endpoints:

- AT+TCPCONNECT="gsm.XXXXXXXXXXX.info",60001;
- AT+TCPCONNECT="gsm.XXXXXXXXXXX.info",5555 (last number ”6” is missing);
- AT+TCPCONNECT="91.121.XX.XX",5555 (last number ”6” is missing).

But also intercom’s number ID XX4015:

00017d80 15 40 XX 00 80 4a 78 00 63 00 60 00 66 40 78 00 |.@X..Jx.c.‘.f@x.|
Firmware disassembly

- No disassembler available for PIC24 before
- But changed with IDA 7.2 and of course Ghidra!
Hardware audit tip

Like almost every vendor’s IDE, MPLAB gives status of memory protections/fuse bits:
Other Interfaces

Various other interfaces could be found in the wild

- UART (Universal Asynchronous Receiver/Transmitter): to interface to bootloader (ex: uBoot) and device terminal
- JTAG (Joint Test Action Group): to communicate with the different devices of the PCB
- SPI (Serial Peripheral Interface): communication MCU ↔ other peripherals
- I²C: link MCU, EEPROMs, and other modules
- others In-chip interfaces, etc.

These interfaces can be found with logic analyzers, probes, but also dedicated tools sometimes...
Device to interface

Various devices could be used to get accesses to an interface:

- The famous SEGGER JLink that works like a charm, but expensive depending on options...
- Bus pirate v3 (warning v4 not mature enough)
- BusVoodoo → supports 14 TTL/CMOS protocols
- HydraBUS → another powerful swiss knife (include a funny NFC modules for emulation and could be used to bruteforce JTAG PINs)
- and so on.

Sometimes rare/industrial protocols and MCUs could also be supported by Trace32 tools → it has a costs
Bruteforcing JTAG and UART PINs

For almost 200€ with JTAGulator
Bruteforcing JTAG and UART PINs (2)

With BUSSide for almost 8€:
Chip-off in last resort

Example with a TSOP48 flash:
Memory protections bypasses

- Block reading by backdooring the entrypoint on PIC18F552 (ex: iCLASS keys extraction)
- Cold-Boot stepping attacks on STM32F0 series
- UV-C attacks
- RDP2 downgrade to RDP1 on STM32F1 and STM32F3 (ex: TREZOR wallet hack → wallet.fail)
- and so on.
1 Requirements

2 Attracting mobile devices

3 Capturing mobile data of a famous intercom in France

4 Hard way

5 Other interesting targets

6 Other interesting targets

7 The futur

8 conclusion
Other targets

- Like intercoms: use of Mobile network is convenient → no wires no problem
- Overcases:
  - Deposit cases;
  - Alarms;
  - Connected cars...
1 Requirements

2 Attracting mobile devices

3 Capturing mobile data of a famous intercom in France

4 Hard way

5 Other interesting targets

6 Other interesting targets

7 The future

8 Conclusion
Other targets

- Like intercoms: use of Mobile network is convenient → no wires no problem
- Overcases:
  - Deposit cases;
  - Alarms;
  - Connected cars...
Garage hacker: the CAN bus

- ODB/ODB2 interface: a lot of interest
- Possible to interact in the CAN bus
- But too many messages are broadcasted in it → needs processing to focus on interesting messages

However, the car as many interfaces that interacts with the CAN bus
Connected cars

- Mobile network is generally used
- Possible to install applications
- GPRS is generally used for middle class cars → really easy to intercept
- But parking cars are also well isolated → Modmobjam not needed
Our target

- Enable the installation of applications
- Can be update
- Plenty of available applications:
  - Twitter application and Facebook (WTF?)
  - Meteo
  - GPS
  - etc.

And all of that "in the air"
Hunting for mobile modules remotely

Using a BladeRF:
Issues in our context

- The servers could not be contacted with an arbitrary connection :/
- We can still poison/hook all DNS queries and get requests from clients → attack the client with a fake server
Client-side attack: new captures

 Surprise: all requests made by the board computer and apps are in clear HTTP...

```
| 19 1.450318826 | 102.168.99.2 | 102.168.99.254 | HTTP | 013 POST /Service/InitSession/ | HTTP/1.1 | (application/x-protobuf) |
| 19 7.536599065 | 102.168.99.2 | 10.01.80.203 | HTTP | 52 HEAD http://master.coyoterts.com HTTP/1.1 |
| 26 13.686817736 | 102.168.99.2 | 10.01.80.203 | HTTP | 52 HEAD http://master.coyoterts.com HTTP/1.1 |
| 65921 022.70428110 | 102.168.99.2 | 10.01.80.203 | HTTP | 52 HEAD http://master.coyoterts.com HTTP/1.1 |
| 65923 045.703883356 | 102.168.99.2 | 10.01.80.203 | HTTP | 52 HEAD http://master.coyoterts.com HTTP/1.1 |
| 66965 974.461373208 | 102.168.99.254 | 102.168.99.2 | HTTP | 173 HTTP/1.0 404 File not found |
| 66993 974.918419068 | 102.168.99.254 | 102.168.99.2 | HTTP | 52 HEAD http://master.coyoterts.com HTTP/1.1 |
| 79395 990.593015750 | 102.168.99.2 | 102.168.99.254 | HTTP | 406 POST /api/app/call HTTP/1.1 (application/x-protobuf) |
| 79491 990.594776502 | 102.168.99.254 | 102.168.99.2 | HTTP | 300 HTTP/1.0 501 Unsupported method ('POST') (text/html) |
| 79549 991.484023266 | 102.168.99.254 | 102.168.99.2 | HTTP | 406 POST /api/app/call HTTP/1.1 (application/x-protobuf) |
| 79593 992.48710425 | 102.168.99.2 | 102.168.99.2 | HTTP | 300 HTTP/1.0 501 Unsupported method ('POST') (text/html) |
| 79535 992.484544176 | 102.168.99.254 | 102.168.99.2 | HTTP | 406 POST /api/app/call HTTP/1.1 (application/x-protobuf) |
| 1048. 1508.1445388 | 102.168.99.2 | 102.168.99.254 | HTTP | 406 POST /api/app/call HTTP/1.1 (application/x-protobuf) |
| 1048. 1509.1456076 | 102.168.99.2 | 102.168.99.254 | HTTP | 406 POST /api/app/call HTTP/1.1 (application/x-protobuf) |
| 1048. 1591.1456861 | 102.168.99.2 | 102.168.99.254 | HTTP | 406 POST /api/app/call HTTP/1.1 (application/x-protobuf) |
| 1049. 1501.8855224 | 102.168.99.2 | 102.168.99.254 | HTTP | 406 POST /api/app/call HTTP/1.1 (application/x-protobuf) |
```
Client-side attack: sweets

Hypertext Transfer Protocol
- POST /api/app/call HTTP/1.1
  Content-Type: application/x-protobuf; charset=utf-8
  Accept-Encoding: gzip
  User-Agent: Dalvik/1.6.0 (Linux; U; Android 4.0.4; ARM2-MX6DQ Build/UNKNOWN)
  Host: fr.aw.atos.net
  Connection: Keep-Alive
  Content-Length: 91
  
  [Full request URI: http://fr.aw.atos.net/api/app/call]
  [HTTP request 1/1]
  [Response in frame: 70533]
  File Data: 91 bytes

Media Type
Opportunities

Remember the Android version is 4.0.4:

- Some apps perform web requests → JavaScript Interface
  RCE
- Other request XML files → XXE attacks
- And all other CVE to replay!
Spotted API

POST api/app/call HTTP/1.1
Content-Type: application/x-protobuf; charset=utf-8
Accept-Encoding: gzip
User-Agent: Dalvik/1.6.0 (Linux; U; Android 4.0.4; ARM2-MX6DQ Build/UNKNOWN)
Host: fr-.aw.atos.net
Connection: Keep-Alive
Content-Length: 91

0
@dd5ee7f410efe36e5ef12d144f2d11fe890f85432c6e37c64d558daf3ccb8bb5....FR".fr_FR...."...2.HTTP/1.0 501 Unsupported method ('POST')
Server: SimpleHTTP/0.6 Python/2.7.15
Date: Thu, 30 Aug 2018 11:57:36 GMT
Connection: close
Content-Type: text/html

<head>
<title>Error response</title>
</head>
<body>
<h1>Error response</h1>
<p>Error code 501.
Message: Unsupported method ('POST').
Error code explanation: 501 = Server does not support this operation.
</body>

Very similar to mobile app API calls! But no “OAuth” token?!
API: “Mobile app” VS “Cars/others...”

**Mobile APP**
- open and close car door
- start/stop the clim
- all of these actions are authentified → OAuth, etc.
- uses HTTPS → well verified by default on new Android device

**Cars and others**
- open and close car door
- start/stop the clim
- talks on HTTP
- sometimes use only SMS messages
- use only identification
- payload are sometimes encrypted with a same shared key
- rare cases: mutual authentication (especially on external dongles)

In most cases car board computers needs to be reversed.
Interception in a parking station

> 10 board computers collected in the fake base station
Read more about this

- Our blog post: Hunting mobile devices endpoints
- More stuff could be found on other systems...
- Other case: The ComboBox in BMW
Requirements

2 Attracting mobile devices

3 Capturing mobile data of a famous intercom in France

4 Hard way

5 Other interesting targets

6 Other interesting targets

7 The future

8 Conclusion
mPCI-e

perfect for embedded radio

osmoTRX is not well supported at the moment, but patience!

fit perfectly on APU2, UP2 and Orange Pi rk3399 boards
APU2 example
1. Requirements
2. Attracting mobile devices
3. Capturing mobile data of a famous intercom in France
4. Hard way
5. Other interesting targets
6. Other interesting targets
7. The future
8. Conclusion
Conclusion

- A lot of IoT devices use the mobile network to be managed in remote.
- Mobile interception techniques could be applied on IoT device.
- Techniques are accessible → equipments, tools and tricks are not so expensive.
- Modmobmap and Modmobjam → when physical accesses are not possible on targeted devices.
- But some devices only have a 3G or a LTE stack.
- Interceptions on UMTS and LTE requires a custom (U)SIM (unless there is a missing auth check in BB).
- Hardware hacking → complementary but also a last ressort sometimes.
Downloads

- **Modmobmap:**
  - https://github.com/Synacktiv/Modmobmap

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

- Joffrey Czarny (@_Sn0rkY)
- Priya Chalakkal (@priyachalakkal)
- Rachelle Boissard (@rachelle_off)
- Troopers staff (@WEareTROOPERS)
- Guillaume Delugré (@lapinhib0u) → spotting few mistakes in slide 3
- And of course → You all ;)
THANK YOU FOR YOUR ATTENTION,

ANY QUESTIONS?