Modmob tools and tricks

Using cheap tools and tricks to attack mobile devices in practice

By Sébastien Dudek

Troopers - NGI

March 18th 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
- First time at Troopers =)!

This presentation

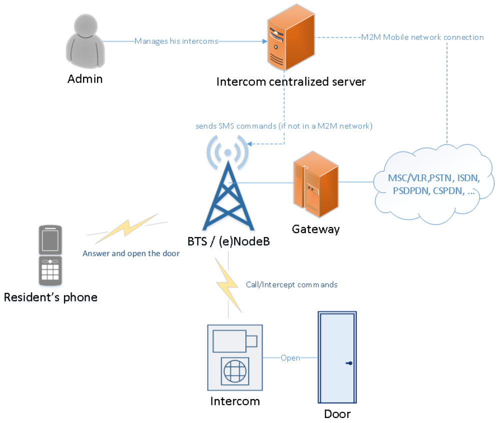
- Few reminders:
  - talk about interception techniques in practice
  - existing tools
- Our contribution:
  - feedbacks of our tests (mobile phones, intercoms, cars...)
  - tools we made (Modmobmap and Modmobjam);
  - some cheap tricks;
  - some hardware attacks.

+ meet us tomorrow at Telco Security day $\rightarrow$ Modmob tools internals, updates, and more! ;)

Introduction

- Mobile network $\rightarrow$ more than 30 years
  - 1G: analogic, very large band $\rightarrow$ 400 MHz;
  - 2G: FDMA (25 MHz) in combination with TDMA (in Europe);
  - 3G: WCDMA (5-20 MHz)
  - 4G: Multi-carrier CDMA or OFDM (5-20 MHz)
  - ...
- Evolution of modulation techniques and encoding $\rightarrow$ 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

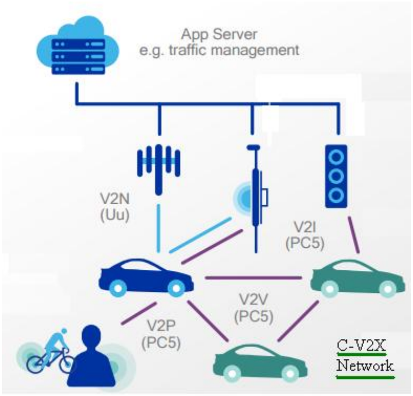


Admin
Manages his intercoms
Intercom centralized server
M2M Mobile network connection
sends SMS commands (if not in a M2M network)
MSC/VLR,PSTN, ISDN, PSDPDN, CSPDN, ...
Gateway
BTS / (e)NodeB
Answer and open the door
Resident's phone
Call/Intercept commands
Open
Intercom
Door

Pretty the same with connected cars!

5G is coming...

- LTE-A(dvanced)++ $\rightarrow$ 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 $\rightarrow$ 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.

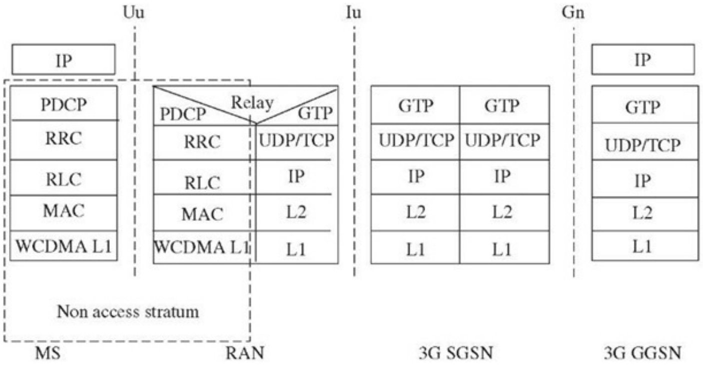
Security of communications (2)

	GSM	3G	4G
Client authentication	YES	YES	YES
Network authentication	NO	Only if USIM is used (not SIM)	YES
Signaling integrity	NO	YES	YES
Encryption	A5/1	KASUMI | SNOW-3G	SNOW-3G | AES | ZUC...

→ 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

Software-Defined radio

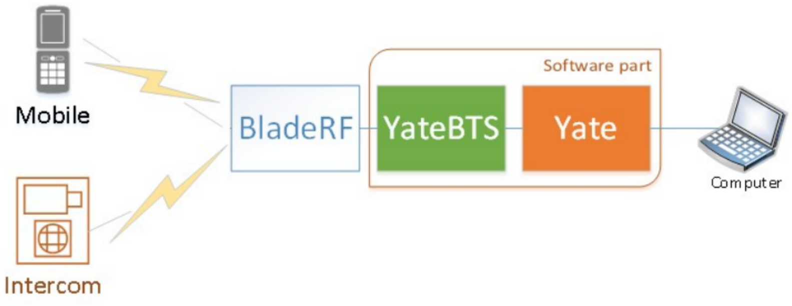
To interface to devices using the mobile network:

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

Alternatives

- sysmoBTS for GSM and GPRS
- sysmoNITB for 3G/LTE $\rightarrow$ requires a custom/vulnerable femtocell
- LTE LabKit by Yate for LTE;
- Amarisoft LTE $\rightarrow$ 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 $\rightarrow$ helped some researchers to find vulns in CDMA baseband stacks ;)

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

```
...
[gprs]
Enable=yes
...
```

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

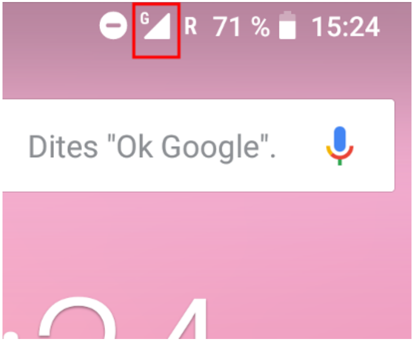
```
...
[ggsn]
DNS=8.8.8.8 8.8.4.4 ; its preferable to use your own servers for client side attacks
IP.MaxPacketSize=1520
IP.ReuseTimeout=180
IP.TossDuplicatePackets=no
Logfile.Name=/tmp/sgsn.log
MS.IP.Base=192.168.99.1
MS.IP.MaxCount=254
TunName=sgsntun
...
```

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):



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'ill see how to revisit it with cheap equipments + some style ;)

Method 1: Custom SIM/USIM cards

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

$\rightarrow$ Make the fake BTS/(e)NodeB act as a legit BTS

sysmocom
systems for mobile communications GmbH
custom-tailored GSM solutions
from RAN to Core Network
Visit us at http://sysmocom.de/
• First-hand expertise in protocol R&D from A-bis to SS7/MAP
• Support, training and development for OpenBSC, OsmoSGSN
• Low-cost GSM pico-cell platform sysmoBTS
• Small, PBX-style autonomous GSM networks

Method 1: Custom SIM/USIM cards

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

$\rightarrow$ Make the fake BTS/(e)NodeB act as a legit BTS

sysmocom
systems for mobile communications GmbH
custom-tailored GSM solutions
from RAN to Core Network
Visit us at http://sysmocom.de/
• First-hand expertise in protocol R&D from A-bis to SS7/MAP
• Support, training and development for OpenBSC, OsmoSGSN
• Low-cost GSM pico-cell platform sysmoBTS
• Small, PBX-style autonomous GSM networks
sysmoUSIM-SJS1

Caution

Becaution with PIN auto-typing $\rightarrow$ use a SIMtrace tool to get the typed PIN

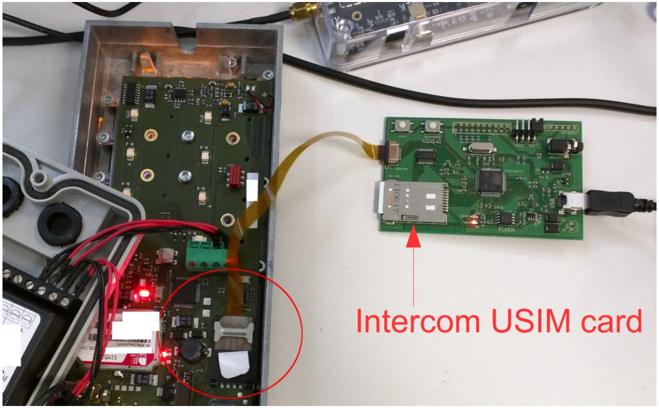
Program sysmoUSIM cards

- Could be entirely configured $\rightarrow$ 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 50024782 -x 001 -y 01 -i
9017000000***** -s 89882110000002****** [...]
> Ki : 6abb9ae663f9889eddaae298cdcb4ec6
> OPC : 074a3a73ed3c54e1960e9e5732ff35b1
> ACC : None
```

SIMtrace for the rescue

Sniff auto-typed PINs with the Osmocom SIMtrace:



Method 2: Faraday cage

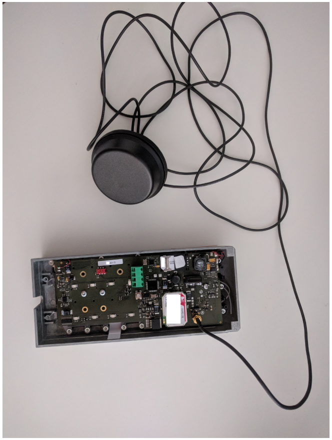
- Mostly cumbersome and expensive
- But could be improvised considering several elements:
  - Frequency;
  - Wavelength;
  - Power of reception or transmission;
  - Distance between the receiver and the transmitter.
- Cage with meshes $\rightarrow$ optimised windows against reflection of the electric 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

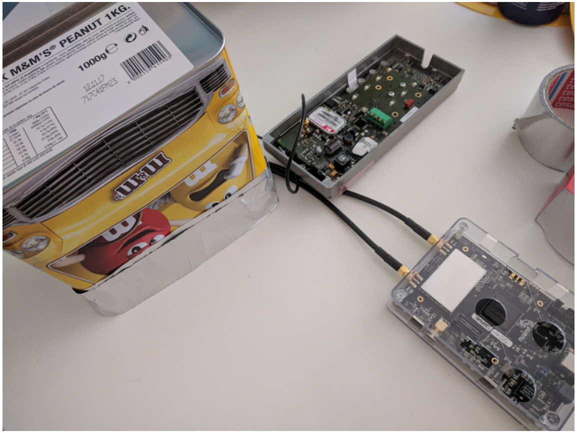
Space optimisation

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 chear 2G/3G/4G jammer and rework it
- Or perform smart-jamming:
  1. monitor and collect cells data
  2. jam precise frequencies from collected cells $\rightarrow$ 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 ;)

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:

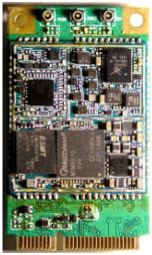
- Airprobe or GR-GSM
- OpenLTE: *LTE_fdd_dl_scan*
- srsLTE with srsUE

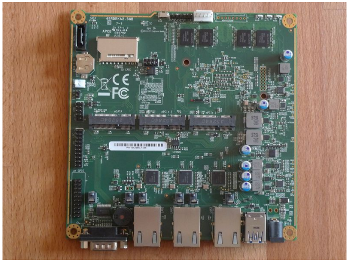
No 3G

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

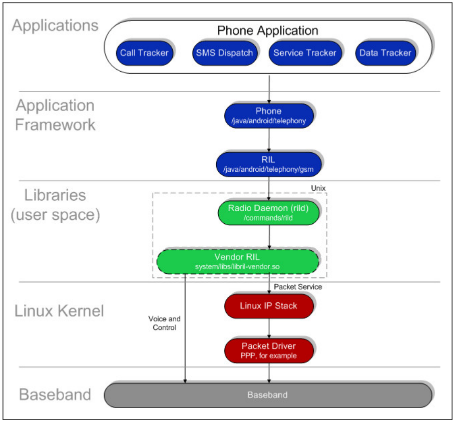

PCengines APU2

Cheaper way

- U/EC20 3G/LTE modem
- And an adaptater with (U)SIM slot

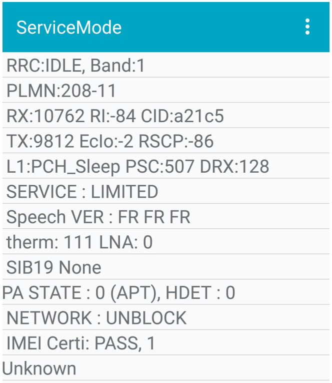
RIL on Android

- Daemon forwards commands/messages: application ⇆Vendor RIL
- vendor library is prorietary 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/and transmission (TX/UP) (E/U)ARFCN (Absolute Radio Frequency Channel Number)
  - PLMN (Public Land Mobile Network) number
  - and so on.



```
RRC:IDLE, Band:1
PLMN:208-11
RX:10762 RI:-84 CID:a21c5
TX:9812 Eclo:-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
```

ServiceMode in Samsung

Samsung ServiceMode in brief

1. **#0011#* secret code handled by *ServiceModeApp_RIL ServiceModeApp* activity
2. ServiceModeApp $\rightarrow$IPC connection $\rightarrow$*SecFactoryPhoneTest SecPhoneService*
3. *ServiceModeApp* starts the service mode $\rightarrow$*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 contraints 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 $\rightarrow$MS looks for cells satisfying few conditions (cell of the selected PLMN, not barred, pathloss between MS and BTS below a thresold, and so on.)
- Cells are checked in a descending order of the signal strength
- If a suitable is found $\rightarrow$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 $\rightarrow$MS looks for cells satisfying few conditions (cell of the selected PLMN, not barred, pathloss between MS and BTS below a thresold, and so on.)
- Cells are checked in a descending order of the signal strength
- If a suitable is found $\rightarrow$MS camps on it and tries to register

Verified through DIAG and ServiceMode

If registration fails $\rightarrow$MS camps to another cell until it can register $\rightarrow$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
```

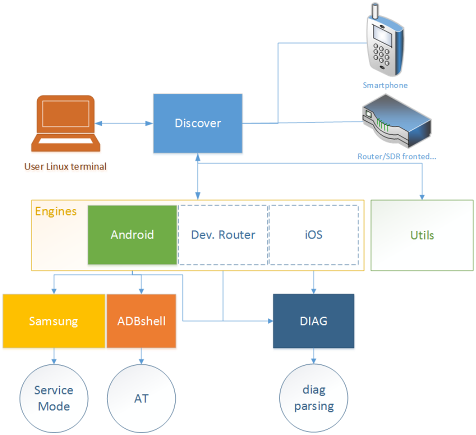
It is possible to:

- 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  (XXXXXXXXX)]
 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  (XXXXXXXXXX)]
 Network type=2G
 PLMN=208—20
 ARFCN=76
  [...]
  [+] New cell detected [CellID/PCI—DL_freq  (XXXXXXXXXX)]
 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:

```
{
    "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"
    },
[...]
}
```

$\rightarrow$ but we'll see how it could be used for Jamming purposes!

Jamming in general

With a portable/chineeese device

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

Desktop jammers

Jamming in general

With a portable/chineese device

Desktop jammers

- heavy, cumbersome but powerfull
- 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! $\rightarrow$ Modmobjam $\rightarrow$ use Software-Defined radio

”Smart” jamming

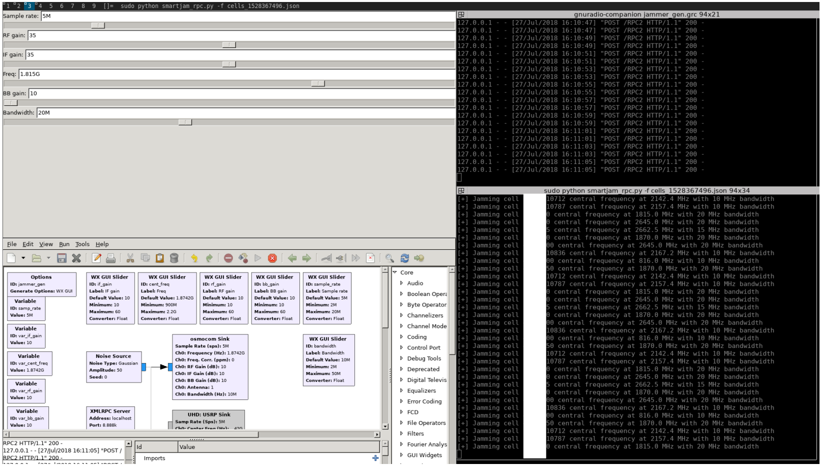
- Jam only targeted cells
- Stealth against monitors
- In 3 steps:
  1. scan cells with Modmobmap;
  2. target an operator;
  3. and jam only targeted channels;

We have also made a tool for that! $\rightarrow$ Modmobjam $\rightarrow$ 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



```
1 2 3 4 5 6 7 8 9 []= sudo python smartjam_rpc.py -f cells_1528367496.json
```

Sample rate: 5M
RF gain: 35
IF gain: 35
Freq: 1.815G
BB gain: 10
Bandwidth: 20M

Options — ID: jammer_gen; Generate Options: WX GUI

Variable — ID: samp_rate; Value: 5M

Variable — ID: var_if_gain; Value: 10

Variable — ID: var_cent_freq; Value: 1.8742G

Variable — ID: var_rf_gain; Value: 10

Variable — ID: var_bb_gain; Value: 10

WX GUI Slider — ID: if_gain; Label: IF gain; Default Value: 10; Minimum: 10; Maximum: 60; Converter: Float

WX GUI Slider — ID: cent_freq; Label: Freq; Default Value: 1.8742G; Minimum: 900M; Maximum: 2.2G; Converter: Float

WX GUI Slider — ID: rf_gain; Label: RF gain; Default Value: 10; Minimum: 10; Maximum: 60; Converter: Float

WX GUI Slider — ID: bb_gain; Label: BB gain; Default Value: 10; Minimum: 10; Maximum: 60; Converter: Float

WX GUI Slider — ID: sample_rate; Label: Sample rate; Default Value: 5M; Minimum: 2M; Maximum: 20M; Converter: Float

Noise Source — Noise Type: Gaussian; Amplitude: 50; Seed: 0

osmocom Sink — Sample Rate (sps): 5M; Ch0: Frequency (Hz): 1.8742G; Ch0: Freq. Corr. (ppm): 0; Ch0: RF Gain (dB): 10; Ch0: IF Gain (dB): 10; Ch0: BB Gain (dB): 10; Ch0: Antenna: 1; Ch0: Bandwidth (Hz): 10M

WX GUI Slider — ID: bandwidth; Label: Bandwidth; Default Value: 10M; Minimum: 2M; Maximum: 50M; Converter: Float

XMLRPC Server — Address: localhost; Port: 8.888k

UHD: USRP Sink — Samp Rate (Sps): 5M

```
RPC2 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2018 16:11:05] "POST /
RPC2 HTTP/1.1" 200 -
```

gnuradio-companion jammer_gen.grc 94x21

```
127.0.0.1 - - [27/Jul/2018 16:10:47] "POST /RPC2 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2018 16:10:47] "POST /RPC2 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2018 16:10:49] "POST /RPC2 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2018 16:10:49] "POST /RPC2 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2018 16:10:51] "POST /RPC2 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2018 16:10:51] "POST /RPC2 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2018 16:10:53] "POST /RPC2 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2018 16:10:53] "POST /RPC2 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2018 16:10:55] "POST /RPC2 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2018 16:10:55] "POST /RPC2 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2018 16:10:57] "POST /RPC2 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2018 16:10:57] "POST /RPC2 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2018 16:10:59] "POST /RPC2 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2018 16:10:59] "POST /RPC2 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2018 16:11:01] "POST /RPC2 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2018 16:11:01] "POST /RPC2 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2018 16:11:03] "POST /RPC2 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2018 16:11:03] "POST /RPC2 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2018 16:11:05] "POST /RPC2 HTTP/1.1" 200 -
127.0.0.1 - - [27/Jul/2018 16:11:05] "POST /RPC2 HTTP/1.1" 200 -
```

sudo python smartjam_rpc.py -f cells_1528367496.json 94x34

```
[+] Jamming cell [redacted] 10712 central frequency at 2142.4 MHz with 10 MHz bandwidth
[+] Jamming cell            10787 central frequency at 2157.4 MHz with 10 MHz bandwidth
[+] Jamming cell            0 central frequency at 1815.0 MHz with 20 MHz bandwidth
[+] Jamming cell            0 central frequency at 2645.0 MHz with 20 MHz bandwidth
[+] Jamming cell            5 central frequency at 2662.5 MHz with 15 MHz bandwidth
[+] Jamming cell            0 central frequency at 1870.0 MHz with 20 MHz bandwidth
[+] Jamming cell            00 central frequency at 2645.0 MHz with 20 MHz bandwidth
[+] Jamming cell            10836 central frequency at 2167.2 MHz with 10 MHz bandwidth
[+] Jamming cell            00 central frequency at 816.0 MHz with 10 MHz bandwidth
[+] Jamming cell            50 central frequency at 1870.0 MHz with 20 MHz bandwidth
[+] Jamming cell            10712 central frequency at 2142.4 MHz with 10 MHz bandwidth
[+] Jamming cell            10787 central frequency at 2157.4 MHz with 10 MHz bandwidth
[+] Jamming cell            0 central frequency at 1815.0 MHz with 20 MHz bandwidth
[+] Jamming cell            0 central frequency at 2645.0 MHz with 20 MHz bandwidth
[+] Jamming cell            5 central frequency at 2662.5 MHz with 15 MHz bandwidth
[+] Jamming cell            0 central frequency at 1870.0 MHz with 20 MHz bandwidth
[+] Jamming cell            00 central frequency at 2645.0 MHz with 20 MHz bandwidth
[+] Jamming cell            10836 central frequency at 2167.2 MHz with 10 MHz bandwidth
[+] Jamming cell            00 central frequency at 816.0 MHz with 10 MHz bandwidth
[+] Jamming cell            50 central frequency at 1870.0 MHz with 20 MHz bandwidth
[+] Jamming cell            10712 central frequency at 2142.4 MHz with 10 MHz bandwidth
[+] Jamming cell            10787 central frequency at 2157.4 MHz with 10 MHz bandwidth
[+] Jamming cell            0 central frequency at 1815.0 MHz with 20 MHz bandwidth
[+] Jamming cell            0 central frequency at 2645.0 MHz with 20 MHz bandwidth
[+] Jamming cell            5 central frequency at 2662.5 MHz with 15 MHz bandwidth
[+] Jamming cell            0 central frequency at 1870.0 MHz with 20 MHz bandwidth
[+] Jamming cell            00 central frequency at 2645.0 MHz with 20 MHz bandwidth
[+] Jamming cell            10836 central frequency at 2167.2 MHz with 10 MHz bandwidth
[+] Jamming cell            00 central frequency at 816.0 MHz with 10 MHz bandwidth
[+] Jamming cell            50 central frequency at 1870.0 MHz with 20 MHz bandwidth
[+] Jamming cell            10712 central frequency at 2142.4 MHz with 10 MHz bandwidth
[+] Jamming cell            10787 central frequency at 2157.4 MHz with 10 MHz bandwidth
[+] Jamming cell            0 central frequency at 1815.0 MHz with 20 MHz bandwidth
```

Analyzing GPRS data

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

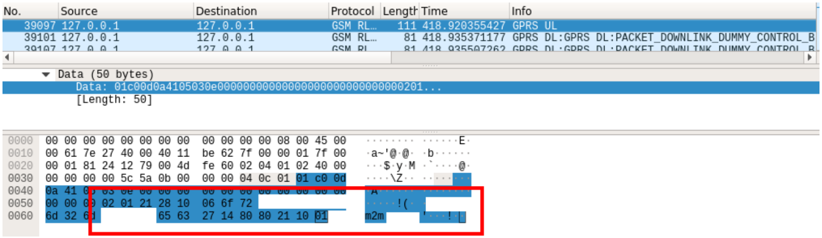
```
nipc list registered
IMSI              MSISDN
----------------- -----------------
20801XXXXXXXXXXX     69691320681
```

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



```
No.     Source        Destination     Protocol Length Time           Info
39097   127.0.0.1     127.0.0.1       GSM RL…   111  418.920355427  GPRS UL
39101   127.0.0.1     127.0.0.1       GSM RL…    81  418.935371177  GPRS DL:GPRS DL:PACKET_DOWNLINK_DUMMY_CONTROL_B
39107   127.0.0.1     127.0.0.1       GSM RL…    81  418.935507262  GPRS DL:GPRS DL:PACKET_DOWNLINK_DUMMY_CONTROL_B
```

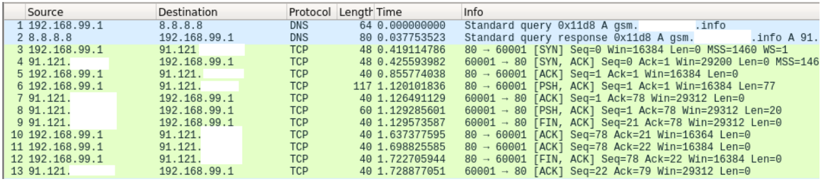
```
    ▾ Data (50 bytes)
        Data: 01c00d0a4105030e00000000000000000000000000000201...
        [Length: 50]
```

```
0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ········ ······E·
0010  00 61 7e 27 40 00 40 11  be 62 7f 00 00 01 7f 00   ·a~'@·@· ·b······
0020  00 01 81 24 12 79 00 4d  fe 60 02 04 01 02 40 00   ···$·y·M ·`····@·
0030  00 00 00 00 5c 5a 0b 00  00 00 04 0c 01 01 c0 0d   ····\Z·· ········
0040  0a 41 05 03 0e 00 00 00  00 00 00 00 00 00 00 00   ·A······ ········
0050  00 00 00 02 01 21 28 10  06 6f 72 [redacted]        ·····!(· [redacted]
0060  6d 32 6d [redacted] 65 63  27 14 80 80 21 10 01      m2m[redacted] '···!·
```

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

Capture exchanges

On the tun interface dedicated to SGSN:



	Source	Destination	Protocol	Length	Time	Info
1	192.168.99.1	8.8.8.8	DNS	64	0.000000000	Standard query 0x11d8 A gsm.[redacted].info
2	8.8.8.8	192.168.99.1	DNS	80	0.037753523	Standard query response 0x11d8 A gsm.[redacted].info A 91.
3	192.168.99.1	91.121[redacted]	TCP	48	0.419114786	80 → 60001 [SYN] Seq=0 Win=16384 Len=0 MSS=1460 WS=1
4	91.121.[redacted]	192.168.99.1	TCP	48	0.425593982	60001 → 80 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=146
5	192.168.99.1	91.121.[redacted]	TCP	40	0.855774038	80 → 60001 [ACK] Seq=1 Ack=1 Win=16384 Len=0
6	192.168.99.1	91.121.[redacted]	TCP	117	1.120101836	80 → 60001 [PSH, ACK] Seq=1 Ack=1 Win=16384 Len=77
7	91.121.[redacted]	192.168.99.1	TCP	40	1.126491129	60001 → 80 [ACK] Seq=1 Ack=78 Win=29312 Len=0
8	91.121.[redacted]	192.168.99.1	TCP	60	1.129285601	60001 → 80 [PSH, ACK] Seq=1 Ack=78 Win=29312 Len=20
9	91.121.[redacted]	192.168.99.1	TCP	40	1.129573587	60001 → 80 [FIN, ACK] Seq=21 Ack=78 Win=29312 Len=0
10	192.168.99.1	91.121.[redacted]	TCP	40	1.637377595	80 → 60001 [ACK] Seq=78 Ack=21 Win=16364 Len=0
11	192.168.99.1	91.121.[redacted]	TCP	40	1.698825585	80 → 60001 [ACK] Seq=78 Ack=22 Win=16384 Len=0
12	192.168.99.1	91.121.[redacted]	TCP	40	1.722705944	80 → 60001 [FIN, ACK] Seq=78 Ack=22 Win=16384 Len=0
13	91.121.[redacted]	192.168.99.1	TCP	40	1.728877051	60001 → 80 [ACK] Seq=22 Ack=79 Win=29312 Len=0

In that case: two server ports identified $\rightarrow$ 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("011e4d25636014006600000000000000090000011e1540XX[...]"))
Out[8]: 320
In [9]: data = s.recv(1024)
In [10]: data
Out[11]: '2018/09/07 15:09:01\n'
```

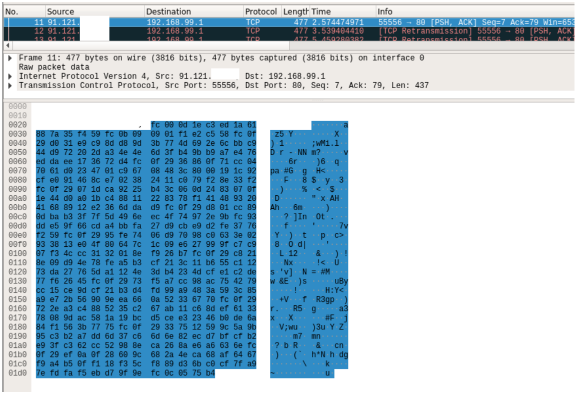
In that case: two server ports identified $\rightarrow$ 60001/tcp and 55556/tcp

Identification

And could noticed that messages where only identified:

Strange messages

When updating the device: some unknown messages are exchanged on port 55556/tcp



```
No.  Source           Destination    Protocol Length Time        Info
  11 91.121.[redacted] 192.168.99.1  TCP      477 2.574474971  55556 → 80 [PSH, ACK] Seq=7 Ack=79 Win=653
  12 91.121.[redacted] 192.168.99.1  TCP      477 3.539404410  [TCP Retransmission] 55556 → 80 [PSH, ACK]
  13 91.121.[redacted] 192.168.99.1  TCP      477 5.459280382  [TCP Retransmission] 55556 → 80 [PSH, ACK]

▸ Frame 11: 477 bytes on wire (3816 bits), 477 bytes captured (3816 bits) on interface 0
  Raw packet data
▸ Internet Protocol Version 4, Src: 91.121.[redacted] Dst: 192.168.99.1
▸ Transmission Control Protocol, Src Port: 55556, Dst Port: 80, Seq: 7, Ack: 79, Len: 437

0000
0010
0020                           fc 00 0d 1e c3 ed 1a 61            .......a
0030  88 7a 35 f4 59 fc 0b 09  09 01 f1 e2 c5 58 fc 0f   .z5.Y... .....X..
0040  29 d0 31 e9 c9 8d d8 9d  3b 77 4d 69 2e 6c bb c9   ).1..... ;wMi.l..
0050  44 d9 72 20 2d a3 4e 4e  6d 3f b4 9b b9 a7 e4 76   D.r -.NN m?.....v
0060  ed da ee 17 36 72 d4 fc  0f 29 36 86 0f 71 cc 04   ....6r.. .)6..q..
0070  70 61 d0 23 47 01 c9 67  08 48 3c 80 00 19 1c 92   pa.#G..g .H<.....
0080  cf e0 91 46 8c e7 02 38  24 11 c0 79 f2 8e 33 f2   ...F...8 $..y..3.
0090  fc 0f 29 07 1d ca 92 25  b4 3c 06 0d 24 83 07 0f   ..)....% .<..$...
00a0  1e 44 d0 a0 1b c4 88 11  22 83 78 f1 41 48 93 20   .D...... ".x.AH. 
00b0  41 68 89 12 e2 36 6d da  d9 fc 0f 29 d8 01 cc 89   Ah...6m. ...)....
00c0  0d ba b3 3f 7f 5d 49 6e  ec 4f 74 97 2e 9b fc 93   ...?.]In .Ot.....
00d0  dd e5 9f 66 cd a4 bb fa  27 d9 cb e9 d2 fe 37 76   ...f.... '.....7v
00e0  f2 59 fc 0f 29 95 fe 74  06 d9 70 98 c0 63 3e 02   .Y..)..t ..p..c>.
00f0  93 38 13 e0 4f 80 64 7c  1c 09 e6 27 99 9f c7 c9   .8..O.d| ...'....
0100  07 f3 4c cc 31 32 01 8e  f9 26 b7 fc 0f 29 c8 21   ..L.12.. .&...).!
0110  8e 09 d9 4e 78 fe a5 b3  cf 21 3c 11 b6 55 c1 12   ...Nx... .!<..U..
0120  73 da 27 76 5d a1 12 4e  3d b4 23 4d cf e1 c2 de   s.'v]..N =.#M....
0130  77 f6 26 45 fc 0f 29 73  f5 a7 cc 98 ac 75 42 79   w.&E..)s .....uBy
0140  cc 15 ce 9d cf 21 b3 d4  fd 99 a9 48 3a 59 3c 85   .....!.. ...H:Y<.
0150  a9 e7 2b 56 90 9e ea 66  0a 52 33 67 70 fc 0f 29   ..+V...f .R3gp..)
0160  72 2e a3 c4 88 52 35 c2  67 ab 11 c6 8d ef 61 33   r....R5. g.....a3
0170  78 08 9d ac 58 1a 19 bc  d5 ce e3 23 46 b0 de 6a   x...X... ...#F..j
0180  84 f1 56 3b 77 75 fc 0f  29 33 75 12 59 9c 5a 9b   ..V;wu.. )3u.Y.Z.
0190  95 c3 b2 a7 dd 6d 37 c6  6d 6e 82 ec d7 bf cf b2   .....m7. mn......
01a0  e9 3f c3 62 cc 52 98 8e  ca 26 8a e6 a6 63 6e fc   .?.b.R.. .&...cn.
01b0  0f 29 ef 0a 0f 28 60 9c  68 2a 4e ca 68 af 64 67   .)...(`. h*N.h.dg
01c0  f9 a4 b5 0f f1 18 f3 5c  f8 89 d3 6b c0 cf 7f a9   .......\ ...k....
01d0  7e fd fa f5 eb d7 9f 9e  fc 0c 05 75 b4            ~....... ...u.
```

Strange messages (1)

By a naive approach it looked to be encrypted:

```
$ 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 $\rightarrow$ 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 $\rightarrow$ free and stable
- Secrets of the SIM should be configured (ex. sysmoUSIM):
  - RAND: generated challenge by the HSS (Home Suscriber Server) in the HLR/AuC $\rightarrow$ 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

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 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 ISMI 9017000000*****
```

Problems with IoT modems

IoT modems use Cat M1 and NB-IoT $\rightarrow$ 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:

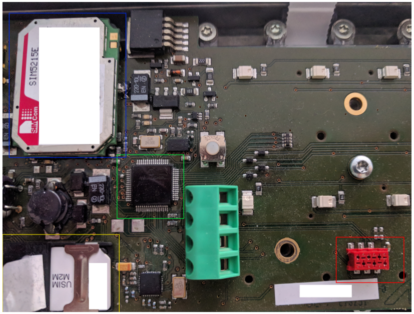
- whitelist of connections to the backend;
- use of client side certificates;

Moreover, USIM card could be embeeded $\rightarrow$ potentially accessible via SPI interface $\rightarrow$ try a kind of relay attack

Identifying components

The 3G intercom

- SIM/USIM slot (yellow)
- 3G modem (blue)
- MCU (Microcontroller Unit) (green)
- A strange interface (red)

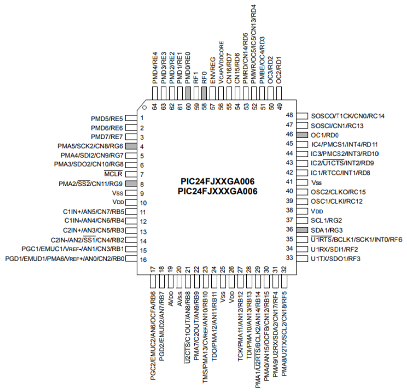


Microchip - PIC24FJ128 - GA006

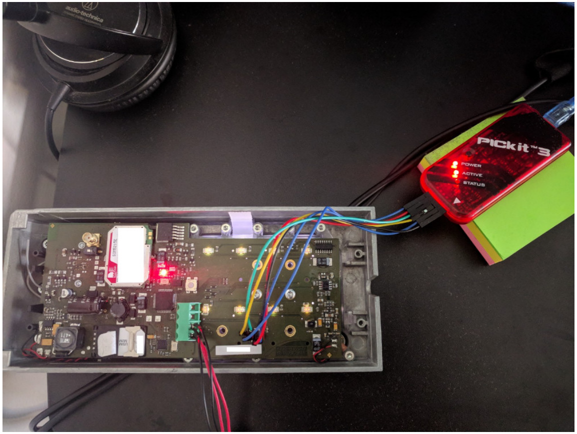
Use schematics to identify PINs via continuity tests:

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

```
0001ab00  02 00 78 00 00 80 fa 00  00 00 06 00 41 54 00 00  |..x.........AT..|
0001ab10  2b 4e 00 00 45 54 00 00  43 4c 00 00 4f 53 00 00  |+N..ET..CL..OS..|
0001ab20  45 0d 00 00 00 2b 00 00  43 4c 00 00 49 50 00 00  |E....+..CL..IP..|
0001ab30  3a 20 00 00 22 1b 00 00  df 22 00 00 2c 1b 00 00  |: .."....."..,...|
0001ab40  ef 00 00 00 45 52 00 00  52 4f 00 00 52 00 00 00  |....ER..RO..R...|
0001ab50  41 54 00 00 2b 43 00 00  4f 50 00 00 53 3d 00 00  |AT..+C..OP..S=..|
0001ab60  33 2c 00 00 32 0d 00 00  00 41 00 00 54 2b 00 00  |3,..2....A..T+..|
0001ab70  43 4f 00 00 50 53 00 00  3f 0d 00 00 00 2b 00 00  |CO..PS..?....+..|
0001ab80  43 4f 00 00 50 53 00 00  3a 20 00 00 1b ef 00 00  |CO..PS..: ......|
0001ab90  2c 1b 00 00 ef 2c 00 00  22 1b 00 00 df 22 00 00  |,....,.."....."..|
0001aba0  2c 1b 00 00 ef 00 00 00  2b 43 00 00 4f 50 00 00  |,.......+C..OP..|
0001abb0  53 3a 00 00 20 30 00 00  00 41 00 00 54 2b 00 00  |S:.. 0...A..T+..|
0001abc0  43 4f 00 00 50 53 00 00  3d 34 00 00 2c 32 00 00  |CO..PS..=4..,2..|
0001abd0  2c 1b 00 00 eb 2c 00 00  32 0d 00 00 00 41 00 00  |,....,..2....A..|
0001abe0  54 2b 00 00 43 53 00 00  51 0d 00 00 00 2b 00 00  |T+..CS..Q....+..|
0001abf0  43 53 00 00 51 3a 00 00  20 1b 00 00 ef 2c 00 00  |CS..Q:.. ....,..|
0001ac00  1b ef 00 00 00 41 00 00  54 2b 00 00 43 52 00 00  |.....A..T+..CR..|
0001ac10  45 47 00 00 3f 0d 00 00  00 2b 00 00 43 52 00 00  |EG..?....+..CR..|
0001ac20  45 47 00 00 3a 20 00 00  1b ef 00 00 2c 1b 00 00  |EG..: ......,...|
[...]
```

Firmware analysis: strings (2)

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

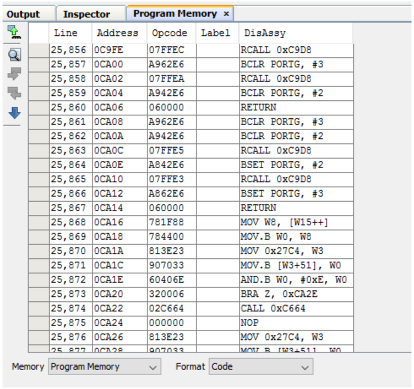
- AT+TCPCONNECT=”gsm.XXXXXXXXX.info”,60001;
- AT+TCPCONNECT=”gsm.XXXXXXXXX.info”,5555 (last number ”6” is missing);
- AT+TCPCONNECT=”91.121.XX.XX”,5555 (last number ”6” is missing).

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!

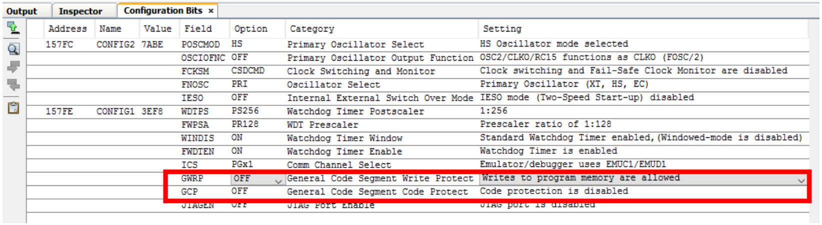


Line	Address	Opcode	Label	DisAssy
25,856	0C9FE	07FFEC		RCALL 0xC9D8
25,857	0CA00	A962E6		BCLR PORTG, #3
25,858	0CA02	07FFEA		RCALL 0xC9D8
25,859	0CA04	A942E6		BCLR PORTG, #2
25,860	0CA06	060000		RETURN
25,861	0CA08	A962E6		BCLR PORTG, #3
25,862	0CA0A	A942E6		BCLR PORTG, #2
25,863	0CA0C	07FFE5		RCALL 0xC9D8
25,864	0CA0E	A842E6		BSET PORTG, #2
25,865	0CA10	07FFE3		RCALL 0xC9D8
25,866	0CA12	A862E6		BSET PORTG, #3
25,867	0CA14	060000		RETURN
25,868	0CA16	781F88		MOV W8, [W15++]
25,869	0CA18	784400		MOV.B W0, W8
25,870	0CA1A	813E23		MOV 0x27C4, W3
25,871	0CA1C	907033		MOV.B [W3+51], W0
25,872	0CA1E	60406E		AND.B W0, #0xE, W0
25,873	0CA20	320006		BRA Z, 0xCA2E
25,874	0CA22	02C664		CALL 0xC664
25,875	0CA24	000000		NOP
25,876	0CA26	813E23		MOV 0x27C4, W3

Memory: Program Memory   Format: Code

Hardware audit tip

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



Address	Name	Value	Field	Option	Category	Setting
157FC	CONFIG2	7ABE	POSCMOD	HS	Primary Oscillator Select	HS Oscillator mode selected
			OSCIOFNC	OFF	Primary Oscillator Output Function	OSC2/CLKO/RC15 functions as CLKO (FOSC/2)
			FCKSM	CSDCMD	Clock Switching and Monitor	Clock switching and Fail-Safe Clock Monitor are disabled
			FNOSC	PRI	Oscillator Select	Primary Oscillator (XT, HS, EC)
			IESO	OFF	Internal External Switch Over Mode	IESO mode (Two-Speed Start-up) disabled
157FE	CONFIG1	3EF8	WDTPS	PS256	Watchdog Timer Postscaler	1:256
			FWPSA	PR128	WDT Prescaler	Prescaler ratio of 1:128
			WINDIS	ON	Watchdog Timer Window	Standard Watchdog Timer enabled,(Windowed-mode is disabled)
			FWDTEN	ON	Watchdog Timer Enable	Watchdog Timer is enabled
			ICS	PGx1	Comm Channel Select	Emulator/debugger uses EMUC1/EMUD1
			GWRP	OFF	General Code Segment Write Protect	Writes to program memory are allowed
			GCP	OFF	General Code Segment Code Protect	Code protection is disabled
			JTAGEN	OFF	JTAG Port Enable	JTAG port is disabled

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 $\leftrightarrow$ other peripherals
- I^2C: 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

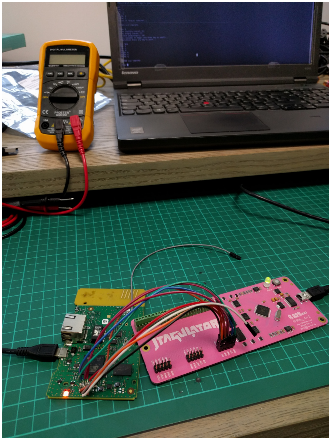
Various devices could be used to get accesses to an interface:

- The famous SEGGER JLink that works like a charm, but expensive depending on options...
- Bus pirate v3 (warning v4 not mature enough)
- BusVoodoo $\rightarrow$ supports 14 TTL/CMOS protocols
- HydraBUS $\rightarrow$ 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 $\rightarrow$ it has a costs

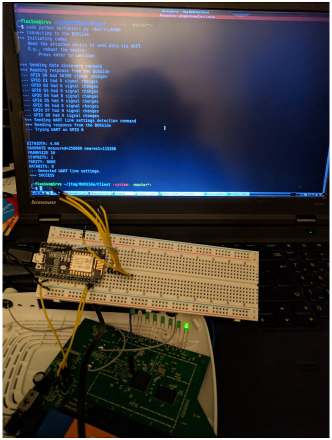
Bruteforcing JTAG and UART PINs

For almost 200€ with JTAGulator



Bruteforcing JTAG and UART PINs (2)

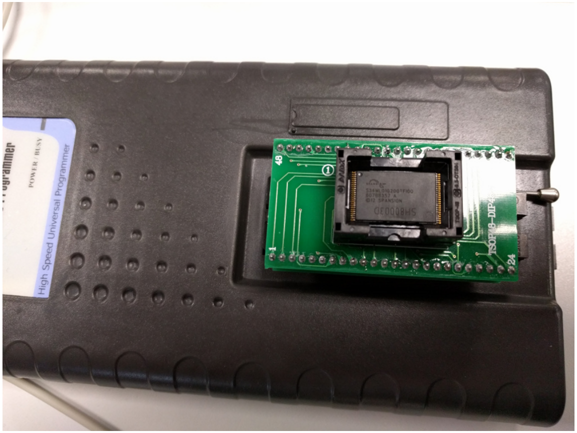
With BUSSide for almost 8€:



```
┌fluxius@irvs ~/jtag/BUSSide/Client ‹system› ‹master*›
└$ sudo python uartdetect.py /dev/ttyUSB0
+++ Connecting to the BUSSide
+++ Initiating comms
    Need the attached device to send data via UART.
    E.g., reboot the device.
        Press enter to continue.

+++ Sending data discovery command
+++ Reading response from the BUSSide
--- GPIO D0 had 56398 signal changes
--- GPIO D1 had 0 signal changes
--- GPIO D2 had 0 signal changes
--- GPIO D3 had 0 signal changes
--- GPIO D4 had 0 signal changes
--- GPIO D5 had 0 signal changes
--- GPIO D6 had 0 signal changes
--- GPIO D7 had 0 signal changes
--- GPIO D8 had 0 signal changes
+++ Sending UART line settings detection command
+++ Reading response from the BUSSide
--- Trying UART on GPIO 0


BITWIDTH: 4.00
BAUDRATE measured=250000 nearest=115200
FRAMESIZE 10
STOPBITS: 1
PARITY: NONE
DATABITS: 8
--- Detected UART line settings.
+++ SUCCESS

┌fluxius@irvs ~/jtag/BUSSide/Client ‹system› ‹master*›
└$
```

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 $\rightarrow$ wallet.fail)
- and so on.

Other targets

- Like intercoms: use of Mobile network is convenient $\rightarrow$ no wires no problem
- Overcases:
  - Deposit cases;
  - Alarms;
  - Connected cars...

Other targets

- Like intercoms: use of Mobile network is convenient $\rightarrow$ no wires no problem
- Overcases:
  - Deposit cases;
  - Alarms;
  - Connected cars...

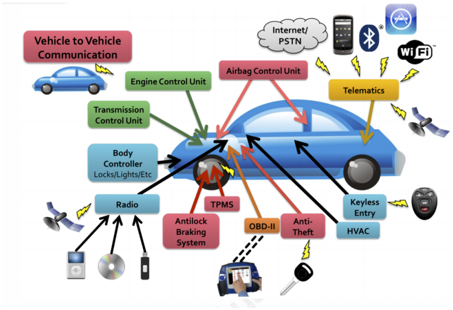
Garage hacker: the CAN bus

- ODB/ODB2 interface: a lot of interest
- Possible to interact in the CAN bus
- But too many messages are broadcasted in it $\rightarrow$ 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

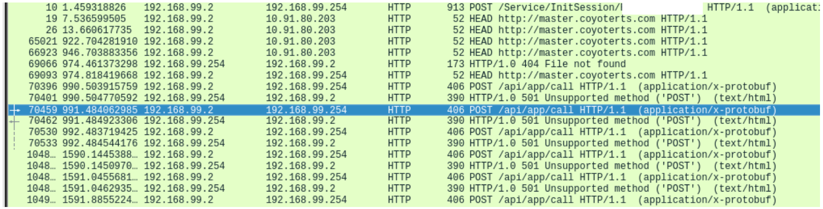
Using a BladeRF:

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 $\rightarrow$ attack the client with a fake server

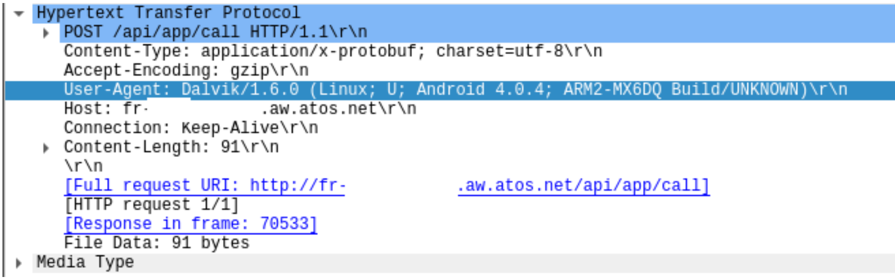
Client-side attack: new captures

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



```
   10 1.459318826   192.168.99.2      192.168.99.254    HTTP     913 POST /Service/InitSession/[redacted] HTTP/1.1  (applicat
   19 7.536599505   192.168.99.2      10.91.80.203      HTTP      52 HEAD http://master.coyoterts.com HTTP/1.1
   26 13.660617735  192.168.99.2      10.91.80.203      HTTP      52 HEAD http://master.coyoterts.com HTTP/1.1
65021 922.704281910 192.168.99.2      10.91.80.203      HTTP      52 HEAD http://master.coyoterts.com HTTP/1.1
66923 946.703883356 192.168.99.2      10.91.80.203      HTTP      52 HEAD http://master.coyoterts.com HTTP/1.1
69066 974.461373298 192.168.99.254    192.168.99.2      HTTP     173 HTTP/1.0 404 File not found
69093 974.818419668 192.168.99.2      192.168.99.254    HTTP      52 HEAD http://master.coyoterts.com HTTP/1.1
70396 990.503915759 192.168.99.2      192.168.99.254    HTTP     406 POST /api/app/call HTTP/1.1  (application/x-protobuf)
70401 990.504770592 192.168.99.254    192.168.99.2      HTTP     390 HTTP/1.0 501 Unsupported method ('POST')  (text/html)
70459 991.484062985 192.168.99.2      192.168.99.254    HTTP     406 POST /api/app/call HTTP/1.1  (application/x-protobuf)
70462 991.484923306 192.168.99.254    192.168.99.2      HTTP     390 HTTP/1.0 501 Unsupported method ('POST')  (text/html)
70530 992.483719425 192.168.99.2      192.168.99.254    HTTP     406 POST /api/app/call HTTP/1.1  (application/x-protobuf)
70533 992.484544176 192.168.99.254    192.168.99.2      HTTP     390 HTTP/1.0 501 Unsupported method ('POST')  (text/html)
1048… 1590.1445388… 192.168.99.2      192.168.99.254    HTTP     406 POST /api/app/call HTTP/1.1  (application/x-protobuf)
1048… 1590.1450970… 192.168.99.254    192.168.99.2      HTTP     390 HTTP/1.0 501 Unsupported method ('POST')  (text/html)
1048… 1591.0455681… 192.168.99.2      192.168.99.254    HTTP     406 POST /api/app/call HTTP/1.1  (application/x-protobuf)
1048… 1591.0462935… 192.168.99.254    192.168.99.2      HTTP     390 HTTP/1.0 501 Unsupported method ('POST')  (text/html)
1049… 1591.8855224… 192.168.99.2      192.168.99.254    HTTP     406 POST /api/app/call HTTP/1.1  (application/x-protobuf)
```

Client-side attack: sweets



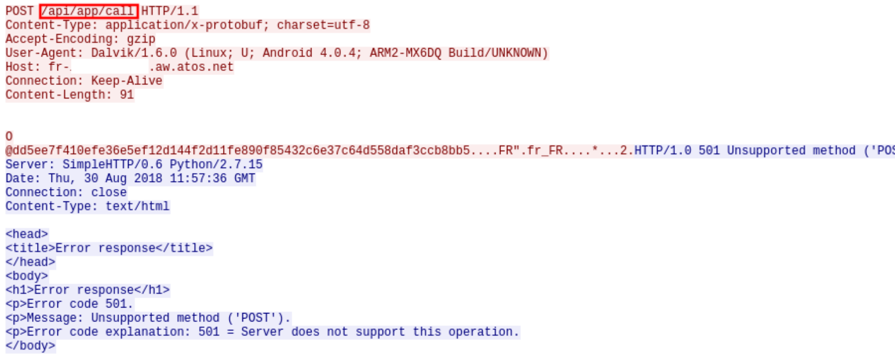
```
▾ Hypertext Transfer Protocol
  ▸ POST /api/app/call HTTP/1.1\r\n
    Content-Type: application/x-protobuf; charset=utf-8\r\n
    Accept-Encoding: gzip\r\n
    User-Agent: Dalvik/1.6.0 (Linux; U; Android 4.0.4; ARM2-MX6DQ Build/UNKNOWN)\r\n
    Host: fr-           .aw.atos.net\r\n
    Connection: Keep-Alive\r\n
  ▸ Content-Length: 91\r\n
    \r\n
    [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 $\rightarrow$ JavaScript Interface RCE
- Other request XML files $\rightarrow$ 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


O
@dd5ee7f410efe36e5ef12d144f2d11fe890f85432c6e37c64d558daf3ccb8bb5....FR".fr_FR....*...2.HTTP/1.0 501 Unsupported method ('POS
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.
<p>Message: Unsupported method ('POST').
<p>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

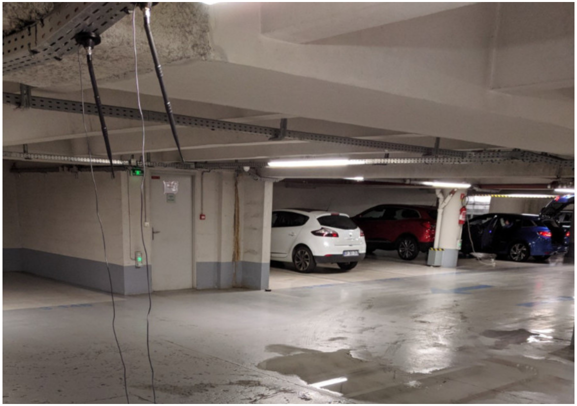
- open and close car door
- start/stop the clim
- all of these actions are authentified $\rightarrow$ OAuth, etc.
- uses HTTPS $\rightarrow$ well verified by default on new Android device

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 (expecially on external dongles)

needs to be reversed

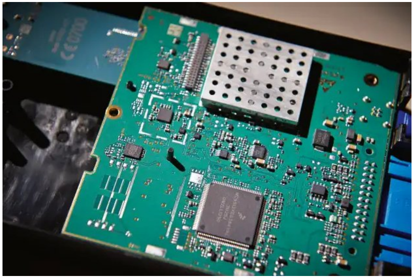
Interception in a parking station



> 10 board computers collected in the fake base station

Read more about this

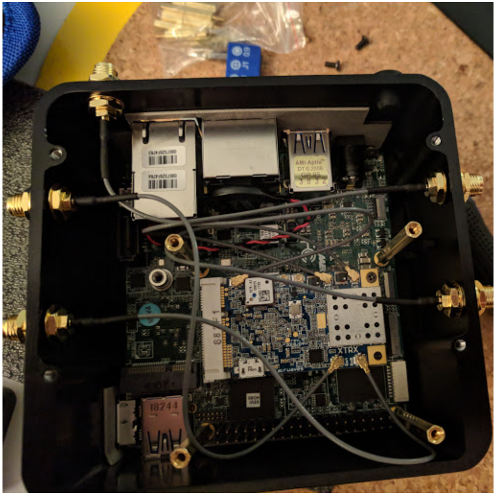
- Our blog post: Hunting mobile devices endpoints
- More stuff could be found on other systems...
- Other case: The ComboBox in BMW https://www.heise.de/ct/artikel/Beemer-Open-Thyself-Security-vulnerabilities-in-BMW-s-ConnectedDrive-2540957.html



XTRX

- mPCI-e
- perfect for embeded radio
- osmoTRX is not well supported at the moment, but patience!
- fit perfectly on APU2, UP2 and Orange PI rk3399 boards

APU2 example



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 $\rightarrow$ equipments, tools and tricks are not so expensive
- Modmobmap and Modmobjam $\rightarrow$ 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 $\rightarrow$ complementary but also a last ressort sometimes

Thanks =)

- Joffrey Czarny (@_Sn0rkY)
- Priya Chalakkal (@priyachalakkal)
- Rachelle Boissard (@rachelle_off)
- Troopers staff (@WEareTROOPERS)
- And of course $\rightarrow$ You all ;)

ANY QUESTIONS?

THANK YOU FOR YOUR ATTENTION,

SYNACKTIV
DIGITAL SECURITY