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 (@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

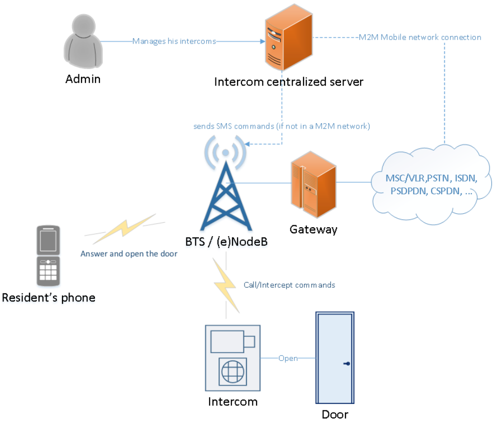
- Few reminders:
  - talk about interception techniques in practice
  - existing tools
- Our contribution:
  - feedbacks of our tests (mobile phones, intercoms, cars...)
  - tools we made (Modmobmap and Modmobjam);
  - some cheap tricks;
  - some hardware attacks.

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

Introduction

- Mobile network $\rightarrow$ more than 30 years
  - 1G: analogic, 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 $\rightarrow$ better capacity, growth services...
- Current use of the mobile network:
  - intercoms, delivery pick-up stations;
  - electric counters;
  - cameras, cars...

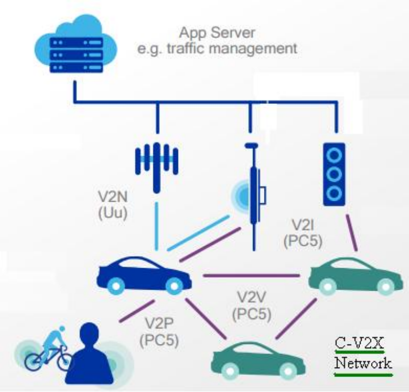
Use of mobile network with intercoms



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 → 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.

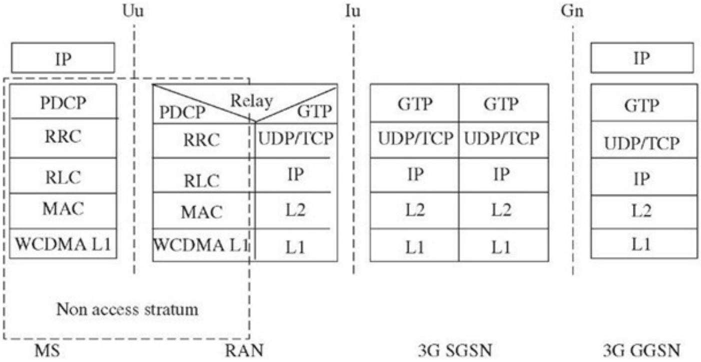
Security of communications (2)

	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

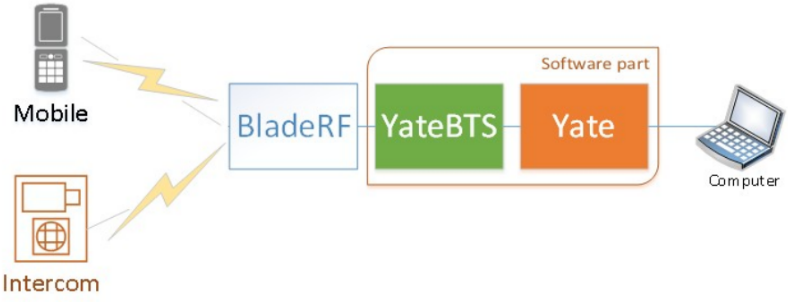
To interface to devices using the mobile network:

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

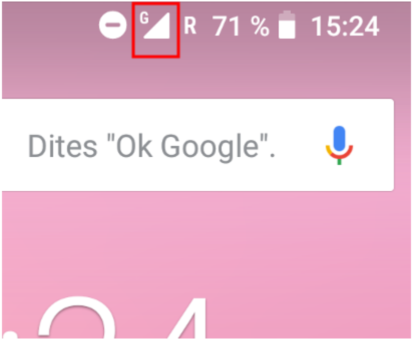
```
...
[ggsn]
DNS=8.8.8.8 8.8.4.4 ; its preferable to use your own servers for client side attacks
IP.MaxPacketSize=1520
IP.ReuseTimeout=180
IP.TossDuplicatePackets=no
Logfile.Name=/tmp/sgsn.log
MS.IP.Base=192.168.99.1
MS.IP.MaxCount=254
TunName=sgsntun
...
```

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

→ 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

Becaution with PIN auto-typing → use a SIMtrace tool to get the typed PIN

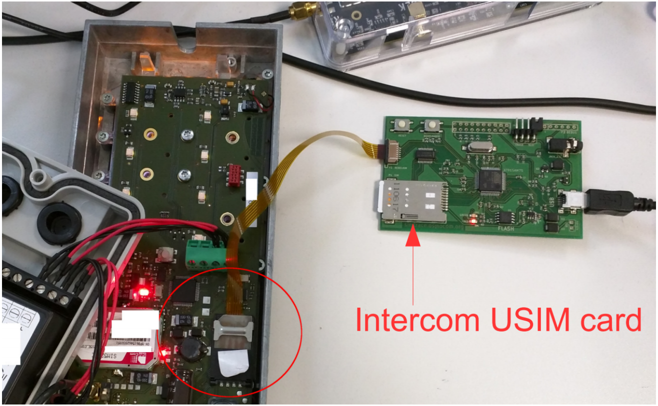
Program sysmoUSIM cards

- 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

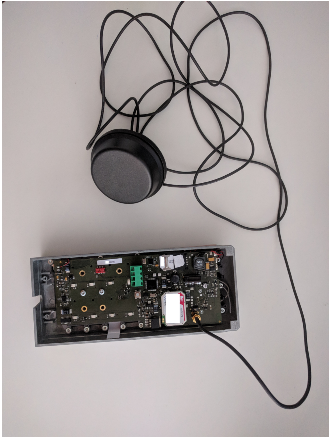
- Mostly cumbersome and expensive
- But could be improvised considering several elements:
  - Frequency;
  - Wavelength;
  - Power of reception or transmission;
  - Distance between the receiver and the transmitter.
- Cage with meshes $\rightarrow$ optimised windows against reflection of the electric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

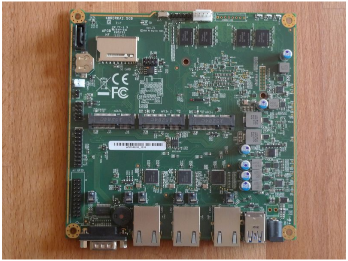
No 3G

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



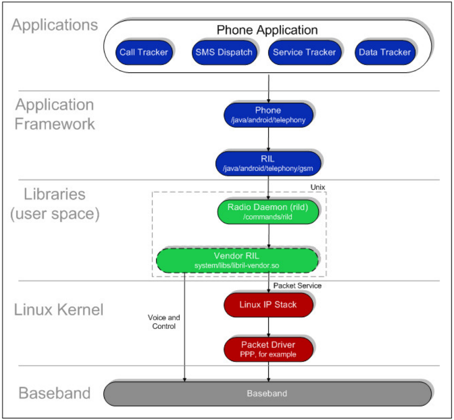
PCengines APU2

Cheaper way

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

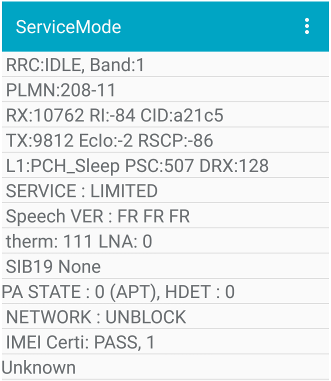
RIL on Android

- 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.



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()*

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 →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 →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
```

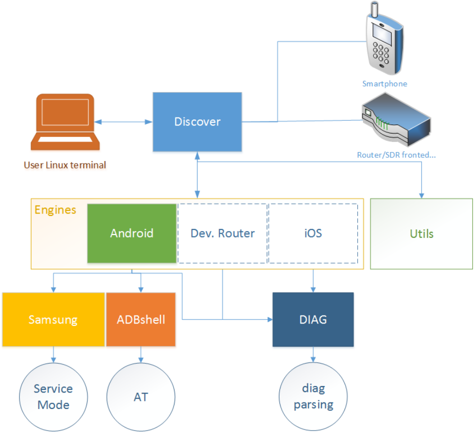
It is possible to:

- 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  (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"
    },
[...]
}
```


→ 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/chineeese 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

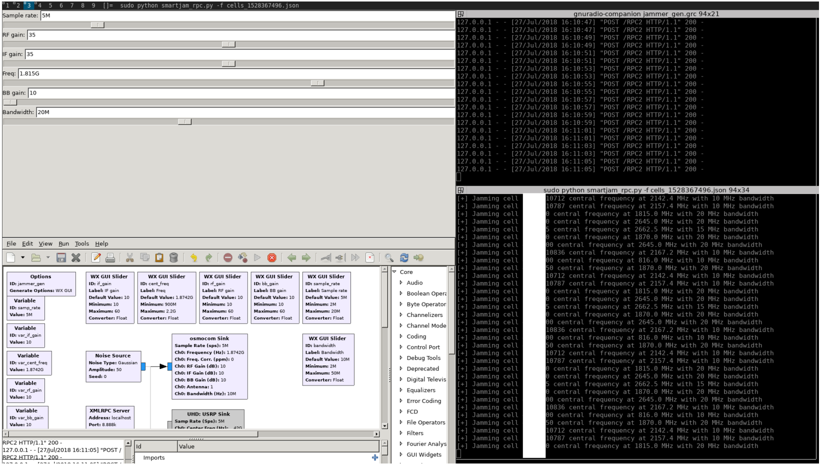
- Jam only targeted cells
- Stealth against monitors
- In 3 steps:
  1. scan cells with Modmobmap;
  2. target an operator;
  3. and jam only targeted channels;

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 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:

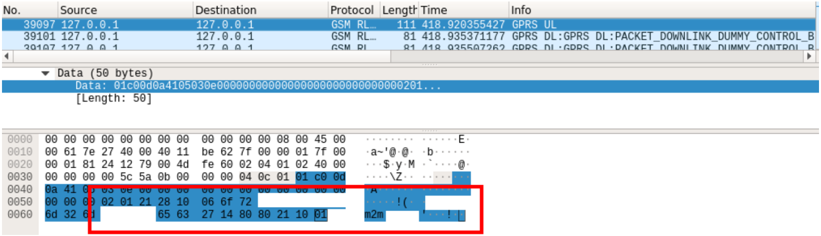
```
nipc list registered
IMSI                 MSISDN
____________________ ____________________
20801XXXXXXXXXXX     69691320681
```

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



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. .info
2	8.8.8.8	192.168.99.1	DNS	80	0.037753523	Standard query response 0x11d8 A gsm. .info A 91.
3	192.168.99.1	91.121	TCP	48	0.419114786	80 → 60001 [SYN] Seq=0 Win=16384 Len=0 MSS=1460 WS=1
4	91.121.	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.	TCP	40	0.855774038	80 → 60001 [ACK] Seq=1 Ack=1 Win=16384 Len=0
6	192.168.99.1	91.121.	TCP	117	1.120101836	80 → 60001 [PSH, ACK] Seq=1 Ack=1 Win=16384 Len=77
7	91.121.	192.168.99.1	TCP	40	1.126491129	60001 → 80 [ACK] Seq=1 Ack=78 Win=29312 Len=0
8	91.121.	192.168.99.1	TCP	60	1.129285601	60001 → 80 [PSH, ACK] Seq=1 Ack=78 Win=29312 Len=20
9	91.121.	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.	TCP	40	1.637377595	80 → 60001 [ACK] Seq=78 Ack=21 Win=16364 Len=0
11	192.168.99.1	91.121.	TCP	40	1.698825585	80 → 60001 [ACK] Seq=78 Ack=22 Win=16384 Len=0
12	192.168.99.1	91.121.	TCP	40	1.722705944	80 → 60001 [FIN, ACK] Seq=78 Ack=22 Win=16384 Len=0
13	91.121.	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 → 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("011e4d25636014006600000000000000000090000011e1540XX[...]"))
Out[8]: 320
In [9]: data = s.recv(1024)
In [10]: data
Out[11]: '2018/09/07 15:09:01\n'
```

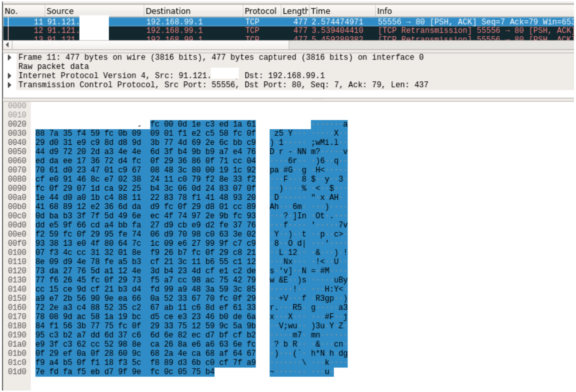
In that case: two server ports identified → 60001/tcp and 55556/tcp

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:

```
$ 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 → 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:

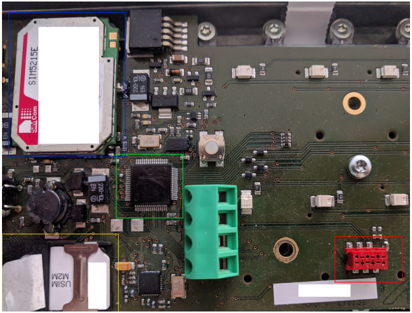
- whitelist of connections to the backend;
- use of client side certificates;

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)

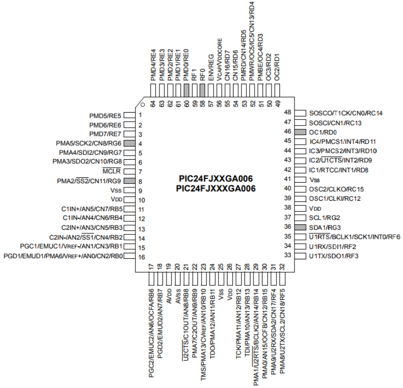


Microchip - PIC24FJ128 - GA006

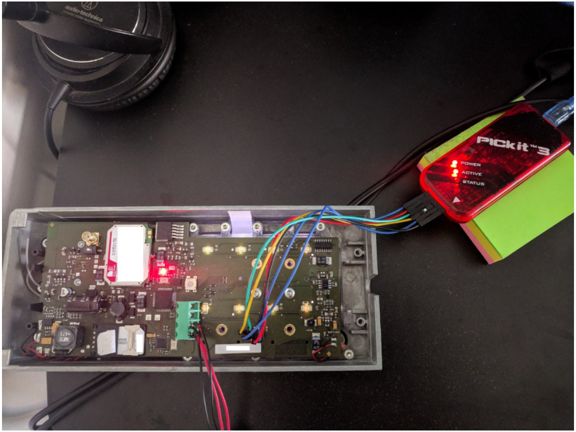
Use schematics to identify PINs via continuity tests:

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:

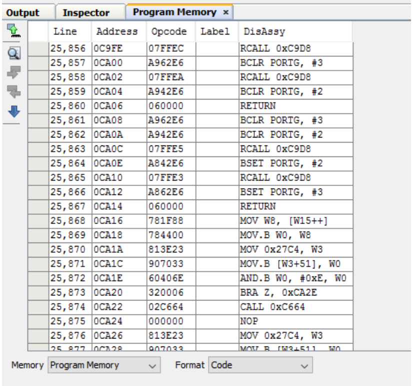
- AT+TCPCONNECT=”gsm.XXXXXXXXX.info”,60001;
- AT+TCPCONNECT=”gsm.XXXXXXXXX.info”,5555 (last number ”6” is missing);
- AT+TCPCONNECT=”91.121.XX.XX”,5555 (last number ”6” is missing).

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

Hardware audit tip

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

Output | Inspector | Configuration 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

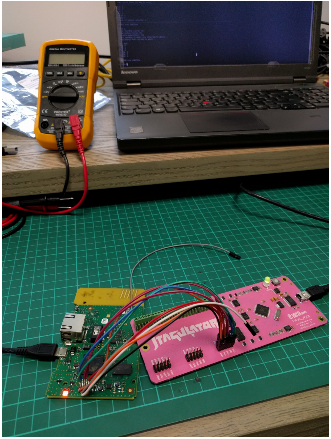
Various devices could be used to get accesses to an interface:

- The famous SEGGER JLink that works like a charm, but expensive depending on options...
- Bus pirate v3 (warning v4 not mature enough)
- BusVoodoo → 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

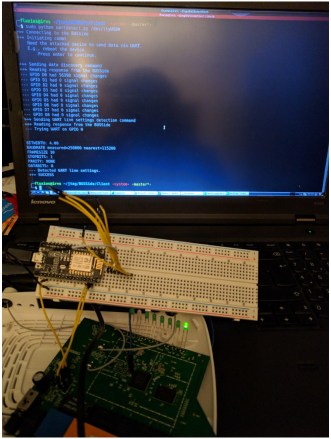
Bruteforcing JTAG and UART PINs

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 $\rightarrow$ 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...

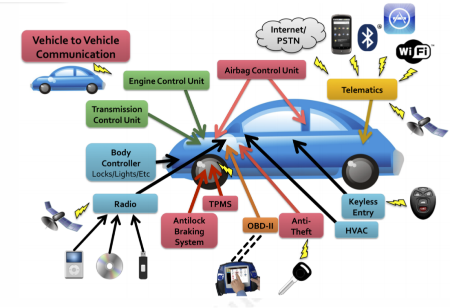
Garage hacker: the CAN bus

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


0
@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...”

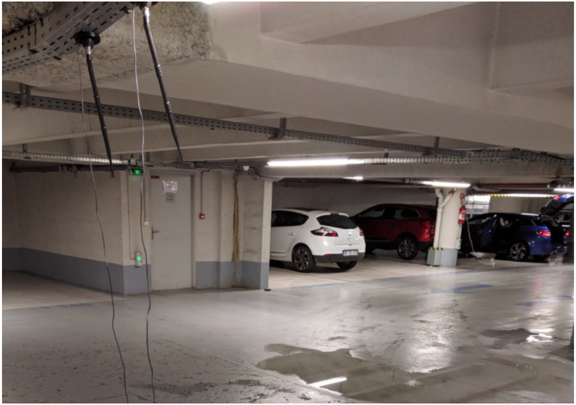
Mobile APP

- open and close car door
- start/stop the clim
- all of these actions are authentified → 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 (expecially on external dongles)

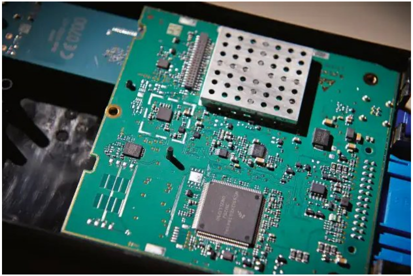
Interception in a parking station



> 10 board computers collected in the fake base station

Read more about this

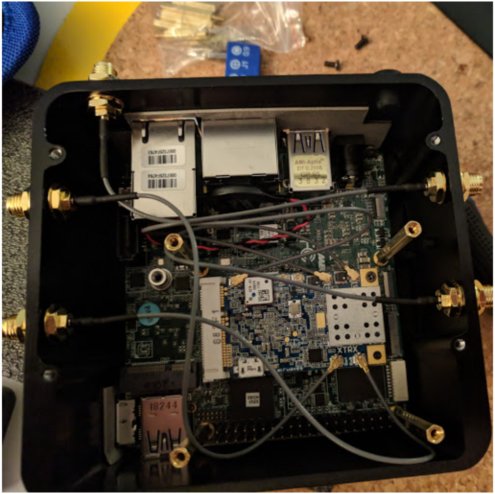
- Our blog post: Hunting mobile devices endpoints
- More stuff could be found on other systems...
- Other case: The ComboBox in BMW https://www.heise.de/ct/artikel/Beemer-Open-Thyself-Security-vulnerabilities-in-BMW-s-ConnectedDrive-2540957.html



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

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) $\rightarrow$ spotting few mistakes in slide 3
- And of course $\rightarrow$ You all ;)

ANY QUESTIONS?

THANK YOU FOR YOUR ATTENTION,

SYNACKTIV
DIGITAL SECURITY