2G

2016 – Linux hacking
bastienbaranoff.wordpress.com/2016/
Year: 2016

Interception téléphonique
Attaque de proximité passive :
Démontrée par Karsten Nohl lors de la conférence blackhat 2010. Elle utilisait un USRP (à partir de 700€) pour la partie scan de fréquence et un disque dur de 2TB pour le cassage du cryptage. Depuis qu'Elonics a produit en grande quantités des clés TNT remplissant les

conditions nécessaires pour pouvoir pratiquer pratiquer de la radio définie logicielle (RTL-SDR) on peut scanner les fréquences téléphoniques pour dix euros. Il existe d'autres

matériels intéressants pour le scan telles que les téléphones compatibles osmocom-bb ou bladeRF ou encore hackRF. Il faut savoir qu'il n'est je crois pas possible de mener l'attaque active avec les adaptateurs TNT parce que il n'ont pas la liaison haute. L'attaque passive se fait en scannant une trame particulière le keystream que l'on soumet au logiciel kraken et qui avec les 2TB de rainbow tables renvoie la ciphering key au bout d'une minute. Vous pouvez trouver le keystream de manière automatique avec le logiciel topguw (qui est obsolète vu que typhon-vx a plus de chance de trouver le bon keystream). Il y a sur mon blog la méthode pour installer topguw et pour décrypter les communications une fois que vous avez la ciphering key avec une clef TNT.Il y a aussi un lien pour la partie kraken et pour télécharger ces rainbow tables. L'attaque passive (sniffing) peut également se faire avec les téléphones compatibles osmocombb. Toujours avec ces téléphones il est aussi possible de faire l'attaque active (fakeBTS ou IMSI-catcher) il y a une image du système ci dessous ou encore la procédure d'installation est aussi disponible pour Kali rolling 2017.3. Il faut noter que les méthodes ci dessus ne marche que pour un téléphone victime en 2G pour l'attaque active et pour le chiffrement A5/1 (MYSTI) l'attaque sur le chiffrement A5/3 (KASUMI) est théoriquement possible (sandwich attack) en deux heures avec un ordinateur de bureau. En ce qui concerne la 3G on peut créer un IMSI-catcher 2.0 à condition d'avoir un accès SS7. Un autre type d'attaque sur la 3G est possible : par downgrade de la 3G vers un fakeBTS 2G par une SDR faisant un déni de service sur le réseau 3G du téléphone victime. Ensuite il est possible de cloner une carte sim à distance à l'aide d'un silent sms envoyé sur le téléphone victime qui renvoie un hash de la signature cryptographique que l'on peut retrouver avec des rainbow tables adéquates ou un cluster de FPGA. Enfin l'accès au réseau SS7 permet lui aussi normalement de hacker les téléphones à distance simplement à l'aide d'un numéro.

Intercept your own GSM signal with RTL SDR
In this tutorial I use Kali 2.0 Sana (it should work with debian too) and a rtl-sdr dongle. Now if have them, you can open up a shell...
First step : Dependencies

```
# sudo apt-get -y install git-core cmake g++ python-dev swig \
pkg-config libfftw3-dev libboost-all-dev libcppunit-dev libgsl0-dev \
libusb-dev libsdl1.2-dev python-wxgtk3.0 python-numpy \
python-cheetah python-lxml doxygen libxi-dev python-sip \
libqt4-opengl-dev libqwt-dev libfontconfig1-dev libxrender-dev \
python-sip python-sip-dev gnuradio libtalloc-dev libpcsclite-dev
```

Second step : Libosmocore

```
# cd /root
# git clone git://git.osmocom.org/libosmocore.git
# cd libosmocore
# autoreconf -i
# ./configure –prefix=/root
# make
# make install
# ldconfig
```
Third step : Airprobe and zmania patch :
```
# cd /root
# git clone git://github.com/scateu/airprobe-3.7-hackrf-patch
# git clone git://github.com/ksnieck/airprobe
# cp /root/airprobe-3.7-hackrf-patch/zmiana.patch /root/airprobe/zmiana.patch
# cd airprobe
# export PKG_CONFIG_PATH=/root/lib/pkgconfig
# patch -p1 < zmiana.patch
# cd gsm-receiver
# ./bootstrap
# autoreconf -i
# ./configure –prefix=/root
# make
# make install
# ldconfig
# cd ../gsmdecode
# ./bootstrap
# ./configure –prefix=/root
# make
# make install
# ldconfig
```
Last step of install on PC : Java RE and Topguw
Download JRE
```
# cd /root
# git clone git://github.com/bastienjalbert/topguw
```

Now you can run topguw by typing
```
# cd topguw
# cd dist
# ‘path_to_your_jre/bin/java’ -jar topguw_git.jar
```
Now you have two choices you can download 1.6TB of rainbow tables to feed kraken and find kc (ciphering key) close to your antenna or if you have mediatek devices (wiko[Confirmed with wiko Lenny], HTC, Huawei[Confirmed with the Y330-U01]...) and if you are rooted you can retrieve your kc by opening a shell on your phone and type
AT+CRSM COMMANDS
USIM SUPPORT
```
$ su
# cat /dev/radio/atci1 &
# kc () {
echo -e “AT+CRSM=176,20256,0,0,9\r” > /dev/radio/atci1
}
```

SIM SUPPORT
AT+CRSM=176,28448,0,0,9
Now on your terminal emulator you just have to type kc to retrieve your kc, do not forget to kill cat process when you have finish by typing kill -9 2345 when 2345 is your process id.
You have to know your ARFCN to retrieve your frequency. To do that download MTK engineering mode on Google play. Open it, in telephony > Go to network selecting and choose

GSM only, then go to network info sélect RR Meas Rep and then check information you will see your ARFCN.
Exploit
Run topguw, start sniff, select your frequency corresponding to your ARFCN. Send à SMS to yourself. Stop sniffing check go.sh configuration choosen by topguw.
Open wireshark, listen to interface lo.
Then open a shell and type

```
# cd /root/airprobe/gsm-receiver/src/python
# ./go.sh capture.cfile 64 0B 1234567890ABCDEF
```

where capture.cfile is your capture generated by topguw, 64 is thé decimation rate of any RTL-SDR, 0B is the configuration found in topguw and 1234567890ABCDEF is your kc.
At this step you should be able to see GSM-SMS in wireshark and read your sms . Hope this help…

Osmocom-bb (IMSI catcher) + Prebuilt VM
In this post we will see how to make a base station with two motorola compatible phones C 115/118/123 and two cables usb serial jack 2.5mm PL2303

OS: kali rolling 2019,1

First step : build the toolchain

```
# nano /etc/apt/sources.list
```

add deb http://old.kali.org kali/sana main non-free contrib

```
# apt install gcc-4.9 g++-4.9
# nano /etc/apt/sources.list
```

comment kali sana

```
# apt-get update && apt-get upgrade
# apt-get install build-essential libgmp-dev libx11-6 libx11-dev flex
libncurses5 libncurses5-dev libncursesw5 libpcsclite-dev zlib1g-dev libmpfr4
libmpc3 lemon aptitude libtinfo-dev libtool shtool autoconf git-core pkg-
config make libmpfr-dev libmpc-dev libtalloc-dev libfftw3-dev libgnutls28-dev
libssl1.0-dev libtool-bin libxml2-dev sofia-sip-bin libsofia-sip-ua-dev sofia-
sip-bin libncursesw5-dev libncursesw5-dbg bison libgmp3-dev alsa-oss
# update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-4.9 10
# update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-7 20
# update-alternatives --install /usr/bin/g++ g++ /usr/bin/g++-4.9 10
# update-alternatives --install /usr/bin/g++ g++ /usr/bin/g++-7 20
# update-alternatives --install /usr/bin/cc cc /usr/bin/gcc 30
# update-alternatives --set cc /usr/bin/gcc
# update-alternatives --install /usr/bin/c++ c++ /usr/bin/g++ 30
# update-alternatives --set c++ /usr/bin/g++
# update-alternatives --config gcc
# update-alternatives --config g++
(choose 4.9)
# apt remove texinfo
# cd /root
# wget http://ftp.gnu.org/gnu/texinfo/texinfo-4.13.tar.gz
# gzip -dc < texinfo-4.13.tar.gz | tar -xf -
# cd texinfo-4.13
# ./configure
# make
# make install
# git clone https://github.com/axilirator/gnu-arm-installer.git gnuarm
# cd gnuarm
```
Run this scripts:
```
# ./download.sh
# ./build.sh
# export PATH=$PATH:/root/gnuarm/install/bin
```
Now you have cross-compiler ready you can build osmocom with your firmware

```
# cd /root
# git clone git://git.osmocom.org/libosmocore.git
# cd libosmocore
# autoreconf -i
# ./configure
# make
# make install
# ldconfig
# cd ..
# git clone git://git.osmocom.org/libosmo-dsp.git
# cd libosmo-dsp
# autoreconf -i
# ./configure
# make
# make install
# cd ..
# git clone https://github.com/osmocom/osmocom-bb trx
# git checkout jolly/testing
# cd src
# nano target/firmware/Makefile
```
It needs TX support Just uncomment ‘CFLAGS += -DCONFIG_TX_ENABLE
```
# make HOST_layer23_CONFARGS=--enable-transceiver
```
https://youtu.be/RIXVYi3dHc4

16/20

```
# cd /root
#git clone https://github.com/bastienbaranoff/imsi-catcher
```
Asterisk version (1.8.13.1) :
```
# nano /etc/apt/sources.list
```
comment kali rolling and add
```
deb http://old.kali.org/kali moto main non-free contrib
# apt update
# apt install asterisk-dev
```

```
# wget https://downloads.asterisk.org/pub/telephony/asterisk/old-
releases/asterisk-1.8.13.1.tar.gz && gzip -dc < asterisk-1.8.13.1.tar.gz |

tar -xf -
# cd /root/asterisk-1.8.13.1
# nano /root/asterisk-1.8.13.1~dfsg1/main/tcptls.c
```
ctrl-W SSLv3
and change cfg->ssl_ctx = SSL_CTX_new(SSLv3_client_method()), by cfg->ssl_ctx = SSL_CTX_new(SSLv23_client_method());
```
#./configure CXX=g++-4.9 CC=gcc-4.9
```

```
# make
# make install
# nano /etc/apt/sources.list
```

comment kali moto and uncomment kali rolling

```
# apt-get update
# apt-get install osmocom-nitb osmo-bts
```

Download open-core-amr

```
# tar xvzf opencore-amr-0.1.5.tar.gz
# cd opencore-amr-0.1.5
# ./configure
# make
# sudo make install
# sudo ldconfig
```

mISDN

```
# rm -Rf /lib/modules/$(uname -r)/kernel/drivers/isdn/hardware/mISDN
# rm -Rf /lib/modules/$(uname -r)/kernel/drivers/isdn/mISDN/
# depmod -a

# apt-get install git build-essential libtool autoconf automake linux-headers-
4.15.0-kali2-all-amd64

# git clone https://github.com/b1-systems/mISDN/
# git clone https://github.com/b1-systems/mISDNuser/
#git clone https://github.com/bbaranoff/osmocombb-ansible
# cd mISDN
# cp /root/osmocombb-ansible/mISDN.patch mISDN.patch
# patch -p1 < mISDN.patch
```

As of Debian 8.5, there is an automake version mismatch, fix it via:

```
# aclocal && automake --add-missing
# ./configure
# cp /root/osmocombb-ansible/mISDN.cfg.default standalone/mISDN.cfg
# make modules
# make modules_install
# depmod -a
# cd ../mISDNuser # make
# ./configure
# make
# make install
# cd example
# make
# cd
# git clone https://github.com/fairwaves/lcr
# cd lcr
# autoreconf -i
# ./configure --with-sip --with-gsm-bs --with-gsm-ms --with-asterisk
# make
# make install
# ldconfig
# cp chan_lcr.so /usr/lib/asterisk/modules/
# cd ../imsi-catcher
```

Place ~/imsi-catcher/asterisk folder in /etc
Place interface.conf, routing.conf and options.conf folder in /usr/local/etc/lcr
Place osmo-bts.cfg and open-bsc.cfg in /root/.osmocom
Change in /etc/asterisk/sip.conf with your sip provider login and pass (ex

diamondcard)
```
# apt-get install alsa-oss
# modprobe snd_pcm_oss
# modprobe snd_mixer_oss
# modprobe mISDN_core
# modprobe mISDN_dsp
```
RUNNING !!!

18/20

First search strong rssi
```
# cd trx/src/
# sudo host/osmocon/osmocon -m c123xor -p /dev/ttyUSB0 -c
target/firmware/board/compal_e88/rssi.highram.bin
```
Ctrl-C remove and put the battery

Shell #1
```
# cd trx/src/
# host/osmocon/osmocon -m c123xor -p /dev/ttyUSB0 -s /tmp/osmocom_l2 -c
target/firmware/board/compal_e88/trx.highram.bin -r 99
```
Shell #2
```
# cd trx/src/host/osmocon/osmocon -m c123xor -p /dev/ttyUSB1 -s /tmp/osmocom_l2.2 -c
target/firmware/board/compal_e88/trx.highram.bin -r 99
```
Shell #3
```
# cd trx/src/host/layer23/src/transceiver/
# sudo ./transceiver -a [YOUR ARFCN FOUND WITH RSSI] -2 -r 99
```
Shell #4
```
# osmo-nitb -c ~/.osmocom/open-bsc.cfg -l ~/.osmocom/hlr.sqlite3 -P -m -C --
debug=DRLL:DCC:DMM:DRR:DRSL:DNM
```
Shell #5
```
# lcr start
```
Shell #6
```
# osmobts-trx -c ~/.osmocom/osmo-bts.cfg -r 99
```
Shell #7
```
#asterisk
#asterisk -rvvvvvv
```
if you use FTDI cable you have to modify osmocon command by
known problems. In some situations (like, apparently, using FTDI serial cables), you might need the -m c123 mode for your MotorolaC123 instead of the normal -m c123xor see THIS
To make your imsi catcher work you have to change Location Area Code LAC in ~/.osmocom/open-bsc.cfg to fit with a LAC near you you can find it in RSSI app

Where to find Osmocombb compatible phone
I want to thank George Tsinikosmaog to have gived to me some stuff so I want to help to sell his product you can find some phone compatible with my blog on ebay on this address for example uberwaves you can contact him at uberwaves@gmail.com. I think he is a person of trust and he can sell you all the things needed to test my IMSI-catcher VM. Hope this can help all osmocombb community

3G CONNECT

This article tells you how to make a 3G connection by using OpenBTS-UMTS and LimeSDR-Mini. At this time connection have been realised but no data work for the moment keep you

in touch...
Install the uhd driver
```
#git clone https://github.com/EttusResearch/uhd.git
```

```
# cd uhd
# git checkout UHD-3.10
# sudo apt-get install libboost-all-dev libusb-1.0-0-dev python-mako doxygen
python-docutils cmake build-essential
# cd host
# mkdir build; cd build
# cmake ..
# make -j4
# make install
# ldconfig
```

Install Libosmocore

```
# cd /root
# git clone git://git.osmocom.org/libosmocore.git
# cd libosmocore
# autoreconf -i
# ./configure
# make
# make install
# ldconfig
# cd ..
```

Install osmo-trx

```
# cd /root
# git clone git://git.osmocom.org/osmo-trx.git
# cd osmo-trx
# autoreconf -i
# ./configure --with-lms
# make
# make install
# ldconfig
# cd ..
make -j4
# make install
# ldconfig
```

SoapySDR and SoapyUHD

```
# git clone https://github.com/pothosware/SoapySDR
# apt-get install cmake g++ libpython-dev python-numpy swig
# mkdir build
# cd build
# cmake ..
# make -j4
# sudo make install
# sudo ldconfig #needed on debian systems
# SoapySDRUtil --info
# git clone https://github.com/pothosware/SoapyUHD
# cd SoapyUHD
# mkdir build && cd build
# cmake ..
# make
# make install
# ldconfig
```

LimeSuite

```
# sudo apt-get install git g++ cmake libsqlite3-dev libi2c-dev libusb-1.0-0-
dev
```

```
#install graphics dependencies
sudo apt-get install libwxgtk3.0-dev freeglut3-dev
# git clone https://github.com/myriadrf/LimeSuite
# cd LimeSuite
# cd build
# cmake ..
Install Apache Thrift
# apt install composer phpunit
# gem update
# cd
# git clone https://github.com/apache/thrift
# cd thrift
# ./bootstrap.sh
# ./configure
# make
# make install
# ldconfig
Install gnuradio
# nano /etc/apt/sources.list
add deb http://deb.debian.org/debian unstable main contrib non-free
# apt update
# apt install gcc-5 g++-5
# update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-5 15
# update-alternatives --install /usr/bin/g++ g++ /usr/bin/g++-5 15
# update-alternatives --install /usr/bin/cc cc /usr/bin/gcc 30
# update-alternatives --set cc /usr/bin/gcc
# update-alternatives --install /usr/bin/c++ c++ /usr/bin/g++ 30
# update-alternatives --set c++ /usr/bin/g++
# update-alternatives --config gcc
# update-alternatives --config g++
and choose gcc and g++ 5
# apt -y install git cmake g++ python-dev swig \
pkg-config libfftw3-dev libboost-all-dev libcppunit-dev libgsl-dev \
libusb-dev libsdl1.2-dev python-wxgtk3.0 python-numpy python-cheetah \
python-lxml doxygen libxi-dev python-sip libqt4-opengl-dev libqwt-dev \
libfontconfig1-dev libxrender-dev python-sip python-sip-dev python-qt4 \
python-sphinx libusb-1.0-0-dev libcomedi-dev libzmq3-dev python-mako
# git clone --recursive git://git.gnuradio.org/gnuradio
# cd gnuradio
# mkdir build
# cd build
# cmake ../
# make -j4
# sudo make install
Installation of gr-osmosdr
# git clone https://git.osmocom.org/gr-osmosdr
# cd gr-osmosdr
# mkdir build
```

```
# cd build
# cmake ..
# make
Installation of gr-gsm
# git clone https://git.osmocom.org/gr-gsm
# cd gr-gsm
# mkdir build
# cd build
# cmake ..
# mkdir $HOME/.grc_gnuradio/ $HOME/.gnuradio/
# make
```
OpenBTS-UMTS
```
apt install libreadline7 libreadline-dev libosip2-dev libortp9 libortp-dev
libzmq5 libzmq3-dev libtool-bin
# pip install zmq
# git clone https://github.com/RangeNetworks/OpenBTS-UMTS
# cd OpenBTS-UMTS
# git submodule init
# git submodule update
# update-alternatives --config gcc
# update-alternatives --config g++
```
and choose gcc and g++ 5
```
# git clone https://github.com/RangeNetworks/libcoredumper
# cd libcoredumper
# ./build.sh
# cd coredumper-1.2.1
# autoreconf -i
# ./configure
# make -j4
# make install
# ldconfig
# tar zxvf asn1c-0.9.23.tar.gz
# cd vlm-asn1c-0959ffb/
# nano configure.ac
```
change
```
AM_INIT_AUTOMAKE=([-Wall -Werror foreign])
```
to
```
AM_INIT_AUTOMAKE=([-Wall foreign])
# autoreconf -i
# ./configure
# make -j4
# make install
# ldconfig
# cd ~/OpenBTS-UMTS
# git clone https://github.com/bbaranoff/OpBTS-LimeMini
# cd TransceiverUHD
# patch -p0 < ../OpBTS-LimeMini/OpBTS1.patch
# patch -p0 < ../OpBTS-LimeMini/OpBTS2.patch
$ sudo NodeManager/install_libzmq.sh
$ ./autogen.sh
$ ./configure
$ make
$ sudo make install
```
To setup OpenBTS-UMTS, we may need to modify the settings such as ARFCN, DNS, Firewall. Navigate to ~/OpenBTS-UMTS/apps folder and open OpenBTS-UMTS.example.sql file with the preferable text editor. You may need to edit following options:
'GGSN.DNS' set to '8.8.8.8' to enable Google DNS.
'GGSN.Firewall.Enable' set to '0' to disable Firewall.
'UMTS.Radio.Band' – set the band you are going to use. Select from 850, 900, 1700, 1800, 1900 or 2100.
'UMTS.Radio.C0' – set the UARFCN. Range of valid values depend upon the

selected operating band. Please, ensure that you are using free operating band.
'UMTS.Radio.RxGain' "35" To avoid Overpower
Now, we need to create folders and working database:

```
$ sudo mkdir /var/log/OpenBTS-UMTS
$ cd /etc/OpenBTS
$ sudo sqlite3 /etc/OpenBTS/OpenBTS-UMTS.db ".read OpenBTS-UMTS.example.sql"
$ sudo cp TransceiverUHD/transceiver ~/OpenBTS-UMTS/
$ sudo cp TransceiverUHD/transceiver apps
```

We also need to setup forwarding in iptables to properly forward data between devices, host machine, and the Internet:

```
$ iptables -t nat -A POSTROUTING -j MASQUERADE -o eth0
$ echo 1 > /proc/sys/net/ipv4/ip_forward
```

If you have Internet connection through the another interface (for example, Wi-Fi), you need to change eth0 to the applicable one (i.e., wlan0). Please note, that you need to setup forwarding in iptables every time after you computer rebooted.
It's important to install Subscriber Registry API and SIP Authentication Server to be able to launch OpenBTS-UMTS. Subscriber Registry controls database of subscriber information and in fact works as HLR (Home Location Registry):

```
# git clone https://github.com/RangeNetworks/subscriberRegistry.git
# cd subscriberRegistry
# git submodule init
# git submodule update
# autoreconf -i
# ./configure
# make
# make install
# mkdir /var/lib/asterisk/
# mkdir /var/lib/asterisk/sqlite3dir/
# cp apps/comp128 ~/OpenBTS-UMTS/
# cp apps/comp128 ~/OpenBTS-UMTS/apps/
# cp apps/comp128 /OpenBTS
```

RUNNING !!!

```
# cd ~/OpenBTS-UMTS/apps
# ./OpenBTS-UMTS
# cd subscriberRegistry/apps
# ./sipauthserve
```

To add the subscriber to the registry, you need to know IMSI and K_i value of your programmable SIM card. There are two common ways to get the values. The only way to use SIMs from another provider is to obtain the K_i through a roaming interface to the provider's HLR/HSS. The second way is to buy test SIM-card and flash it using the programmer and pysim utility (http://cgit.osmocom.org/pysim/). Navigate to ~/subscriberRegistry/NodeManager/ and execute:

```
# git clone https://github.com/osmocom/pySim
# cd pySim
# apt-get install python-pyscard python-serial python-pip
pip install pytlv
# ./pySim-prog.py -j 0
> Name : Magic
> SMSP : e1ffffffffffffffffffffffff0581005155f5ffffffffffff000000
> ICCID : 8901901550000000005
> MCC/MNC : 901/55
> IMSI : 901550000000000
> Ki : bef0ba3847947b0a6fdd5bee6759931a
> OPC : 62d77be45c4cc0636a19a870f20b228b
> ACC : None
```

(# sudo ./nmcli.py sipauthserve subscribers create “name” imsi msisdn ki
Values imsi, msisdn and ki should be taken from your SIM-card.)

```
# python2.7 ./nmcli.py sipauthserve subscribers create "Magic"
IMSI901550000000000 1234567 bef0ba3847947b0a6fdd5bee6759931a
```

Now, connect your phone to the network and test 3G. That’s it!
When the connection realise you should see this

EDIT FOR AT THIS TIME PHONE CONNECT AND WITH AN APN THINGS HAPPEN BUT NO DATA AND THIS ERROR PLEASE COMMENT IF YOU HAVE A SOLUTION

```
OpenBTS-UMTS: TurboCoder.cpp:452: TurboInterleaver::TurboInterleaver(int): Assertion `K
>= 40 && K <= 5114' failed.
Abandon
```

4G srsLTE

```
# sudo apt-get install cmake libfftw3-dev libmbedtls-dev libboost-program-options-dev
libconfig++-dev libsctp-dev
```

RF front-end driver:
UHD: https://github.com/EttusResearch/uhd
SoapySDR: https://github.com/pothosware/SoapySDR
SoapyUHD: https://github.com/pothosware/SoapyUHD
Limesuite https://github.com/myriadrf/Limesuite

```
git clone https://github.com/srsLTE/srsLTE.git
cd srsLTE
mkdir build
cd build
cmake ../
make
make test
make install
srslte_install_configs.sh user
nano /root/.config/srslte/enb.conf
```

Change [rf] section

```
[rf]
dl_earfcn = 3050
tx_gain = 56
rx_gain = 38
device_name = soapy
device_args = rxant=LNAH,txant=BAND2
time_adv_nsamples = 70
burst_preamble_us = 0
```

edit /root/.config/srslte/user_db.csv with your sim card value
running

```
sudo srsenb
sudo srsepc
```

Note: The UE database file(user_db.csv) has to be within the same directory where you would run the srsepc command.

Modmobmap
Patched https://github.com/bbaranoff/Modmobmap

Modmobmap is a tool aimed to retrieve information of cellular networks.
As shown in the first

(https://www.rump.beer/2018/slides/modmobmap.pdf), this tool is able to retrieve information of 2G, 3G, 4G and more cellular network types with minimum requierement: only a phone with ServiceMode.

For the moment, the tool has only been tested and developped for the following devices:
– Samsung Galaxy S3 via [xgoldmon (Modmobmap's edition)](https://github.com/FIUxIuS/xgoldmon);
– Samsung Galaxy S4;
– Samsung Galaxy S5;
– Samsung Galaxy Note 2 with LTE;

Moreover, as it's compatible for XGold via Modmobmap's forked of *xgoldmon*, this tools should also be able to work with devices supported by *xgoldmon* as well:
– Samsung Galaxy S4 GT-I9500 (this is the version without LTE!)
– Samsung Galaxy Nexus GT-I9250 (has to be rooted!)
– Samsung Galaxy S2 GT-I9100
– Samsung Galaxy Note 2 GT-N7100

Note that all devices should be rooted. In any other case, you will have to use the DFR technique by hand!

Also: Patches, or engines, for other devices are very much welcomed! 😉

Requirements

Here are the following requirements:
– Python 2 or 3;
– Last Android SDK to run ADB: https://developer.android.com/studio/#downloads;
– A compatible mobile phone;
– A valid/unvalid SIM card (just in case to provide an IMSI number).

How to use

The tool is provided with a quick help that shows you the required argument as follows:

“`

```
python modmobmap.py -h
usage: modmobmap.py [-h] [-m MODULE] [-n NETWORKS] [-o] [-s ANDROIDSDK]
[-a ATMODE] [-f FILE]

Mobile network mapping tool with cheap equipments

optional arguments:
-h, –help show this help message and exit
-m MODULE, –module MODULE
Module to use (e.g: “servicemode” by default)
-n NETWORKS, –networks NETWORKS
Networks in MCCMNC format splitted with commas
-o, –cached_operator
Use operator in cache to speed up the passive scan
-s ANDROIDSDK, –sdk ANDROIDSDK
Android SDK path
-a ATMODE, –at ATMODE
```

```
AT access mode. If host put something like
“/dev/ttyUSBxx. By default it uses ADB.”
-f FILE, –file FILE File to parse. For the moment it could be used in
combination with AT mode host.
```
“`

Assuming the Android SDK is installed in */opt/Android*, the tool can be quickly started as follows:

“`
```
$ sudo python modmobmap.py
=> Requesting a list of MCC/MNC. Please wait, it may take a while…
Found 2 operator(s)
{u’20810′: u’F SFR’, u’20820′: u’F-Bouygues Telecom’}
[+] Unregistered from current PLMN
[+] New cell detected [CellID/PCI-DL_freq (4XXX-81)]
Network type=2G
PLMN=208-10
ARFCN=81
[+] New cell detected [CellID/PCI-DL_freq (6XXXXXX-2950)]
Network type=3G
PLMN=208-20
Band=8
Downlink UARFCN=2950
Uplink UARFCN=2725
[+] New cell detected [CellID/PCI-DL_freq (3XX-6300)]
Network type=4G
PLMN=208-10
Band=20
Downlink EARFCN=6300
[+] New cell detected [CellID/PCI-DL_freq (3XX-2825)]
Network type=4G
PLMN=208-10
Band=7
Downlink EARFCN=2825
[+] New cell detected [CellID/PCI-DL_freq (3XX-1675)]
Network type=4G
PLMN=208-10
Band=3
Downlink EARFCN=1675
[…]
```
“`

Note: If the Android SDK is installed anywhere else, you can use the *-s* parameter to specify its directory.

Speed-up the passive scan
—————————

When looking for operators, an AT command is sent to the modem. If you want to speed-up the scanning, you can hardcoded the operators to the following file `cache/operators.json`:

“`
```
{
“20801”: “Orange”,
“20810”: “F SFR”,
“20815”: “Free”,
“20820”: “F-Bouygues Telecom”
}
```
“`

Only the MCC/MNC codes are inmportant. Then you can re-launch the tool as follows:

“`
```
$ sudo python modmobmap.py -o
=> Requesting a list of MCC/MNC. Please wait, it may take a while…
Found 4 operators in cache, you choose to reuse them.
Found 4 operator(s)
{u’20810′: u’F SFR’, u’20820′: u’F-Bouygues Telecom’, u’20815′: u’Free’, u’20801′: u’Orange’}
[+] Unregistered from current PLMN
[+] New cell detected [CellID/PCI-DL_freq (XXXX-10614)]
Network type=3G
PLMN=208-10
Band=1
Downlink UARFCN=10614
Uplink UARFCN=9664
[…]
[+] New cell detected [CellID/PCI-DL_freq (XXX-3501)]
Network type=4G
PLMN=208-20
Band=8
Downlink EARFCN=3501
[…]
[+] Unregistered from current PLMN
=> Changing MCC/MNC for: 20815
[+] New cell detected [CellID/PCI-DL_freq (XXX-2825)]
Network type=4G
PLMN=208-15
Band=7
Downlink EARFCN=2825
[…]
=> Changing MCC/MNC for: 20801
[+] New cell detected [CellID/PCI-DL_freq (XXXXX-3011)]
Network type=3G
PLMN=208-1
Band=8
Downlink UARFCN=3011
Uplink UARFCN=2786
[…]
```
“`

Note we have been able to detect other cells the AT command *AT+COPS* did not returned.

A complet list of MCC and MNC codes could be retrieved anywhere on internet and in Wikipedia:
https://en.wikipedia.org/wiki/Mobile_country_code

Focusing some operators

It is possible to tell *Modmobmap* to focus only on specific operators with the *-m* argument:

“`
```
$ sudo python modmobmap.py -n 20801
```

```
=> Manual MCC/MNC processing…
Found 1 operator(s)
{‘20801’: ‘20801’}
[…]
=> Changing MCC/MNC for: 20801
[+] New cell detected [CellID/PCI-DL_freq (XXX-1675)]
Network type=4G
PLMN=208-01
Band=3
Downlink EARFCN=1675
[+] New cell detected [CellID/PCI-DL_freq (XXXXX-3011)]
Network type=3G
PLMN=208-1
Band=8
Downlink UARFCN=3011
Uplink UARFCN=2786
=> Changing network type for 3G only
[+] New cell detected [CellID/PCI-DL_freq (XXXXX-2950)]
Network type=3G
PLMN=208-1
Band=8
Downlink UARFCN=2950
Uplink UARFCN=2725
```

Using Modmobmap with xgoldmon
——————————

With XGold modems, the use of xgoldmon will be required. But for now, only the fork for *Modmobmap* works to retrieve exact information of cells via the DIAG interface, and could be downloaded at: https://github.com/FIUxIuS/xgoldmon

Then after compiling, the tool *xgoldmon* could be started using the *-m* parameter like this:

```
sudo ./xgoldmon -t s3 -m /dev/ttyACM1
```

This will create a FIFO file that will be requested by Modmobmap later:

```
$ ls
celllog.fifo Makefile screenshot-mtsms-while-in-a-call.png xgoldmon
```

Then we can start running *Modmobmap* as follows precising the AT serial interface (*/dev/ttyACM0*) and the fifo file created by *xgoldmon* (* Requesting a list of MCC/MNC. Please wait, it may take a while…

```
Found 4 operators in cache, you choose to reuse them.
Found 4 operator(s)
{‘20801’: ‘Orange’, ‘20810’: ‘F SFR’, ‘20815’: ‘Free’, ‘20820’: ‘F-Bouygues Telecom’}
[+] New cell detected [CellID/PCI-DL_freq (0x7XXXX-65535)]
Network type=3G
PLMN=208-1
Downlink UARFCN=65535
Uplink UARFCN=2850
```

```
[+] Unregistered from current PLMN
[+] New cell detected [CellID/PCI-DL_freq (0x7XXXX-3011)]
Network type=3G
PLMN=208-1
Downlink UARFCN=3011
Uplink UARFCN=2786
[…]
[+] Unregistered from current PLMN
=> Changing MCC/MNC for: 20810
[+] New cell detected [CellID/PCI-DL_freq (0x3XXXXX-3075)]
Network type=3G
PLMN=208-10
Downlink UARFCN=3075
Uplink UARFCN=2850
[…]
```
“`

Note that retrieving results from AT+COPS command could take a lot of time and sometime would need to restart the tool. If the tool is blocked on the operator retrieving step, please use cached or targeted operators features instead.

Saving results
— — — — —

The process could be stopped any time when killing the process with a keyboard interrupt signal. Then results will be automatically save in a JSON file as follows:

“`
```
[…]
^C[+] Cells save as cells_1528738901.json
```
“`
Modmobjam

A smart jamming proof of concept for mobile equipments that could be powered with Modmobmap

For more information, this little tool has been presented during SSTIC rump 2018:

english slides: https://www.synacktiv.com/ressources/sstic_rump_2018_modmobjam.pdf
french presentation: https://static.sstic.org/rumps2018/SSTIC_2018-06-14_P10_RUMPS_22.mp4

Warning

You should be warned that Jamming is illegal and you’re responsible for any damages when using it on your own.

Prerequisites

a radio devices that is enabled to transmit signal (HackRF, USRP, bladeRF, and so on.)
GNU Radio installed
Modmobmap to perform automatic smartjamming: https://github.com/Synacktiv/Modmobmap

Usage

Manual jamming

If you have a HackRF or any device compatible with osmocom drivers, you can directly run the code provided in GRC/jammer_gen.py as follows:

```
$ python GRC/jammer_gen.py
```

For those who want to use another device like USRP, edit the GNU Radio block schema GRC/jammer_gen.grc:

```
$ gnuradio-companion GRC/jammer_gen.grc
```

Then you can configure the central frequency with the WX GUI to target a frequency. But this tool has also a feature to do it automatically.

Automatic smartjamming

To automate jamming, you can first get a list of we the Modmobmap that saves a JSON file after monitoring surrounding cells in a precise location. This JSON file looks as follows:

```
$ cat cells_<generated timestamp>.json
{
   "****-***50": {
      "PCI": "****",
      "PLMN": "208-01",
      "TAC": "50****",
      "band": 3,
      "bandwidth": "20MHz",
      "eARFCN": 1850,
      "type": "4G"
   },
   "7-***": {
      "PLMN": "208-20",
      "arfcn": 1018,
      "cid": "***",
      "type": "2G"
   },
   "****:-****12": {
      "PLMN": "208-1",
      "RX": 10712,
      "TX": 9762,
      "band": 1,
      "type": "3G"
   },
   [...]
}
```

After generating this file containing cells to jam, you can launch the RPC client that communicate with GRC/jammer_gen.py as follows:

```
$ python smartjam_rpcclient.py -f cells_<generated timestamp>.json
```

Then leverage the gain for transmission and you should observe that a lot of noise is overflowing the targeted cells with gaussian noise.

Jamming session

Please note that the delay between each targeted cell can be set with a provided arguments '-d' (see arguments helper).

OpenAirInterface 4G/LTE with LimeSDR_Mini
OS ubuntu 17.04

```
# apt-file search gtp.ko | grep lowlatency
linux-image-4.13.0-36-lowlatency: /lib/modules/4.13.0-36-
lowlatency/kernel/drivers/net/gtp.ko
# apt install linux-image-4.13.0-36-lowlatency
```

SoapySDR

```
# git clone https://github.com/pothosware/SoapySDR
# apt-get install cmake g++ libpython-dev python-numpy swig
# mkdir build
# cd build
# cmake .. 3/20

# make -j4
# sudo make install
# sudo ldconfig #needed on debian systems
# SoapySDRUtil --info
```

LimeSuite

```
# sudo apt-get install git g++ cmake libsqlite3-dev libi2c-dev libusb-1.0-0-
dev
#install graphics dependencies
sudo apt-get install libwxgtk3.0-dev freeglut3-dev
# git clone https://github.com/myriadrf/LimeSuite
# cd LimeSuite
# cd build
# cmake ..
```

Download & Compile the eNB on 17.04

```
# git clone https://github.com/myriadrf/trx-lms7002m

# wget http://open-cells.com/d5138782a8739209ec5760865b1e53b0/opencells-mods-
20170710.tgz

# tar xf opencells-mods-20170710.tgz
# git clone https://gitlab.eurecom.fr/oai/openairinterface5g.git
# cd openairinterface5g
# git checkout 08b8b3142df16831396a5283a015564ff56bf91c
# git apply ../opencells-mods/eNB.patch
# source oaienv
# ./cmake_targets/build_oai -I
# ./cmake_targets/build_oai -c -w LMSSDR --eNB -x
```

Download and patch EPC

```
# cd
# git clone https://gitlab.eurecom.fr/oai/openair-cn.git
# cd openair-cn
# git checkout develop
# git apply opencells-mods/EPC.patch
# source oaienv; cd scripts
# ./build_hss -i
```

set your MySQL password and remember it!
Answer yes to install: freeDiameter 1.2.0
phpmyadminn choose apache, configure database for phpmyadmin with
dbconfig-common: yes, password: same as MySQL for simplicity

Install 3PP SW for mme and spgw
```
# ./build_mme -i
```
Do you want to install freeDiameter 1.2.0: no
Do you want to install asn1c rev 1516 patched? : no
Do you want to install libgtpnl ? : yes
wireshark permissions: as you prefer
```
# ./build_spgw -i
```

4/20

Do you want to install libgtpnl ? : no
Compile the EPC nodes
```
# cd ~/openair-cn; source oaienv; cd scripts
# ./build_hss
# ./build_mme
# ./build_spgw

# nano openairinterface5g/targets/PROJECTS/GENERIC-LTE-
EPC/CONF/enb.band7.tm1.25PRB.lmssdr.conf
```

Change tx_gain 127 and rx_gain 160 je careful i am not sure it is necessary and it is max values
Change to this
```
////////// MME parameters:
mme_ip_address = ( { ipv4 = “127.0.0.20”;
ipv6 = “192:168:30::17”;
active = “yes”;
preference = “ipv4”;
}
);
NETWORK_INTERFACES :
{
ENB_INTERFACE_NAME_FOR_S1_MME = “lo”;
ENB_IPV4_ADDRESS_FOR_S1_MME = “127.0.0.10/8”;
ENB_INTERFACE_NAME_FOR_S1U = “lo”;
ENB_IPV4_ADDRESS_FOR_S1U = “127.0.0.10/8”;
ENB_PORT_FOR_S1U = 2152; # Spec 2152
};
```
And this
In the eNB config file, you need also to set the MCC and MNC as per your SIM card:
```
tracking_area_code = “1”;
mobile_country_code = “208”;
mobile_network_code = “92”;
```
Install this configuration for EPC
For the EPC, we install in OAI default directory: /usr/local/etc/oai
```
# sudo mkdir -p /usr/local/etc/oai
# sudo cp -rp ~/opencells-mods/config_epc/* /usr/local/etc/oai
# cd ~/openair-cn; source oaienv; cd scripts
# ./check_hss_s6a_certificate /usr/local/etc/oai/freeDiameter hss.OpenAir5G.Alliance
# ./check_mme_s6a_certificate /usr/local/etc/oai/freeDiameter mme.OpenAir5G.Alliance
```
5/20

Only the SGi output to internet need to be configured.
```
# nano /usr/local/etc/oai/spgw.conf,
```
your should set the interface that is connected to Internet, and,
to tell to the PGW to implement NAPT for the UE traffic

```
PGW_INTERFACE_NAME_FOR_SGI = “wlp2s0”;
PGW_MASQUERADE_SGI = “yes”;
```
Change wlp2s0 with your interface
For the SIM card, you’ll have more to do:
SIM MCC/MNC should be duplicated in a couple of files
eNB: See above in eNB configuration chapter
MME file: /usr/local/etc/oai/mme.conf to update
```
GUMMEI_LIST = ( MCC=”208″ ; MNC=”92″; MME_GID=”4″ ; MME_CODE=”1″; } );
TAI_LIST = ({MCC=”208″ ; MNC=”92″; TAC = “1”; } );
```
A HSS database in text is in: opencells-mods/opencells_db.sql
for phpmyadmin
```
# sudo ln -s /usr/share/phpmyadmin /var/www/html
```
10 users is network 208/92 (a French test network) are also created
```
# nano /usr/local/etc/oai/hss.conf
```
HSS Configure the password for MySQL set password as the password you created during MySQL installation
uncomment #OPERATOR_key = “11111111111111111111111111111111”; # OP key matching your database
```
# cd ~/openair-cn/scripts
# ./hss_db_import 127.0.0.1 root linux oai_db ~/opencells-mods/opencells_db.sql
```
supposed your password is linux
program the Usim
```
apt-get install python-pyscard python-serial python-pip
pip install pytlv
```
to find ki matching you should k at startup of
```
# ./run_hss
```
to find opc
```
# ./auchss.py -o 11111111111111111111111111111111 -k
6874736969202073796D4B2079650A73
# sudo python pySim-prog.py –type=”sysmoUSIM-SJS1″ –mcc=208 –mnc=92 –
imsi=208920100001108 –opc=777f0406a78d9598b0330d63f4c52199 – 6/20

ki=6874736969202073796D4B2079650A73 –iccid=8988211000000227713 –pin-
adm=40303607 –acc=0200
```

Change pin-adm=40303607 –acc=0200 with your own values
Running !!
```
# cd ~/openair-cn/scripts
# sudo ./run_hss
# sudo ./run_mme
# sudo ./run_spgw
# cd ~/openairinterface5g

# sudo -E targets/bin/lte-softmodem.Rel14 -O targets/PROJECTS/GENERIC-LTE-
EPC/CONF/enb.band7.tm1.25PRB.lmssdr.conf –rf-config-file ~/trx-lms7002m/config-
limeSDR/LimeSDR_Mini_v06.ini -d
```

The last puzzle piece with the lousy bursty throughput and those log errors:
```
[PHY][W][eNB 0, CC 0] frame 733, subframe 4, UE 0: ULSCH consecutive error count
reached 20, triggering UL Failure
[MAC][I][UL_failure_indication] [eNB 0][UE 0/80e6] Frame 733 subframeP 4 Signaling UL
Failure for UE 0 on CC_id 0 (timer 0)
[PHY][E]ERROR: Format 1A: rb_alloc (1ff) > RIV_max (144)
```
was partly solved by using the lte-softmodem -d switch Enable soft scope and L1 and L2 stats (Xforms), since it was built with the -x –xforms option, and partly by randomly moving the phone around and noticing there was a sweet spot where Firefox would download and

install very fast.
Good luck