Kamal & Sayak

GSM Wireless Sniffer using Software Defined Radio

Nicolae Crisan
Department of Communications
Faculty of Electronics, Telecommunications and Information Technology, Technical University of Cluj-Napoca
Cluj-Napoca, Romania
Nicolae.Crisan@com.utcluj.ro

Maria Condrea
Department of Communications
Faculty of Electronics, Telecommunications and Information Technology, Technical University of Cluj-Napoca
Cluj-Napoca, Romania
-

Abstract—In a software defined radio architecture, many components and modules that have been typically implemented in hardware are allowed to be replaced with software components. This innovative concept has evolved toward visual programming in which case the signal processing is following a flow-graph to create very versatile and outstanding design. This implementation leads to an open collection of tools written in C and Python that are allowing the most complex signal processing design under GNU Radio umbrella. Starting with the SpectrumWave project at Massachusetts Institute of Technology (MIT), the GNU Radio, is now among the most useful tool, for wireless packet sniffing together with software defined radio (SDR). In mobile communications, the providers are still using a combination of the old 2G (GSM) technologies together with the new 3G (UMTS) and 4G (LTE & WiMAX). The wireless sniffing will be largely used especially on universities to analyze the protocols and to evaluate the overall performances of wireless networks. This paper puts together the software packages required to trigger the sniffing approach of the GSM-900, and to capture the BCCH & CCCH frames [1].

Keywords—Software Defined Radio; GNU-Radio, GSM, Mobile Communications

I. Introduction

The most advanced penetration testing platform, *Kali-Linux* could be a handful solution for any start in the matter. Under *Kali-Linux Rolling* one can find and ready to use, many tools like *Wireshark* for network sniffing [2]. The main advantage of the *Kali-Linux* distribution against the *Ubuntu* or *Debian* is, in fact, related to the special packages installed that are useful in software penetration and testing. Nevertheless, there is no distribution ready for GSM sniffing so; there are many to be done before starting the capturing. The most important step is to identify a low-cost SDR that is suited for the sniffing approach. One of the low-cost SDR available on the market is the RTL-SDR, initially designed for the DVB-T (Digital Video Terrestrial Broadcasting) and also for DAB (Digital Audio Broadcasting) and FM reception [3]. There are many versions of the RTL-SDR of the kind, but, only the one that is working with the R820T or with the E4000 tuner is compatible with the *GNU Radio Osmo* libraries and drivers. The RT820T tuner operates between 24 and 1766 MHz. E4000 tuner it rises the higher frequency at 2200 MHz but is not as sensitive as R820T. One of the most attractive that fits better with the purpose of the paper is the one with the black dongle MCX connector (see figure 1) that allows the connection of an external antenna tuned for GSM-900 frequency band. The MCX connector has less insertion loss at 900 MHz and experiences a better adaptation to the external 50 Ohms antenna. This adaptation must be taken especially in microwaves, and the dongles with the 75 Ohms TV connectors must be avoided. This connector has low quality and is mismatching the surge impedance of the antenna cable (50 Ohms).

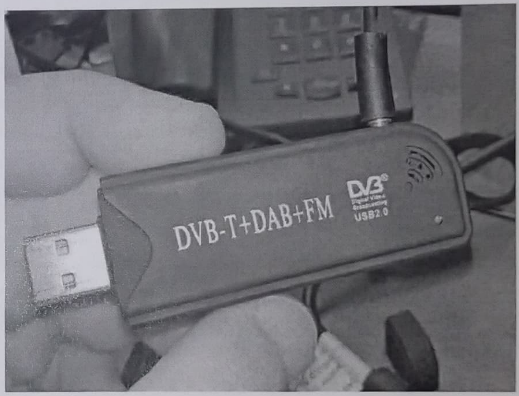


Figure 1: RTL-SDR with the R820T tuner, compatible with GNU-Radio

GNU Radio comes with Osmo libraries and is working with the DVB dongle via USB I/O port. The SDR is receiving the analog RF signal using an antenna tuned at the GSM frequency band. At its input is operating the R820T tuner that amplifies the RF signal and down converts the frequency from 900 MHz to an intermediate frequency. An analog to digital converter is built in, inside the RTL2832U chip together with the demodulator and the decoder. Nevertheless, the demodulating and the decoding are canceled by the Osmo driver and the samples are redirected to the I/O port. The maximum sampling rate is 3.2 Msps, so, according to the Nyquist criterion, the maximum convertible signal's frequency is 1.6 MHz. Any signal that has a narrower frequency bandwidth than 1.6 MHz can be processed online or offline, in GNU Radio. The E-GSM 900 downlink (baseband to phone) band in Romania is planned between 925 – 960 MHz with 125 active channels indexed from 0 to 124. The span between two consecutive subcarriers is 200 KHz. The ideal sampling rate for the GSM signal conversion is about 1.6 Msps at the resolution of the converter (which is 8 bits per sample).

II. INSTALLING THE SOFTWARE IN KALI-LINUX ROLLING EDITION

The last edition of the Kali-Linux is at this moment [2] the Rolling and is the one the experiments are prepared for in this paper. The preparations demand the installation of the "libosmocore" (some libraries and drivers used by the RTL-SDR and the GNU-Radio), gnuradio-companion (GNU-Radio packets). The most important thing is to update first all Linux packets with the following commands [3]:

```
> apt updates
> apt upgrade –y
> apt-get install Kali-Linux-all
> apt-get install flashplugin-nonfree
> update-flashplugin-nonfree –install
```

All these commands are written in at the terminal requiring root privileges so the sudo commands must be placed in the front of each line if one is not logged in as root.

The step two is to install using the packages downloader all applications that are related to the names "Osmo," "GSM" and "SDR" from the menu - *Aplications/Usual Application/ System/ Package Downloader*.

The step three is to install all dependencies as follows:

```
> apt-get install hackrf libhackrf-dev libhackrf0
> apt-get –y install git-core autoconf automake libtool g++ python-dev swig libpcap0.8-dev
> apt-get install gnuradio gnuradio-dev gr-osmosdr
> apt-get install git cmake libboost-all-dev libcppunit-dev swig doxygen liblog4cpp5-dev python-scipy
```

The step four is to install and compile the "libosmocore" library that makes the connection between the acquired samples and the gnu-radio, possible.

```
> apt-get install build-essential libtool shtool autoconf automake git-core pkg-config make gcc
> apt-get install libpcsclite-dev
> git clone https://github.com/ptrkrysik/gr-gsm.git
> cd gr-gsm
> git clone git://git.osmocom.org/libosmocore.git
> cd libosmocore
> autoconf –i
> ./configure
> make
> make install
> ldconfig –i
```

In the gr-gsm directory one has to compile the gr-gsm packages in a new subdirectory called "build" with the following commands in the terminal [4]:

```
> mkdir build
> cd build
> cmake ..
> make install
> ldconfig
```

In the step five we have to edit the file "config.conf" using the root privileges and put in the next lines:

```
[grc]
Local-blocks_path=/usr/local/share/gnuradio/grc/blocks
```

The last but not least thing is to install the software that calibrates the RTL-SDR dongle for GSM packets capturing:

```
> git clone https://github.com/steve-m/kalibrate-rtl
> cd kalibrate-rtl
> ./bootstrap
> ./configure
> make
> make install
> ldconfig
```

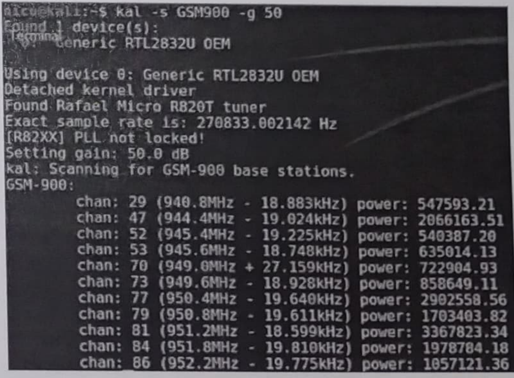
Now the system is ready for GSM sniffing, and all we have to check before is whether the *WireSharck* is installed. If not, then we can installed it using the Package *Downloader*.

III. GSM WIRESHARK CAPTURE OVER THE SDR AND AIRPROBE

The *Airprobe* is an application developed in a project known as the *GSM_Sniffing* project under linux. It uses already installed repository from the previous section to receive and decode the signal. In this case, the *Airprobe* is capable of decoding only the downstream signal from base station to phone capturing and decode the management channels. In order to see the available cells in the current location one have to run the *kal* commands [3] with the parameters:

```
> kal –s GSM900 –g 40.
```

The letter *g* stands for the RF gain in dB. The maximum available RF gain for the RTL-SDR is 40 dB. The program will detect and show a list of the available channels with the necessary frequency shift for the local oscillator (see figure 2.a).



```
nicu@kali:~$ kal -s GSM900 -g 50
Found 1 device(s):
  0:  Generic RTL2832U OEM

Using device 0: Generic RTL2832U OEM
Detached kernel driver
Found Rafael Micro R820T tuner
Exact sample rate is: 270833.002142 Hz
[R82XX] PLL not locked!
Setting gain: 50.0 dB
kal: Scanning for GSM-900 base stations.
GSM-900:
        chan: 29 (940.8MHz - 18.883kHz) power: 547593.21
        chan: 47 (944.4MHz - 19.024kHz) power: 2066163.51
        chan: 52 (945.4MHz - 19.225kHz) power: 540387.20
        chan: 53 (945.6MHz - 18.748kHz) power: 635014.13
        chan: 70 (949.0MHz + 27.159kHz) power: 722904.93
        chan: 73 (949.6MHz - 18.928kHz) power: 858649.11
        chan: 77 (950.4MHz - 19.640kHz) power: 2902558.56
        chan: 79 (950.8MHz - 19.611kHz) power: 1703403.82
        chan: 81 (951.2MHz - 18.599kHz) power: 3367823.34
        chan: 84 (951.8MHz - 19.810kHz) power: 1978784.18
        chan: 86 (952.2MHz - 19.775kHz) power: 1057121.36
```

Figure 2. The list of the domestic GSM900 channels at Str. Observatorului nr. 1 Cluj-Napoca Romania

The frequency shift for each channel will be useful later. Now we can use the installed program *gqrx* and start it from the terminal with any parameters. We can adjust the RF and IF gain to avoid distortions and to have a good signal to noise ratio of about 20 dB at least (see figure 3). The next step is to start the *WireShark* from the terminal and to select the *loopback* and to filter after the *gsmtap* protocol. The WireShark will start capturing the network packets, but no GSM packets will be available at start . The *Airprobe* will simulate the *GSM* packets over the *loopback* to make these Wireshark packets to show themselves.

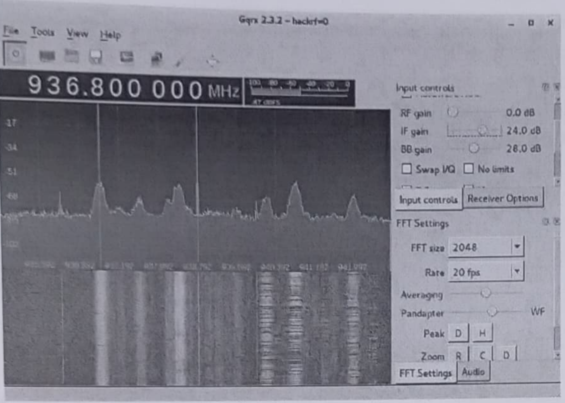

```
936.800 000 MHz
RF gain   0.0 dB
IF gain   24.0 dB
BB gain   26.0 dB
FFT size  2048
Rate      20 fps
```

Figure 3. The graphical interface of the *gqrx* program and the spectrum of the GSM signal

> *gnu radio-companion airprobe_rtlsdr.py –s 1e6 –f 9522000000*

This command from the terminal is opening the gnu-radio interface and start the python source that will capture the GSM packets at 952.2 MHz (channel 86 taken from figure 2). The frequency shift for the local oscillator is extracted from the FCCH as a mobile phone normally does [5]. The command also starts the GSM live monitor written in Python by Piotr Krysik. The program uses the shift frequency to lock the local oscillator frequency over the one of the BTS. Then the packets start to flow down continuously in the Wireshark screen interface (figure 4).

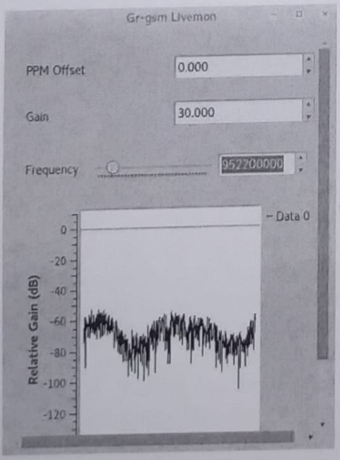

```
PPM Offset  0.000
Gain        30.000
Frequency   952200000
```

Figure 4. Gr-gsm Livemon's program interface written by Piotr Krysik in python

One can see in figure 3.b that the temporary mobile identifier number that locates the subscriber in the cell is visible. The input power signal of -33 dBm at the BTS is reported too. The phone uses this information to adjust its output power [5].

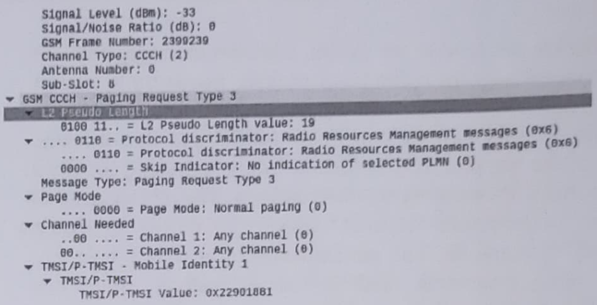
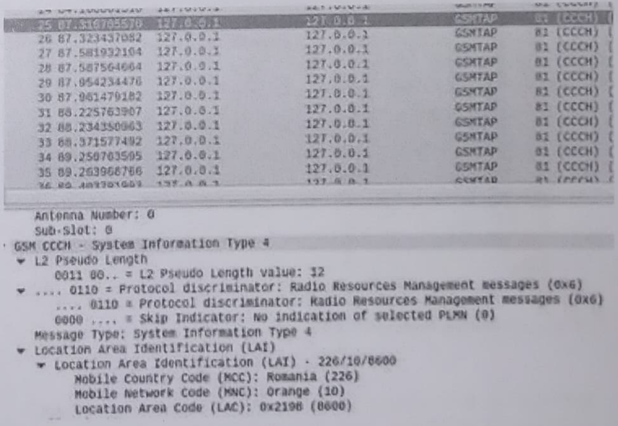

```
        Signal Level (dBm): -33
        Signal/Noise Ratio (dB): 0
        GSM Frame Number: 2399239
        Channel Type: CCCH (2)
        Antenna Number: 0
        Sub-Slot: 0
  ▾ GSM CCCH - Paging Request Type 3
    ▾ L2 Pseudo Length
          0100 11.. = L2 Pseudo Length value: 19
    ▾ .... 0110 = Protocol discriminator: Radio Resources Management messages (0x6)
          .... 0110 = Protocol discriminator: Radio Resources Management messages (0x6)
          0000 .... = Skip Indicator: No indication of selected PLMN (0)
        Message Type: Paging Request Type 3
    ▾ Page Mode
          .... 0000 = Page Mode: Normal paging (0)
    ▾ Channel Needed
          ..00 .... = Channel 1: Any channel (0)
          00.. .... = Channel 2: Any channel (0)
    ▾ TMSI/P-TMSI - Mobile Identity 1
      ▾ TMSI/P-TMSI
            TMSI/P-TMSI Value: 0x22901881
```

Figure 5. Screenshot with the CCCH – System Information with signal level, GSM frame, GSM sub-slot number and mobile identity TMSI (Temporary Mobile Subscriber Identity)

One can easily see now in figure 5 that the weakness of the CCCH channels are the redundancy of the information [6]. In the first line the sentence "Signal Level (dBm): " remains unchanged from one packet to the other. This let open the enter door for the attacker. He will use the encrypted stream of the known sentence to feed in a xor gate. The second input of the gate is fed with the known sequence. The temporary key Ki will be available at the output of the xor gate. Once the temporary key has been identified the attacker will initiate the decoding process of the current packet for wich the temporary key is still unchanged.


```
25 87.[illegible]   127.0.0.1   127.0.0.1   GSMTAP   81 (CCCH)
26 87.323437082   127.0.0.1   127.0.0.1   GSMTAP   81 (CCCH)
27 87.581932104   127.0.0.1   127.0.0.1   GSMTAP   81 (CCCH)
28 87.587564664   127.0.0.1   127.0.0.1   GSMTAP   81 (CCCH)
29 87.054234476   127.0.0.1   127.0.0.1   GSMTAP   81 (CCCH)
30 87.961479182   127.0.0.1   127.0.0.1   GSMTAP   81 (CCCH)
31 88.225763907   127.0.0.1   127.0.0.1   GSMTAP   81 (CCCH)
32 88.234350063   127.0.0.1   127.0.0.1   GSMTAP   81 (CCCH)
33 88.371577492   127.0.0.1   127.0.0.1   GSMTAP   81 (CCCH)
34 89.250703595   127.0.0.1   127.0.0.1   GSMTAP   81 (CCCH)
35 89.263968766   127.0.0.1   127.0.0.1   GSMTAP   81 (CCCH)

      Antenna Number: 0
      Sub-Slot: 0
  GSM CCCH - System Information Type 4
  ▾ L2 Pseudo Length
        0011 00.. = L2 Pseudo Length value: 12
  ▾ .... 0110 = Protocol discriminator: Radio Resources Management messages (0x6)
        .... 0110 = Protocol discriminator: Radio Resources Management messages (0x6)
        0000 .... = Skip Indicator: No indication of selected PLMN (0)
      Message Type: System Information Type 4
  ▾ Location Area Identification (LAI)
    ▾ Location Area Identification (LAI) - 226/10/8600
          Mobile Country Code (MCC): Romania (226)
          Mobile Network Code (MNC): Orange (10)
          Location Area Code (LAC): 0x2198 (8600)
```

Figure 5. Screenshot with the Wireshark captured frames

In figure 5 is a screenshot with the Wireshark capture frames. Many frames of all kind can be captured for later processing. In this case, the brute force technique also can be used to find the keys for the packets that are not allowing the redundancy. These are uncorrelated packets that are carrying data or voice messages. Nevertheless, these packets are safe taking into the consideration that some single packets decrypted are useless without the others. For SMS messages that are traveling along SDCCH channels, the redundancy is avoided so the attacker can't exploit this door [7]. Although, an attacker can send to the victim, a known message, with the text that replaces the redundancy packets, necessary to exploit the weakness. In the very proximity with his victim, the attacker can capture the encrypted packets and find the key. One of the big problems is that the banks are using the SMS mechanism to authenticate their clients. The mechanism of client identification via GSM phones by SMSs is clearly classified as unsafe. The mechanism of access as a hole that are using passwords and a secure internet connection compensate the weakness and narrow the door for an attacker.

For the GSM providers will be hard to counteract this weakness because of the standard constraints and the number of the phone that are using the GSM standard. One of the most available solution to counteract this kind of weaknesses in the future is to allow the hardware reconfiguration through software. This approach probably will close the door, but it surely opens the window.

IV. RESULTS AND CONCLUSIONS

The paper presents a set of tools and devices that are allowing GSM packets' capturing and decoding. This approach could be valuable for understanding and analyzing the GSM standard regarding the protocol and the link security. The approach in the paper works only with the BCCH&CCCH decoding. The method exploits the GSM weakness taking in the redundancy of the BCCH&CCCH packets that undergo the packets' ciphering. As long as the redundancy generates known patterns, the repetitive chunks can be exploited together with the chipper data. After the "xor" operation between the two, the output can lead to the Ki-sequence used for encrypting the entire packet. This method couldn't be as efficient for example with the SDCCH in one attempts of SMS data extraction. But even in this case, an SMS message sent to the victim can offer the redundancy needed for the Ki extraction [7]. The paper follows the procedure described by Piotr Krysik and his implementation (Gr-gsm Livemon) in Python [4].

REFERENCES

[1] S. Zahan, Telefonia Digitala în Rețele de Telecomunicații, Cluj Napoca, Ed. Albastră, 2005

[2] Kalli Linux Documentation, www.kali.org, unpublished.

[3] RTL_SDR, Airprobe the usege documentations with wireshark, unpublished, http://www.rtl-sdr.com/rtl-sdr-tutorial-analyzing-gsm-with-airprobe-and-wireshark/

[4] Documentation about the use of the gr-gsm for GSM traffic sniffing, unpublished,https://www.ckn.io/blog/2015/11/29/gsm-sniffing-sms-traffic/

[5] Sauter, M., From GSM to LTE: An Introduction to Mobile Networks and Mobile Broadband , Wiley, first edition (February 7, 2011)

[6] S. Islam, I. UI Haq, A. Saeed, Secure end-toend SMS communication over GSM networks, Applied Sciences and Technology (IBCAST), 13-17 Jan. 2015.

[7] K. Nohl, S. Munaut, "Wideband GSM sniffing," 27 Chaos Communications Congress, 2010, http://tinyurl.com/33ucl2g.