Back

Matt Lewis Hardware & Embedded Systems Tutorial/Study Guide

May 19, 2016 17 mins read

GSM/GPRS Traffic Interception for Penetration Testing Engagements

technologies and devices. It's important to cover all aspects of connectivity of a device being tested which is why we have built a GSM/GPRS interception capability. There are a number of different devices and systems that make use of GSM/GPRS, non-exhaustively we commonly see:

- Mobile Applications (and perform mobile malware analysis)
- Smart Meters
- IoT devices
- Automotive Telematics Units (TCUs)
- Point-of-Sale (PoS) and PIN Entry Devices (PEDs)
- Alarm Systems
- Gambling Machines
- Mobile HotSpots

deployed, however, there are a few security design flaws within GSM such as:

- Encryption is not compulsory – Most telecoms operators use encryption in their networks, however, not every device will alert users when no encryption is used
- Lack of Mutual Authentication – The network authenticates the Mobile Station (MS) and not vice versa
- Cell Reselection mechanism - It's possible to force the MS to switch to a rogue Base Transceiver Station (BTS)

While next generation GSM networks are available (3G/UMTS and 4G/LTE) it appears that most GSM modules used in consumer electronics are backward compatible, meaning that when there is no 3G/4G network coverage or someone is jamming the 3G/4G frequencies (an illegal activity), devices will fall back and start to operate on the 2G network. According to rumours in the telecoms industry we have reached 3G sunset and in a number of markets UMTS 3G will be discontinued while 2G will continue to stay, allowing 2G/4G

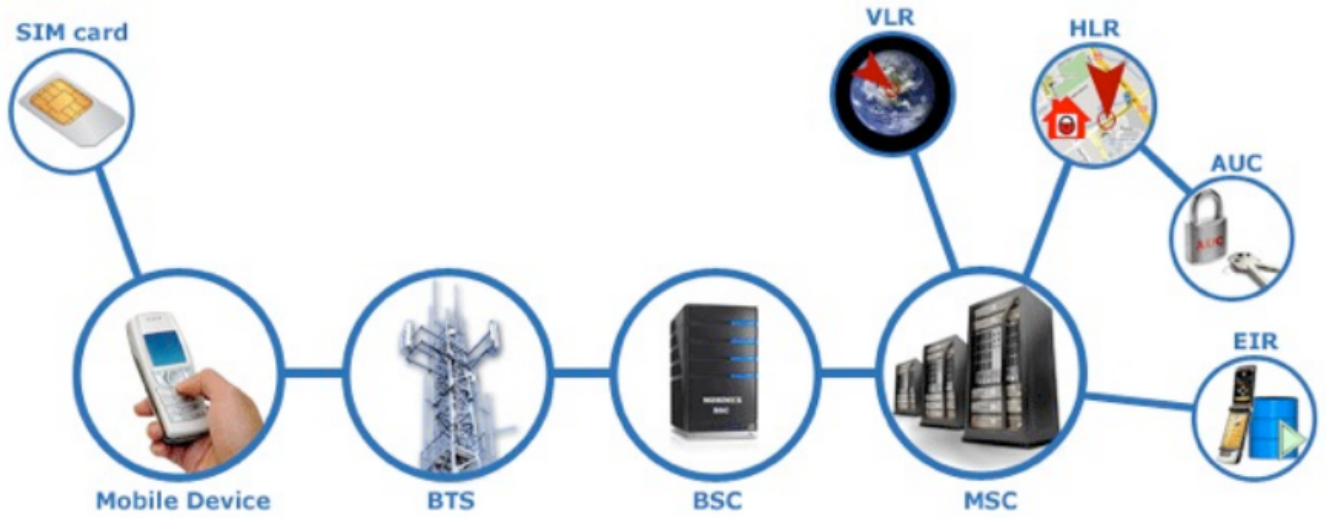
Running radio equipment and working with a GSM base station requires transmitting in the officially-regulated radio spectrum. Therefore, to legally run a base station an official allowance for a particular frequency is necessary. Here in the UK, OFCOM issues these licences therefore you should always contact OFCOM when you want to operate BTS. In our lab we use a dedicated professional RF-shielded cage so that we do not interfere with commercial operators.

How do we do this?

Currently there are at least a few Open Source projects that we could use to set up our own small-scale GSM network in a lab environment. To name only a few:

software running under a Debian 8 Linux distribution and with Ettus Research USRP B200 software defined radio. It seems like this setup is most cost effective and could be used with generic radio hardware and minimal additional components are easy to deploy. In addition, OpenBTS is in itself quite a mature project with good support. OpenBTS projects currently support second generation (2G) and third generation of the GSM protocol (3G/UMTS) at early alpha stage.

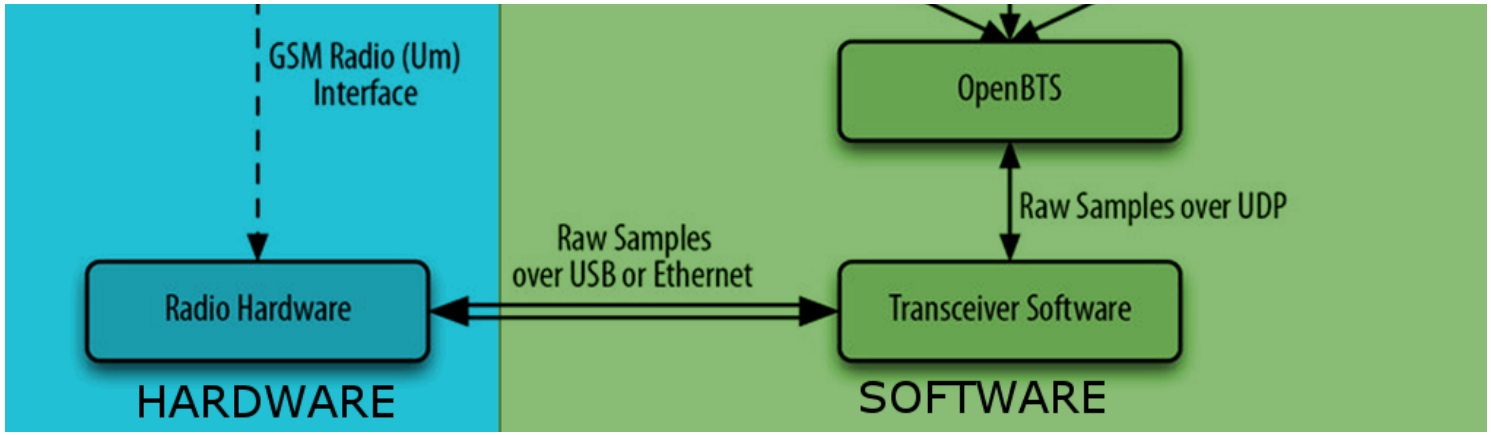
The basic GSM architecture is shown below:



its coverage range, registering the mobile station in its coverage and as soon as the mobile station invokes service a free channel is assigned to it. The MS (Mobile Station) sends its voice or digital signal to the BTS which sends it to the BSC (Base Station Controller) at which point the BSC sends it to the MSC (Mobile Switching Centre). The MSC connects to the other side Mobile Station/PSTN phone or connects to the SMSC (Short Message Service Centre) if the service is for SMS or SGSN (Serving GPRS Support Node) for Internet service. Thus BTS is the first contact for connection or release of a mobile service.

OpenBTS

OpenBTS is a software-based GSM access point, allowing standard GSM-compatible mobile phones to be used as SIP (Session Initiation Protocol) endpoints in VoIP networks. OpenBTS replaces the conventional GSM operator core network infrastructure from layer 3 upwards. Instead of relying on external base station controllers for radio resource management, OpenBTS units perform this function internally. Instead of forwarding call traffic through an operator's MSC, OpenBTS delivers calls via SIP to a VoIP soft switch or PBX.

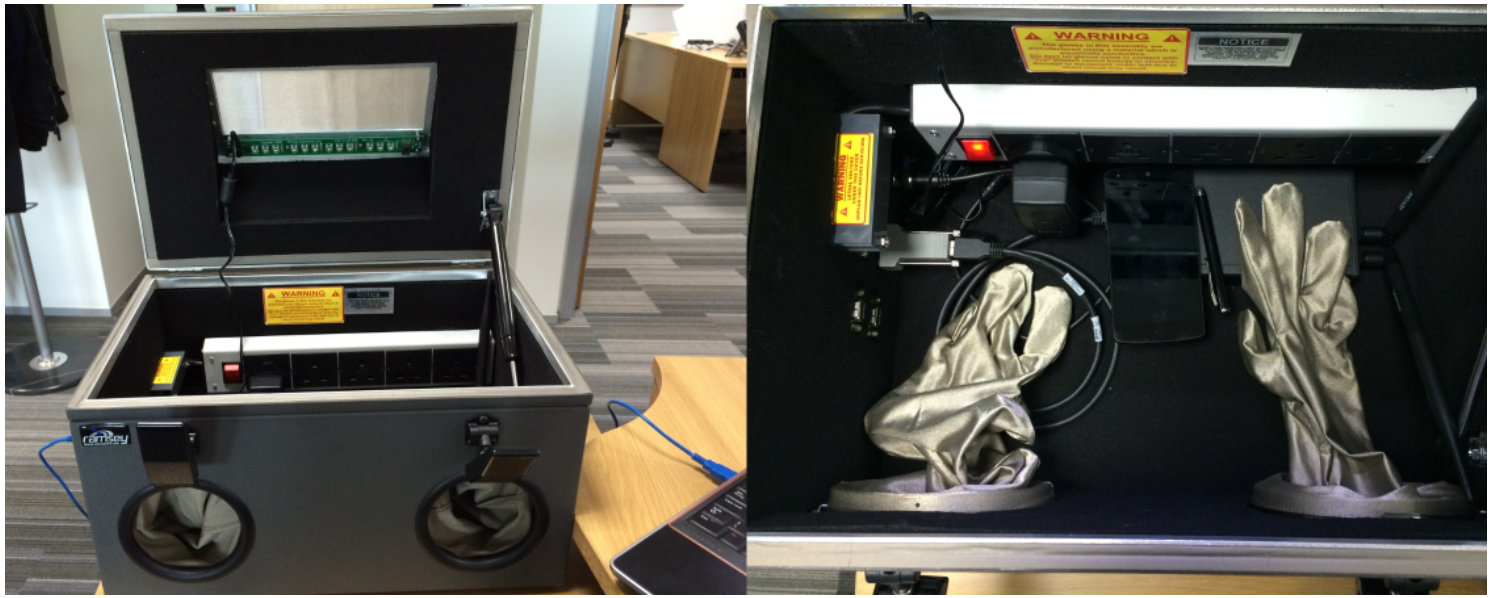


The main advantages of OpenBTS are:

- All handsets and modems connected to OpenBTS appear as a SIP device without the need for any special software on the device
- Hardware can be reduced to a single host computer running Software Defined Radio (SDR)
- All of the software runs on Linux and connects with commonly-used IP protocols

Our Testbed

frequencies within the EU. As GSM network heavily relies on frequency accuracy we have to provide a high-accuracy reference signal by installing a GPSDO-TCXO-MODULE onto the B200 board and two VERT900 antennas. More information about clock-related issues are described here [http://openbts.org/w/index.php/Clocks]. All of the software was able to run on a modern laptop with an i7 processor, SSD drive, 8 GB of RAM and a USB 3.0 controller. The laptop had a Debian 8 Linux distribution installed.

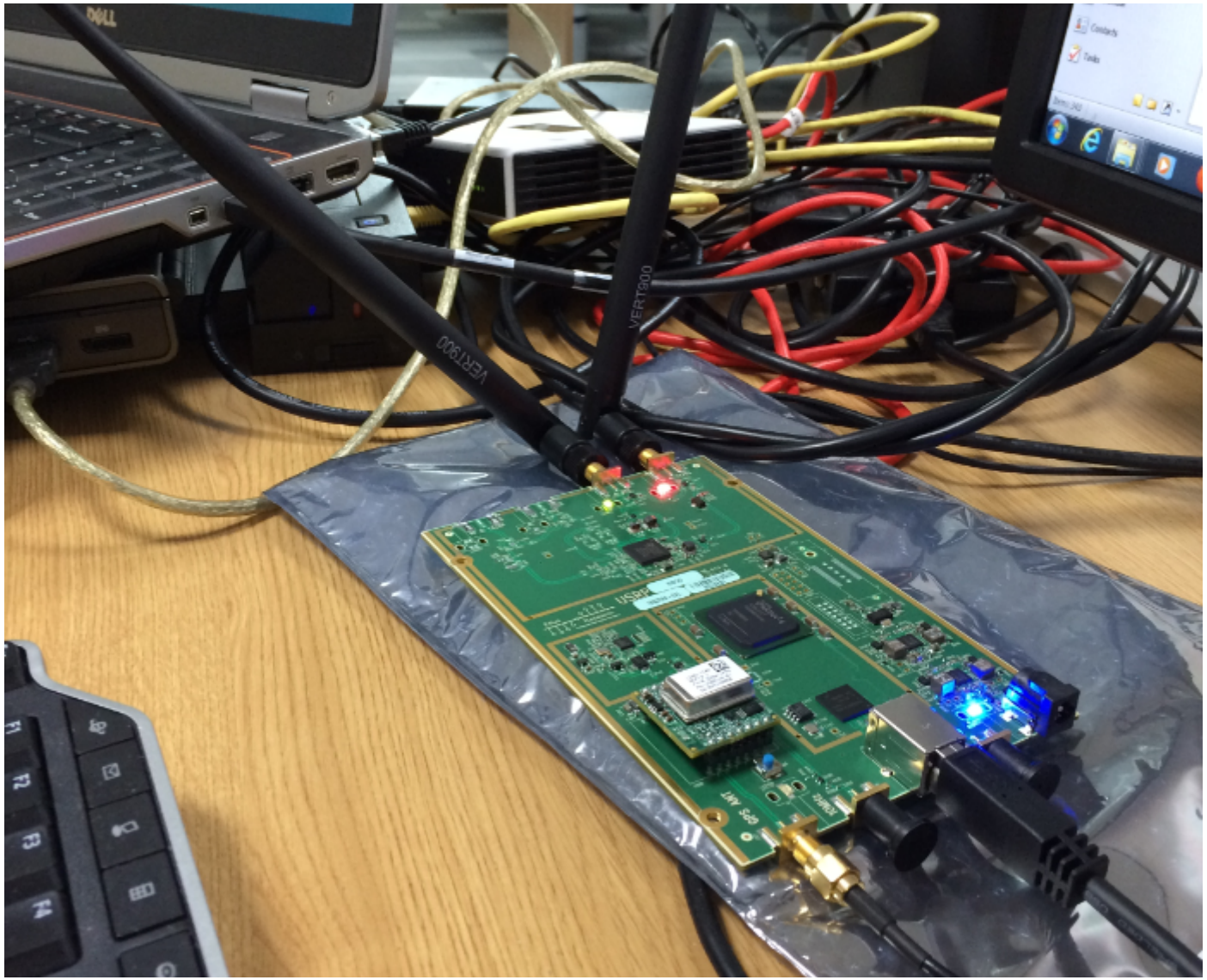


installed before we build OpenBTS from sources. The entire process is widely documented on the OpenBTS project website http://openbts.org/w/index.php/BuildInstallRun

After installation we have a few modules/services which need to be running to operate OpenBTS

- Sipauthserve – maintains subscriber registry database and SIP authorisation server for registration traffic
- Smqueue – Is the store-and-forward message service used to support SMS messaging
- Asterisk – Is a standard VoIP PBX service which is used to route calls within the system and externally
- Openbts – Is the core service comprising the GSM network stack

Hardware


DELL
VERT900
VERT900

by a small external power supply. The SDR hardware implements a completely generic radio that can send and receive raw waveforms in a defined frequency range (i.e., 50 MHz to 6 GHz) with a host application. That host application could be an FM radio implementation that receives the raw waveforms, demodulates the signal, and plays back the audio. This means that radio hardware is no longer designed around a specific application; it is completely up to the host to define the implementation, allowing anyone to create radio applications purely in software.

Basic Operation

If you followed the OpenBTS documentation and installed all dependencies you can start the openbts service by running the command:

You will then be able to use the OpenBTS Command Line Interface (CLI) to interact with your openbts installation. You can find the CLI utility in the /OpenBTS path:

```
root@GSMTB:/home/gsm# cd /OpenBTS/
root@GSMTB:/OpenBTS# ./OpenBTSCLI
OpenBTS Command Line Interface (CLI) utility
Copyright 2012, 2013, 2014 Range Networks, Inc.
Licensed under GPLv2.
Includes libreadline, GPLv2.
Connecting to 127.0.0.1:49300...
Remote Interface Ready.
Type:
 "help" to see commands,
 "version" for version information,
 "notices" for licensing information,
```

from 45 up to 70. The higher the value, the lower the TX power. Basically you shouldn't use too much power in an enclosed environment because this could result in signal saturation. A safe value is 45, however, sometimes 50 or even 55 can be set. When you finish an assessment it is recommended that the power is set to 70 using the OpenBTSCLI to make sure that the next time the equipment is used it does not inadvertently interfere with commercial mobile operators in the event that the user forgets to seal the enclosure.

```
OpenBTS> power
current downlink power -45 dB wrt full scale
OpenBTS>
```

Setting up CellID parameters

First you want to check some configuration options such as your GSM BTS ID. This can be done by typing the cellid command:

What you see here is the MCC and MNC.

MCC stands for Mobile Country Code. It must be exactly three digits as defined in ITU-T E.212. The default value for test networks is 001.

MNC stands for Mobile Network Code. It could be two or three digits. These numbers are assigned by your national regulator. The default value for the test networks is 01.

Basically, our OpenBTS installation is configured to accept all subscribers - no matter what card IMSI number their SIM card has. It works almost like roaming in a mobile network. You can use your phone to manually search for a network operator, and when you see it on the list you can elect to switch. It's worth mentioning that most cell phones have data transmission in roaming disabled, so if you are testing and you have "roamed" to the OpenBTS network then this could prevent the mobile device from sending any traffic. To bypass this you can enable data in roaming on the phone, or if this is not possible you can

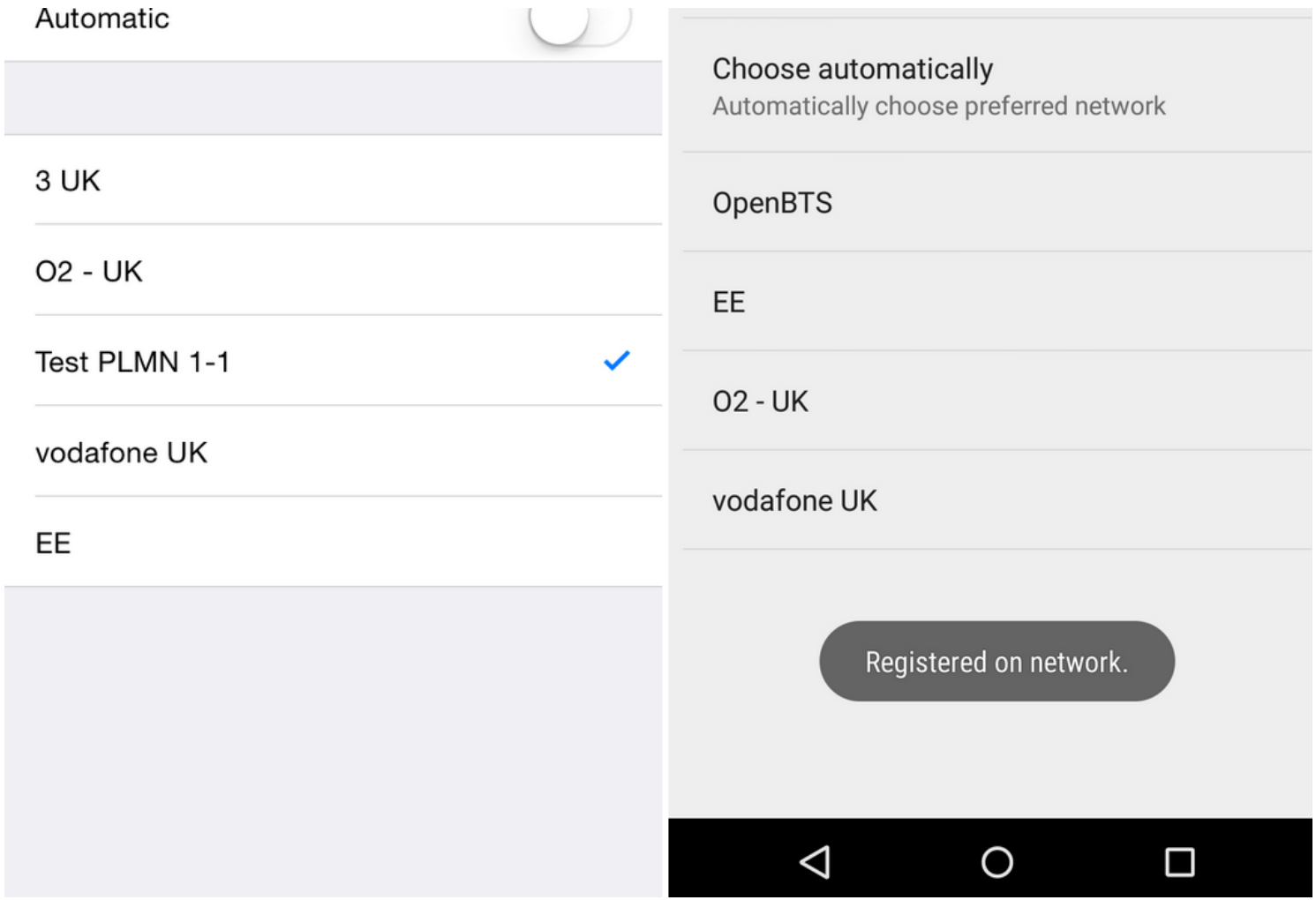
```
OpenBTS> config GSM.Identity.MCC 234
GSM.Identity.MCC is already set to "234", nothing changed

OpenBTS> config GSM.Identity.MNC 20
GSM.Identity.MNC is already set to "20", nothing changed

OpenBTS> cellid
MCC=234 MNC=20 LAC=1010 CI=10

OpenBTS>
```

CAUTION: In the example above we changed cellid to broadcast as the Three network (Hutchison 3G UK Ltd). You should not do this with an open enclosure..

by the phone to the network. To prevent eavesdroppers identifying and tracking the subscriber on the radio interface, the IMSI is sent as rarely as possible and a randomly generated TMSI (Temporary Mobile Subscriber Identity) is sent instead.

Connecting your target

When you run a manual network operator search on your mobile device you should find your test network on the list:


Automatic
3 UK
O2 - UK
Test PLMN 1-1
vodafone UK
EE
Choose automatically
Automatically choose preferred network
OpenBTS
EE
O2 - UK
vodafone UK
Registered on network.

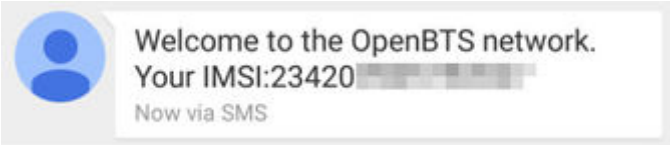
```
IMSI           TMSI IMEI           AUTH CREATED ACCESSED TMSI_ASSIGNED
234200900527575 -    358246838000527 2    264s    264s     0
204043521650775 -    358921020018920 2    11m     11m      0
234261027406775 -    358921020018920 2    7h      6h       0

OpenBTS>
```

As you can see in the above example you will also find the device identification number (IMEI) which could be helpful to identify our target device (Keep in mind that IMEI identifies the device while IMSI identifies the SIM card). Because GSM network authentication heavily relies on the Ki encryption key which is stored securely on the SIM card, OpenBTS supports three types of authentication:

1. AUTH type 2 - unauthenticated. It's connected to your network but it doesn't exists in your HLR

over Air Interface.

When you connect to the OpenBTS network you will receive an SMS message like below:



Managing subscribers

At this point, you are connected to the OpenBTS GSM network but even if you are able to make a test phone call you are not configured as a subscriber of this network (to cut a long story short - you don't have a number assigned to your phone in that network). Your phone - and to be more precise your SIM card - is identified and authenticated using the IMSI number.

It's easy to find out if you have any numbers assigned to your target SIM card or not. Simply connect to the OpenBTS network and type tmsis in OpenBTSCLI. You will see the AUTH

than sufficient to carry on with the test. Unless you are testing/or trying to break GSM air interface encryption you don't need to use full authentication (AUTH 0).

```
OpenBTS> tmsis
IMSI            TMSI IMEI            AUTH CREATED ACCESSED TMSI_ASSIGNED
234335501150083 -    352048064107610 1    285m    81m      0
234207505270905 -    358240052768380 2    268m    268m     0
```

To confirm and to see what number is assigned to your IMSI you could list all subscribers using the following command:

```
root@GSMTB:/OpenBTS# cd /opt/dev/NodeManager/
root@GSMTB:/opt/dev/NodeManager# ./nmcli.py sipauthserve subscribers read
raw request: {"command":"subscribers","action":"read","key":"","value":""}
raw response: {
```

```
            "name" : "Orange"
        },
        {
            "imsi" : "IMSI001010000000000",
            "msisdn" : "1000",
            "name" : "MagicSIM"
        },
        {
            "imsi" : "IMSI234335501150083",
            "msisdn" : "1030",
            "name" : "iPhone5s"
        }
    ]
}
```


As we see, we already have a number assigned to the authenticated (AUTH 1) IMSI number 234335501150083 and we cannot find any entry for the IMSI 234207505270905 which

Subscriber Directory Number".

As we see we have the number 1030 assigned to IMSI 234335501150083, which in this case is a SIM card in an iPhone. It's a good idea to authenticate and to assign MSISDN numbers to your target device SIM card. You will be able to interact with this device from OpenBTSCLI and from other devices connected to the network.

There are two ways to add a subscriber using nmcli.py. To add your target device as a subscriber and assign MSISDN number you need to run the following command:

The first creates a subscriber that will use cached authentication:

```
./nmcli.py sipauthserve subscribers create name imsi msisdn
```

The second creates a subscriber that will use full authentication:

234207505270905. Take note that you need to add the IMSI prefix to the IMSI number in this command.

```
root@GSMTB:/OpenBTS# cd /opt/dev/NodeManager/
root@GSMTB:/opt/dev/NodeManager# ./nmcli.py sipauthserve subscribers create
"Nexus5" IMSI234207505270905 1234
raw request: {"command":"subscribers","action":"create","fields":
{"name":"Nexus5","imsi":"IMSI234207505270905","msisdn":"1234","ki":""}}
raw response: {
        "code" : 200,
        "data" : "both ok"
}
root@GSMTB:/opt/dev/NodeManager#
```

We can reregister our target device in the network and use the tmsis command:

```
OpenBTS>
```

Testing Voice Calls

To perform further tests you can test voice calls by calling these numbers:

- 2600 - Echo service
- 2602 - Noise

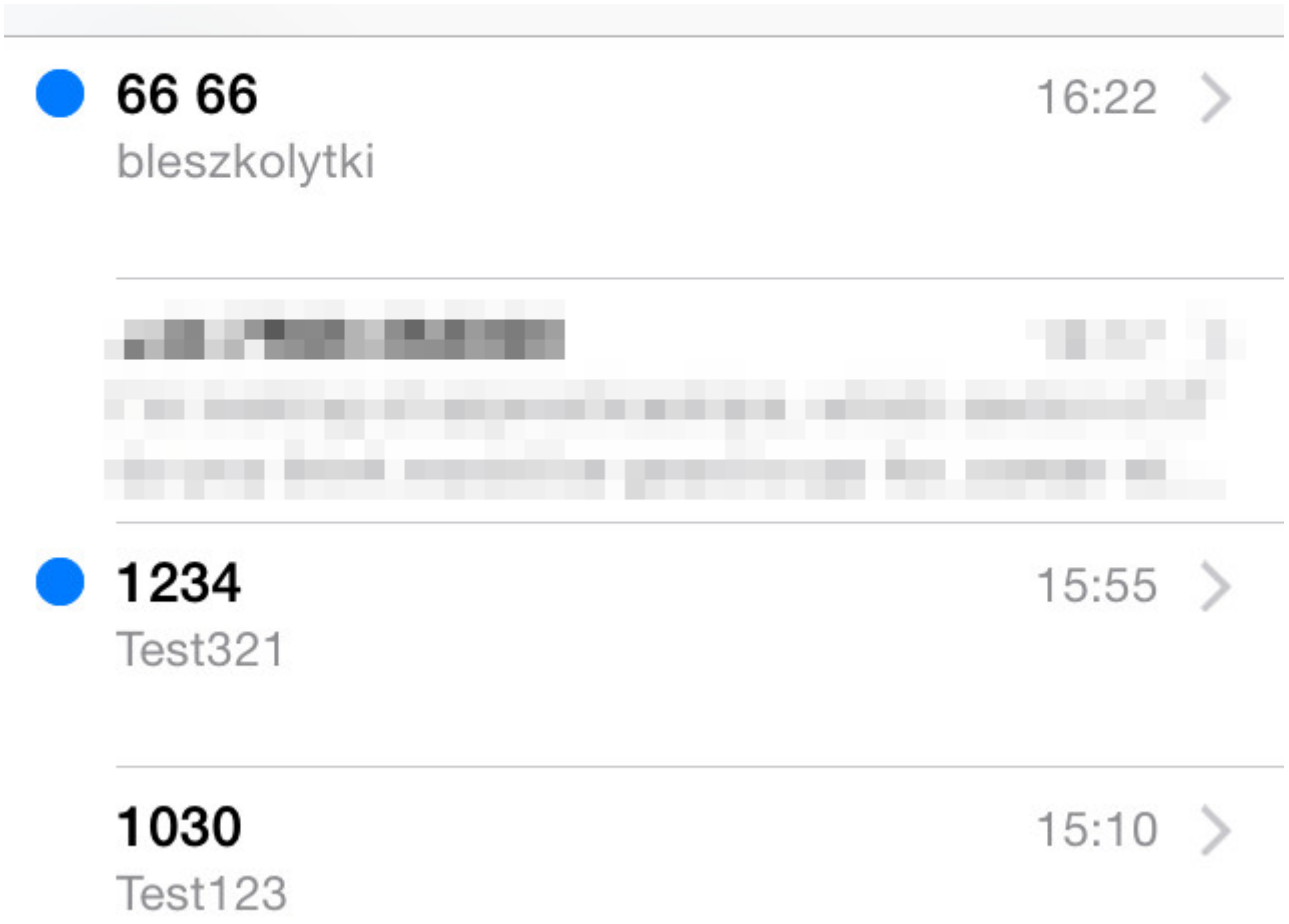
During the test voice calls you could get some additional information from the OpenBTSCLI console

```
OpenBTS> chans
CN TN chan  transaction Signal SNR  FER   TA  TXPWR RXLEV_DL BER_DL Time IMSI
     type  id         dB         pct   sym dBm   dBm      pct
```

```
Imsi=234335501150083 Tmsi=(no tmsi) Imei="") L3TI=8 Service=MOC to=2602
stateAge=(13 sec) stack=( Machine=(InCallMachine tid=106 C0T3 TCH/F CCState=active
PopState=0)))

1 transactions in table

OpenBTS>
```

Testing SMS

We can also check if smqueue is working properly and we can send and receive SMS messages using our network and assigned MSISDN number. You can send a message to yourself using an assigned number, or you can send a message to other devices connected and authenticated in your network.


66 66
16:22
bleszkolytki
1234
15:55
Test321
1030
15:10
Test123

May 14 15:10:32 GSMTB smqueue: NOTICE 1003:1066 2015-05-14T15:10:32.3 smqueue.cpp:2455:main_loop: Got SMS rqst qtag '144201--OBTSorubnzrmmaqfyrxb' from IMSI234335501150083 for smsc
May 14 15:10:32 GSMTB smqueue: NOTICE 1003:1067 2015-05-14T15:10:32.3 smqueue.h:505:get_text: Decoded text: Test123
May 14 15:10:33 GSMTB smqueue: NOTICE 1003:1067 2015-05-14T15:10:33.3 smqueue.h:505:get_text: Decoded text: Test123
May 14 15:10:34 GSMTB smqueue: NOTICE 1003:1067 2015-05-14T15:10:34.7 smqueue.cpp:318:handle_response: Got 200 response for sent msg '144201--OBTSorubnzrmmaqfyrxb' in state 12
May 14 15:55:32 GSMTB smqueue: NOTICE 1003:1066 2015-05-14T15:55:32.0 smqueue.cpp:2455:main_loop: Got SMS rqst qtag '355318--OBTSuuebbbqoumpjhlia' from IMSI234207505270905 for smsc
May 14 15:55:32 GSMTB smqueue: NOTICE 1003:1067 2015-05-14T15:55:32.0 smqueue.h:505:get_text: Decoded text: Test321
May 14 15:55:33 GSMTB smqueue: NOTICE 1003:1067 2015-05-14T15:55:33.0 smqueue.h:505:get_text: Decoded text: Test321

As you can see you can find any SMS messages sent over your network logged in this file. You can also send an SMS message to the target device directly from the OpenBTSCLI console using the following command however it will be not logged in the OpenBTS.log file.

```
OpenBTS> help sendsms
sendsms IMSI src# message... -- send direct SMS to IMSI on this BTS, addressed from source number src#.
OpenBTS> sendsms 234335501150083 6666 bleszkolytki
message submitted for delivery
```

Testing GPRS Connection

To use the GPRS connection on your target device no special APN configuration is required. You can use the configuration provided with the device, however, you need to have at least

To list all iptables rules use the command:

```
root@GSMTB:/etc/OpenBTS# iptables -t nat -L -n -v
Chain PREROUTING (policy ACCEPT 1166 packets, 211K bytes)
 pkts bytes target     prot opt in     out     source
destination

Chain INPUT (policy ACCEPT 1063 packets, 203K bytes
pkts bytes target     prot opt in     out     source
destination

Chain OUTPUT (policy ACCEPT 94 packets, 8816 bytes)
pkts bytes target     prot opt in     out     source
destination

Chain POSTROUTING (policy ACCEPT 56 packets, 6125 bytes)
```

If you don't see the MASQUERADE entry in your iptables ruleset simply run:

```
iptables-restore < /etc/OpenBTS/iptables.rules
```

All GPRS traffic will be routed through the Ethernet connection (eth0) on the laptop connected to the USRP so make sure that you are connected to the Internet.

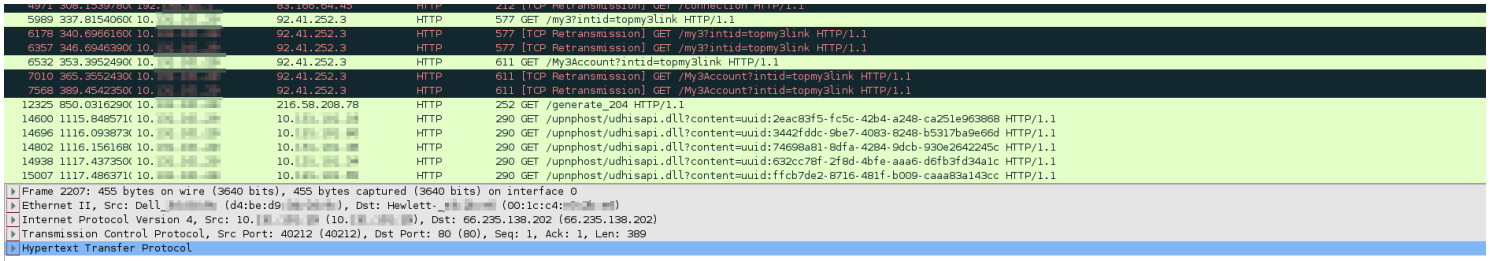
Capturing GPRS data with Wireshark

When you run the openbts service and connect a target device to your GSM network you should be able to use GPRS to connect to the Internet (the OpenBTS laptop needs to be connected to the internet). Basically all GPRS traffic is routed through the sgsn tunnel

```
eth0      Link encap:Ethernet  HWaddr d4:be:d9:34:04:9c
          inet addr:10.20.30.40  Bcast:10.20.30.255  Mask:255.255.255.0
          inet6 addr: fe80::d6be:d9ff:fe34:49c/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:73562 errors:0 dropped:0 overruns:0 frame:0
          TX packets:6515 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:11524083 (10.9 MiB)  TX bytes:1224915 (1.1 MiB)
          Interrupt:20 Memory:e6e00000-e6e20000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:1913718 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1913718 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
```

```
RX packets:5088 errors:0 dropped:0 overruns:0 frame:0
TX packets:4644 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:500
RX bytes:714308 (697.5 KiB)  TX bytes:2503539 (2.3 MiB)
```

All externally-routed IP traffic will have your eth0 IP address as a source address.



Intercepting Data with Burp

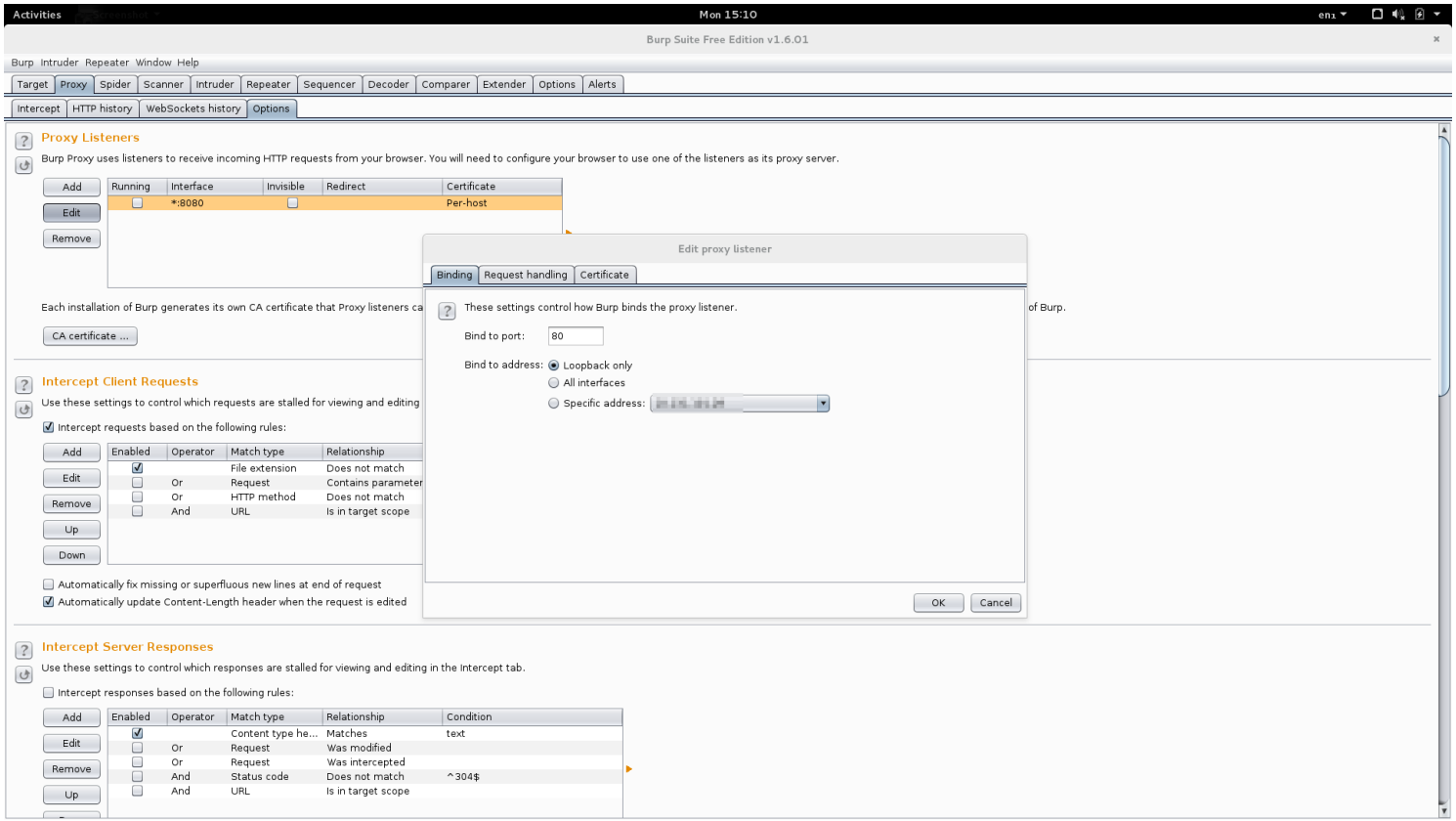
To intercept data with the Burp proxy you need to setup a reverse proxy as described in the links below:

- "Reversing" Non-Proxy Aware HTTPS Thick Clients w/ Burp
- Burp Proxy: Invisible Proxying

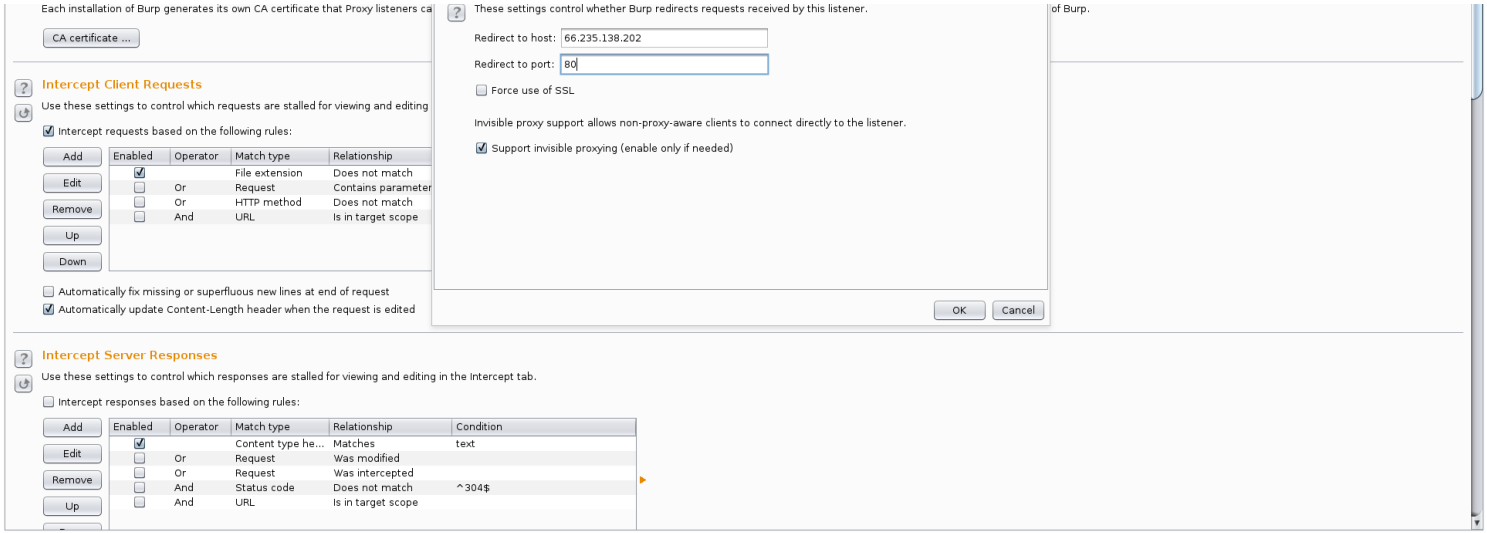
The simplest configuration in few simple steps is:

Run Burp Proxy:

than localhost):



Go to the request-handling tab and enter the real IP address of the host where you redirect all traffic.



All requests sent to your eth0 IP address on port 80 will be intercepted.

More sophisticated configurations need to use iptables to redirect traffic Burp. Take a look at an example here:

[Passing Android Traffic through Burp]

Written by Grzegorz Worona

First published on 19/05/16

Published by Matt Lewis

View all posts by Matt Lewis ->

Here are some related articles you may find interesting

Multiple vulnerabilities identified in Adobe ColdFusion allow an unauthenticated attacker to obtain the service account NTLM password hash, verify the...

- Technical advisories
- Vulnerability Research

November 21, 2023

15 mins read

Author: Alex Jessop (@ThisIsFineChief) Summary Tl;dr This post will delve into a recent incident response engagement handled by NCC Group's Cyber...

- Detection and Threat Hunting
- Digital Forensics and Incident Response (DFIR)
- Threat Intelligence

November 20, 2023

7 mins read

At Fox-IT (part of NCC Group) identifying servers that host nefarious activities is a critical aspect of our threat intelligence. One approach involves looking f...

- Cyber Security
- Detection and Threat Hunting
- Fox-IT
- Fox-IT and European Research
- Research

November 15, 2023

7 mins read

Previous post Next post

View articles by category

Most popular posts

Most recent posts

Technical Advisory: Adobe ColdFusion WDDX Deserialization Gadgets

- Annual Research Report (2)
- Asia Pacific Research (1)
- Awards & Recognition (4)
- Blockchain (3)
- Books (17)
- Business Insights (6)
- Cloud & Containerization (34)
- Cloud Security (18)

Malicious HTTP Servers by Identifying Typos in HTTP Responses

Public Report – WhatsApp Auditable Key Directory (AKD) Implementation Review

Don’t throw a hissy fit; defend against Medusa

- CTFs/Microcorruption (1)
- Current events (1)
- Cyber as a Science (6)
- Cyber Security (402)
- Detection and Threat Hunting (16)
- Digital Forensics and Incident Response (DFIR) (20)
- Disclosure Policy (1)

- Fox-IT (17)
- Fox-IT and European Research (6)
- Gaming & Media (8)
- Hardware & Embedded Systems (103)
- Intern Projects (2)
- iSec Partners (52)
- Machine Learning (28)
- Managed Detection & Response (23)

Research (28)

- Offensive Security & Artificial Intelligence (13)
- Patch notifications (35)
- Presentations (55)
- protocol_name (1)
- Public interest technology (1)
- Public interest technology (10)

Vulnerabilities at Scale (22)

- Research (362)
- Research Paper (20)
- Resources (1)
- Reverse Engineering (47)
- Risk Management & Governance (6)
- Standards (13)
- Technical advisories (215)

(68)

- Tool Release (106)
- Transport (16)
- Tutorial/Study Guide (46)
- UK Research (9)
- Uncategorized (25)
- Virtualization, Emulation, & Containerization (10)
- VSR (32)
- Vulnerability (164)

Call us before you need us.

Our experts will help you.

Get in touch

443316300690

Disclosure Policy

Training

Software Resilience