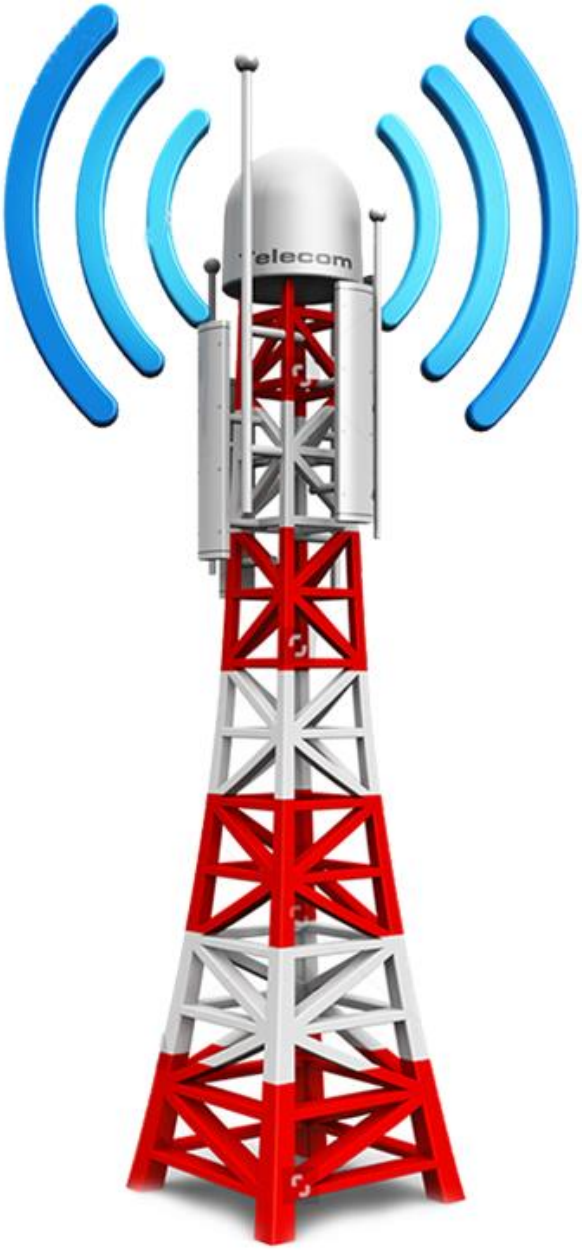


Learning mobile hacking isn't that hard !
But to found these hacking techniques isn't easy at all

Mobile Network Hacking

Who am i

Do you really want to know who am it, but I really want to know who is ready this open research now. So to make it simple visit my LinkedIn page or github let's talk?

https://www.linkedin.com/in/hassan-profile/

https://github.com/VraiHack

Resume

Why this open free **theoretical** research? Very simple, because I was curious to hack mobile network, so I learn new thing I wrote or copy it to save it for later so I can also at the end share happiness with you.

For who this research? Absolutely Not for beginner, for Telecommunication network engineer, for Pentester who have knowledge in mobile network protocol and communication good understanding for authentication, attachment, call flow etc…

What you will learn from it?

- You will understand GSM, GPRS, VOIP, LTE Threats
- You will understand SS7, GTP, Diameter protocol
- You will be able to hack mobile network on your own responsibility with building IMSI catcher with different open source software like Osmocom, srsRAN, OpenLTE with different SDR
- you will be able to understand ethical hacking in mobile telecommunication (working with different open sources tool to scan ss7), learn how to deal with IMSI catcher to test telecommunication network vulnerability, last one was a joke no its not ☺
- Please not, none of these hacking technique mentioned in this open research by myself, it's up to you to install and take the risk, try to use a faraday cage.

Table of contents

References

SS7

Things Hackers Can Do with Your Cell Phone Number | Reader's Digest
Overview - Cellular Network Infrastructure - Open Source Mobile Communications
SS7 Protocols for GSM | Telecom crash courses
GSM Network Connection to SS7 Networks - Broadband Telecommunications
Why SS7 was needed in GSM? - technopediasite-Ultimate Resource For Telecom Technical Support
SIGTRAN PROTOCOL STACK PDF
How To Scan Ports With SCTP On Nmap [Complete] - ElderNode Blog
rfc4666
Technical-report-on-the-SS7-vulnerabilities-and-their-impact-on-DFS-transactions_f-1-1.pdf
comst-2971757-pp.pdf - 08984216.pdf
Gotta Catch 'Em All: Understanding How IMSI-Catchers Exploit Cell Networks | Electronic Frontier Foundation
SCTPscan: SCTP network and port scanner - P1 Security
Queue | Telecom Signalling Attacks - SS7 to All IP - PDFCOFFEE.COM
3G: Practical Attacks Against the SS7 Signaling Protocol - Security Compass Advisory
Some Notes on Utilizing Telco Networks for Penetration Tests – Insinuator.net
GSM Security Map
Overview of GSM, GPRS, and UMTS
31c3-ss7-locate-track-manipulate.pdf
Hacking-related-books/Hacking mobile network via SS7 - interception, shadowing and more by Dmitry Kurbatov.pdf at master · pathakabhi24/Hacking-related-books · GitHub
SS7_Vulnerability_2017_A4.ENG_.0003.03.pdf
SIGNALING SYSTEM 7 (SS7) SECURITY REPORT - PDF Free Download
Attacking SS7-2009-Philipe Langlois-P1security-HES-v10.key - HES2010-planglois-Attacking-SS7.pdf
bh-eu-07-langlois.ppt - bh-eu-07-langlois-ppt-apr19.pdf
Hacking-related-books/Telecommunications Infrastructure - Security SS7 Signalling Security by Philippe Langlois.pdf at master · pathakabhi24/Hacking-related-books · GitHub
Philippe Langlois - SCTPscan Finding entry points to SS7 Networks & T…
#
https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&cad=rja&uact=8&ved=2ahUKEwisrN3_scb0AhVCz4UKHaw3DLcQFnoECBIQAQ&url=https%3A%2F%2Fwww.cellusys.com%2Fdownload%2Fss7-vulnerabilities.pdf&usg=AOvVaw3LV_m_AIuA-sujAyDY29mZ
Attacks on SS7.pdf
Signalling Security in Telecom SS7/Diameter/5G - Interconnect Security SS7-Diameter.pdf
Telecom security from ss7 to all ip all-open-v3-zeronights

GTP

GPRS Tunneling Protocol (GTP)
Vulnerabilities of Mobile Internet (GPRS)
GTP Deployments
Monitoring GTP Traffic | Securing GTP and SCTP Traffic User Guide for Security Devices | Juniper Networks TechLibrary
3GPP TS 29.274 - 29274-d70.pdf
Wireless Internet Networking Carriers Perspective ChihLin I Wireless
GTPing, How To

RADIO MOBILE HACKING

Diameter

Diameter
rfc6733
Diameter Protocol Explained: Diameter AVP Structure
3GPP spec skeleton - ts_129109v060900p.pdf
Diameter Protocol Explained: Diameter Routing Agent (DRA)
What Is AAA?
Radius vs Diameter
Diameter and 3GPP - Cellusys
Philippe Langlois - Hacking HLR HSS and MME core network elements

VOIP

IP Telephony and VoIP Tutorial - Comprehensive Guide
How to attack an infrastructure using VoIP exploitation [Tutorial] | Packt Hub

GSM

Hacking GSM: Building a Rogue Base Station to Hack Cellular Devices
Step by Step guide on how to create 2G network at your own home – Information Technology Blog
GSM with Osmocom Part 4: The Base Station Controller (BSC) – Nick vs Networking
How to Build an IMSI Catcher to Intercept GSM traffic
dpkg - How to remove/install a package that is not fully installed? - Ask Ubuntu
command-not-found.com – osmo-bts-virtual
Setting up Yate and YateBTS with the bladeRF · Nuand/bladeRF Wiki · GitHub

SMS

Quickstart With Kannel. Recently , I got opportunity to work in… | by Sudeep Parajuli | Medium
SMS, appels et courriers électroniques indésirables et/ou frauduleux | Arcep
Kannel 1.4.5 User's Guide

LTE

P1security-LTE_Pwnage v2 PL.pptx - D1T2 - Philippe Langlois - Hacking HLR HSS and MME Core Network Elements.pdf
Top 10 Cyber Threats to Private 5G/LTE Networks - Security Boulevard
7-deadly-threats-4g.pdf
LTE :Mobile Network Security
Paper Title (use style: paper title) - 20151031_100157.pdf
(PDF) Security Threats Against LTE Networks: A Survey: 6th International Symposium, SSCC 2018, Bangalore, India, September 19–22, 2018, Revised Selected Papers
securecomm_camera-ready.pdf
1510.07563.pdf
Microsoft Word - BH-whitepaper-LTE and IMSI catcher myths.docx - eu-15-Borgaonkar-LTE-And-IMSI-Catcher-Myths-wp.pdf
How to create an EVIL LTE Twin. Be very careful when playing with any… | by Adam Toscher | Medium
LTE Phone Number Catcher: A Practical Attack against Mobile Privacy
[REPO]@Telematika | W00t3k/Awesome-Cellular-Hacking
How to install GNU Radio, FFTW, RTL SDR, GrOsmoSDR, and more using PyBombs with dependencies, by rpm/deb or build from source | sMyles

us-20-Quintin-Detecting-Fake-4G-Base-Stations-In-Real-Time.pdf
Detecting false base stations in mobile networks - Ericsson
Easy_4GLTE_IMSI_Catchers_for_Non-Programmers.pdf
Hacking Cellular Networks - Lin_Huan_-_UE_Security.pdf
Blog – 4G and 5G reference software
https://www.synacktiv.com/ressources/synacktiv_mobile_communications_attacks.pdf

Volte

VoLTE in IMS | Real Time Communication
IMS VoLTE Architecture - Voice Over LTE Tutorial
VoLTE Roaming and Interconnection Standard Technology - vol15_2_037en.pdf
VoLTE Call Flow and Procedures - Voice Over IP Tutorial

5G

5G Security Vulnerabilities detailed by Positive Technologies; ITU-T and 3GPP 5G Security specs - Technology Blog
PFCP - Wikipedia
A guide to 5G network security insight report - Ericsson
5G-Implementation-Guideline-v2.0-July-2019.pdf
5G Protocol Stack - User Plane/Control Plane | NETMANIAS

2G 4G VOLTE Hack

alex14324/ss7
SigPloiter/GTScan: The Nmap Scanner for Telco
mgp25/OpenLTE: An open source 3GPP LTE implementation.
open5gs/open5gs: Open5GS is a C-language Open Source implementation for 5G Core and EPC, i.e. the core network of LTE/NR network (Release-16)
Wooniety/srsLTE-Sniffer: Stuff for srsLTE IMSI catcher
srsran/srsRAN: Open source SDR 4G/5G software suite from Software Radio Systems (SRS)
SigPloit – Telecom Signaling Exploitation Framework – SS7, GTP, Diameter & SIP – Julio Della Flora
ss7MAPer – A SS7 pen testing toolkit – Insinuator.net
P1 Labs » Presenting QCSuper: a tool for capturing your 2G/3G/4G air traffic on Qualcomm-based phones
5 best open source bladerf projects.
ernw/ss7MAPer: SS7 MAP (pen-)testing toolkit. DISCONTINUED REPO, please use: https://github.com/0xc0decafe/ss7MAPer/
SecuraBV/SIPWatcher
proceedings-2016/05_LTE_Security_and_Protocol_Exploits.md at master · shmoocon/proceedings-2016 · GitHub
使用GnuRadio + OpenLTE + SDR 搭建4G LTE 基站（上） TYINY的博客-CSDN博客

android

How to Check if Your Android Phone is Rooted
Kali NetHunter | Kali Linux Documentation
GitHub - urbanadventurer/Android-PIN-Bruteforce: Unlock an Android phone (or device) by bruteforcing the lockscreen PIN. Turn your Kali Nethunter phone into a bruteforce PIN cracker for Android devices! (no root, no adb)
GitHub - Ondrik8/HARD_device_attack

Training

Telecom Security Hands-on Course | Training | Course | Training Center - TeleScope
Mobile Device Hacking with SDR | Training Live Streams
3-DAY TRAINING 6 – Hacking Mobile Networks with Software Defined Radios « JD-HITBSecConf2018 – Beijing
Trainings | P1 Security | Telecom Security Network World Leader
4G IMSI Catcher | IMSI Catcher | IMEI Catcher | TMSI Catcher | LTE catcher
Electromagnetic Field GSM Network - Lime Microsystems
CableLabs Launches 10G Challenge: Powering the Future of Broadband Innovation
5G Penetration Testing and Ethical Hacking Training - Tonex Training
Utiliser un Raspberry Pi pour détecter les IMSI Catchers | Silicon
Ensuring SS7 Network Security - Newsletter

Credits for:

OSMOCOM : https://osmocom.org/projects/cellular-infrastructure

BIG Thanks for: *nicksvsnetworking*

This resume can't be done without the awesome work of nickvsnetworking, make sure to visit his website to check his last work
https://nickvsnetworking.com/

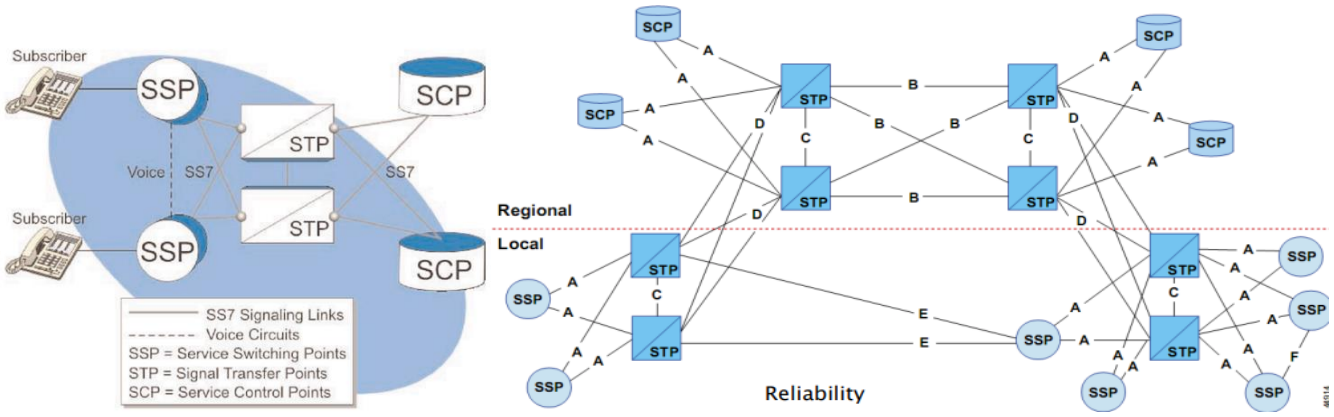
1 SS7 | Sigtran | GTP | Diameter

1.1 SS7 in PSTN network

Signaling System 7 (SS7) is an international telecommunication protocol standard that defines how the network elements in a public switched telephone network (PSTN) exchange information and control signals. Nodes in an SS7 network are called *signaling points*.

It is the system that controls how telephone calls are routed and billed, and it enables advanced calling features and Short Message Service (SMS). It may also be called Signalling System No. 7, Signaling System No. 7 or -- in the United States -- Common Channel Signaling System 7, or CCSS7.

SS7 was first adopted as an international standard in 1988, and the latest revision of the standard was in 1993. It is still the current standard for telephone calls and is in use for both landline and mobile phone service all the way up to and including 5G.

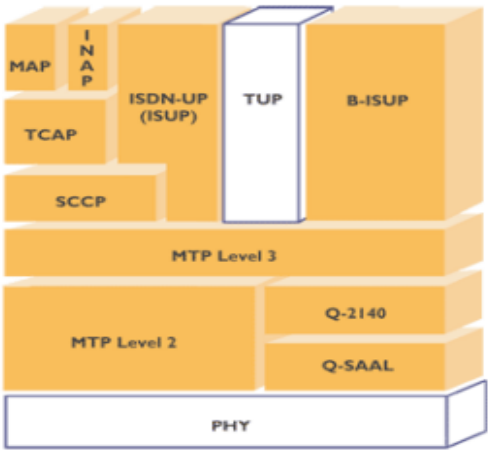


Service Switching Points (SSP) are the telephone "switches" that are interconnected to each other by SS7 links. The SSPs perform call processing on calls that originate, tandem, or terminate at that site.

Signal Transfer Points (STP) are "routers" that relay messages between network switches and databases. Their main function is to route SS7 messages to the correct outgoing signaling link, based on information contained in the SS7 message address fields.

Service Control Points (SCP) contains centralized network databases for providing enhanced services. Examples of services include toll-free numbers and prepaid subscriptions.

1.1.1 SS7 protocol stack



MTP (Message Transfer Part) Layers 1-3: lower level functionality at the Physical, Data Link and Network Level. They serve as a signaling transfer point, and support multiple congestion priority, message discrimination, distribution and routing.

ISUP (Integrated Services Digital Network User Part): network side protocol for the signaling functions required to support voice, data, text and video services in ISDN. ISUP supports the call control function for the control of analog or digital circuit switched network connections carrying voice or data tra!c.

SCCP (Signaling Control Connection Part): supports higher protocol layers such as TCAP with an array of data transfer services including connection-less and connection oriented services. SCCP supports global title translation (routing based on directory number or application title rather than point codes), and ensures reliable data transfer independent of the underlying hardware.

TCAP (Transaction Capabilities Application Part): provides the signaling function for communication with network databases. TCAP provides non-circuit transaction based information exchange between network entities.

MAP (Mobile Application Part): provides inter-system connectivity between wireless systems, and was specifically developed as part of the GSM standard.

INAP (Intelligent Network Application Part): runs on top of TCAP and provides high-level services interacting with SSP, SCP and SDP in an SS7 network.

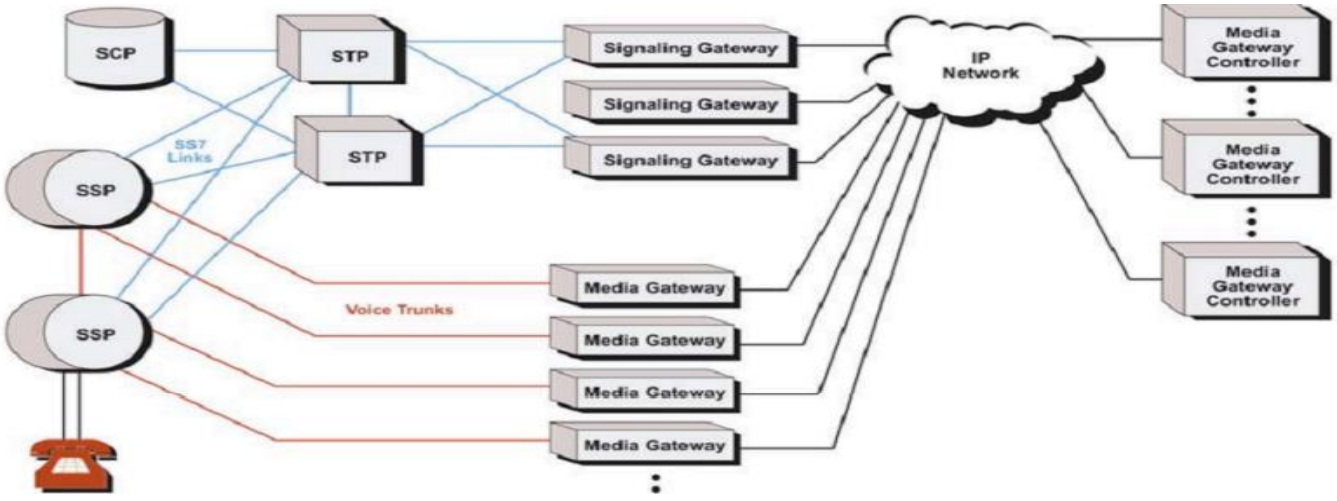
1.2 SIGTRAN in IP Telephony network

IP telephony has to do mainly with digital telephony systems (LAN based IP PBX systems) which use the IP protocol entirely for voice communication.

All components of the IP telephony system use digitized voice which is transferred as IP packets through an IP network (usually the LAN network).

The call control system is usually a software based (softswitch) server or even a hardware device like the Cisco Call Manager Express, which handles all call signaling, call routing, IP phone management etc, again using IP protocol for transport. So think about IP telephony as a bigger concept compared to VoIP.

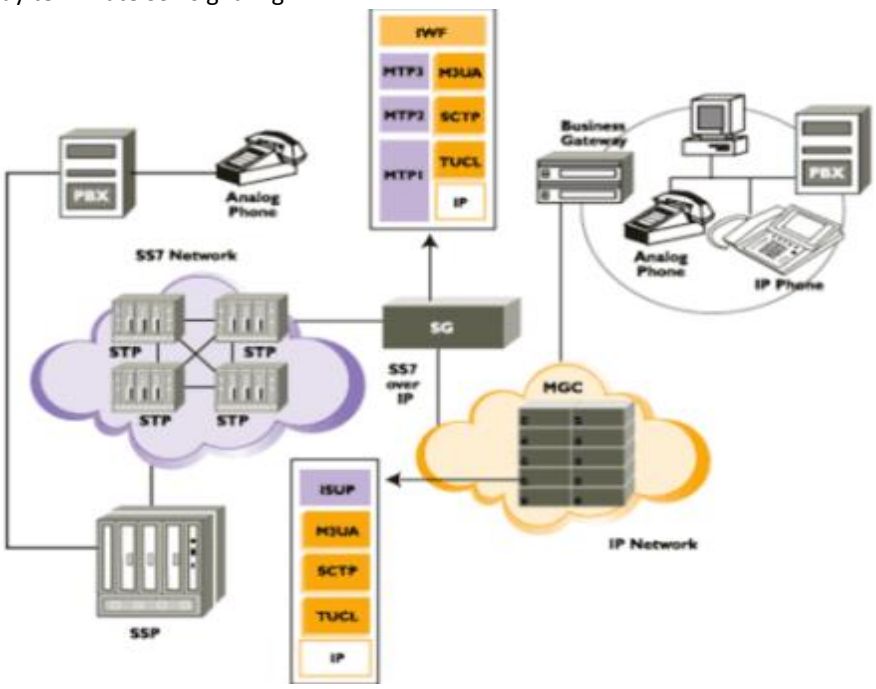
IP Telephony is the overall concept of the modern form of voice communication which harnesses the power and features of VoIP technology in order to offer the overall experience of communicating effectively and with lots of extra features.



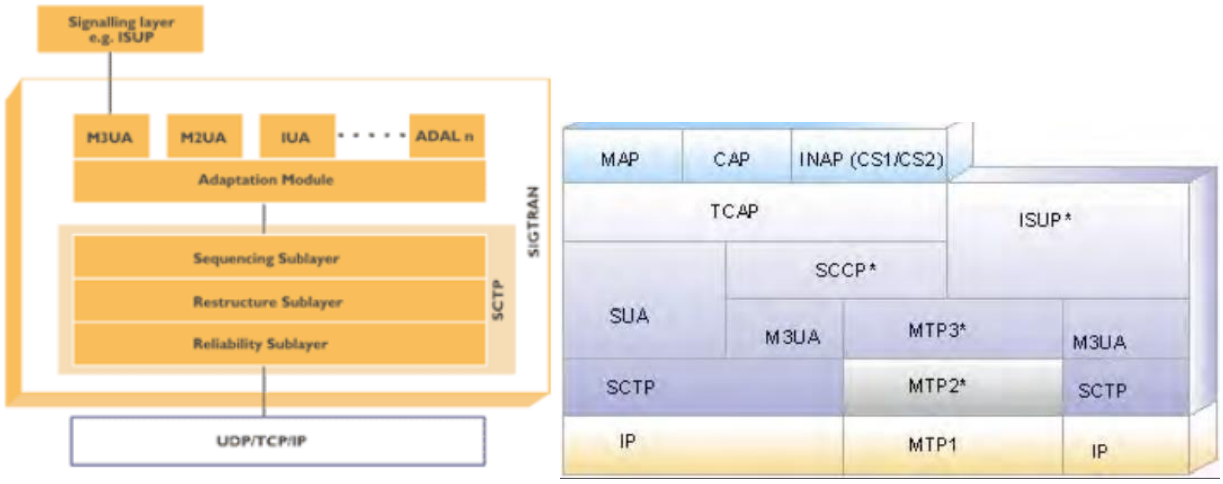
Media Gateway (MGW) terminates voice calls on inter-switch trunks from the PSTN, compresses and packetizes the voice data, and delivers voice packets to the IP network. For ISDN calls from the PSTN, Q.931 signaling information is transported from the MGW to the media gateway controller for call processing.

Media Gateway Controller (MGC) handles the registration and management of resources at the media gateways. An MGC exchanges ISUP messages with CO switches via a signaling gateway. Sometimes called a softswitch.

Signaling Gateway (SGW) provides transparent interworking of signaling between switched circuit and IP networks. The SGW may terminate SS7 signaling



1.2.1 SIGTRAN protocol stack



The SIGTRAN protocols specify the means by which SS7 messages can be reliably transported over IP networks.

The architecture identifies two components: a common transport protocol for the SS7 protocol layer being carried and an adaptation module to emulate lower layers of the protocol. For example:

♣ If the native protocol is MTP (Message Transport Layer) Level 3, the SIGTRAN protocols provide the equivalent functionality of MTP Level 2.

♣ If the native protocol is ISUP or SCCP, the SIGTRAN protocols provide the same functionality as MTP Levels 2 and 3.

♣ If the native protocol is TCAP, the SIGTRAN protocols provide the functionality of SCCP (connectionless classes) and MTP Levels 2 and 3.

1.3 VOIP

The telephone handsets (VoIP phones) translate the analogue voice signal into digital voice (binary voice) which is transferred as IP packets from one phone to another.

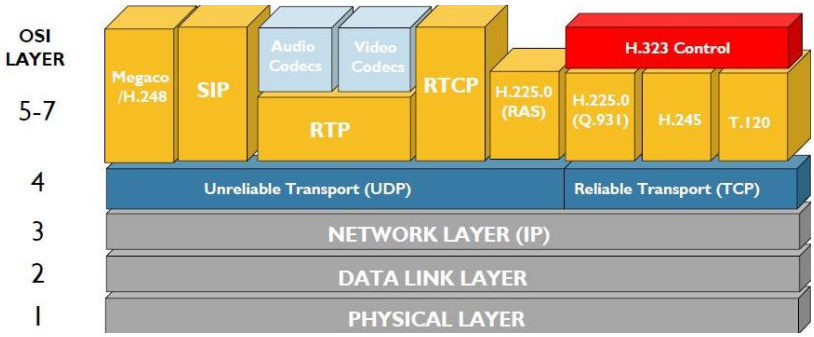
VoIP on the other hand is a subset of IP Telephony. Basically, VoIP is the technology which is used by IP Telephony as the vehicle to transport phone calls.

VoIP is the technology in which the analogue voice signal is digitized (analog to digital conversion) and becomes binary numbers in order to be transferred by the IP protocol.

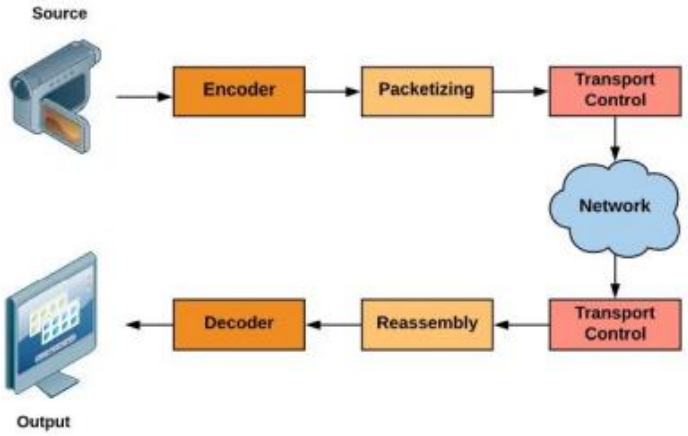
VoIP is the basis for the implementation and functionality of an IP Telephony system. VoIP can also be used by legacy TDM based PBX systems to transport voice calls over an IP WAN network or even over the Internet.

Special voice gateways are used to connect to the legacy PBX telephone system on one end and to the IP network on the other end in order to translate the TDM voice stream into IP voice packets.

1.3.1 VOIP Protocol Stack



- **Real-time Protocol (RTP)** is a transport protocol, specifically over UDP, based on RFC 3550. It is used in real-time multimedia applications and in end-to-end real-time data stream transfer. In order to achieve that, a video, for example, goes through a number of steps:
  - Encoding
  - Packetizing
  - Transport Control
  - Reassembly
  - Decoding



- Although RTP is specified to carry the media stream, there is another protocol that works with RTP called **Real-time Control Protocol (RTCP).** This protocol works side by side with RTP to monitor transmissions and assure Quality of Service (QoS). The aim of RTCP is checking whether there is packet loss during the process.

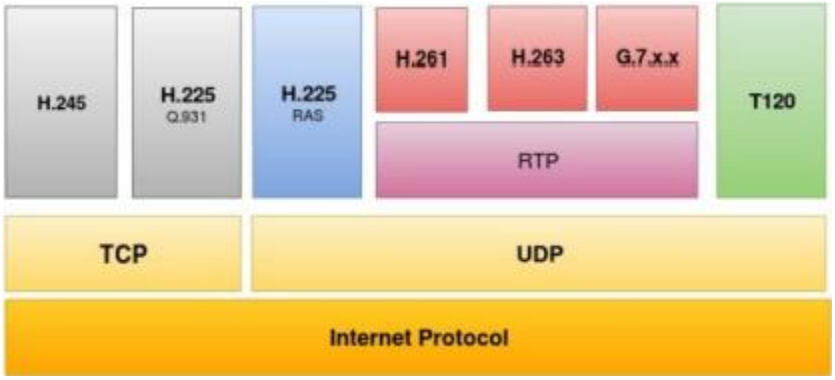
- **H.323** is a data over IP standard introduced by the International Telecommunication Union Standardization Sector (ITU-T). As you can see, this standardization body uses letters to define the scope based on many criteria, listed here:
  - H: For audiovisual and multimedia systems
  - G: For transmission systems and media
  - Q: For switching and signaling
  - T: For terminals for telematic services

- H.323 is one of the oldest packet-based communication systems protocols. Thus, this protocol is stable. The current version is v6. It is well used by many vendors in many products, such as Cisco call manager, NetMeeting, and RadVision.

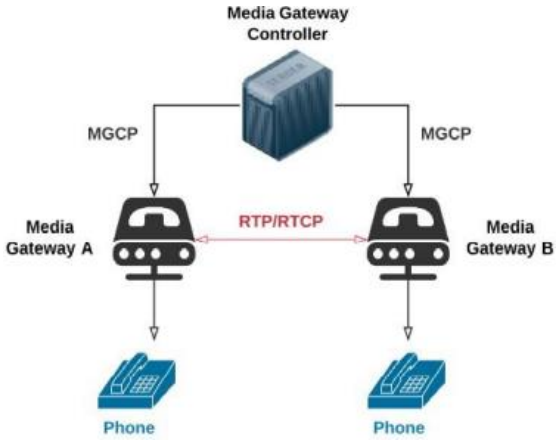
- H.323 uses many types of devices:
  - Terminals: These are user devices such as IP phones and videoconferencing systems.
  - Multipoint control units: These are composed of two logical components— the Multipoint Controller (MC) and the Multipoint Processor (MP). Their role is managing multipoint conferences.
  - Gatekeeper: This is optional. Gatekeepers provide some additional services such as user authentication and address resolution.

- The H.323 stack is based on the following components:
  - IPv4 network layer
  - User datagram protocol layer
  - Real-time protocol
  - Signaling protocols
  - Pre-call setup
  - Video codecs
  - Audio codecs
  - Data

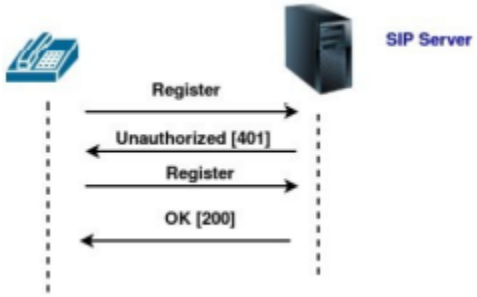
- The following diagram illustrates the different components of the H.323 stack



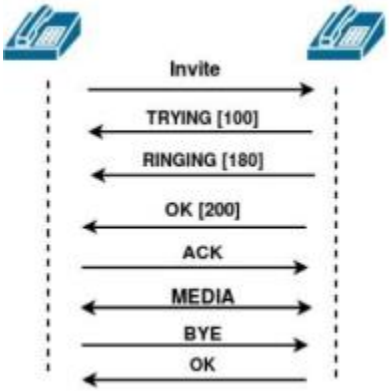
- **Media Gateway Control Protocol (MGCP)** is a protocol developed by Cisco. The goal of MGCP is to handle signals and session management. It is a communication mechanism between media gateway controllers and media gateways. Thus, the control is centralized.
  In other words, the controller communicates with many media gateways. The controller also supervises terminals and registers the new ones in its zone. H.248 is also like H.323, an ITU-based protocol. It is an enhanced version of MGCP. As you can see in the diagram, MGCP is a master-slave protocol:



- **Session Initiation Protocol Session Initiation Protocol (SIP)** is a session management protocol based on the RFC 3261 protocol. It works on both UDP and TCP, and it also supports TLS. It is more scalable than H323. SIP handles calls in the following five steps:
  - User location
  - User availability
  - User capability
  - Session set up
  - Session management
- To start a SIP operation, a registration is needed by the user:



- The following diagram describes the steps required to establish a connection between two user agent clients:



- SIP requests are similar to HTTP requests. They are in the following format:
  METHOD URI SIP/X.X
  HEADER: XXX

- Here, the method is the request type, and we have the following six methods:
  - Register

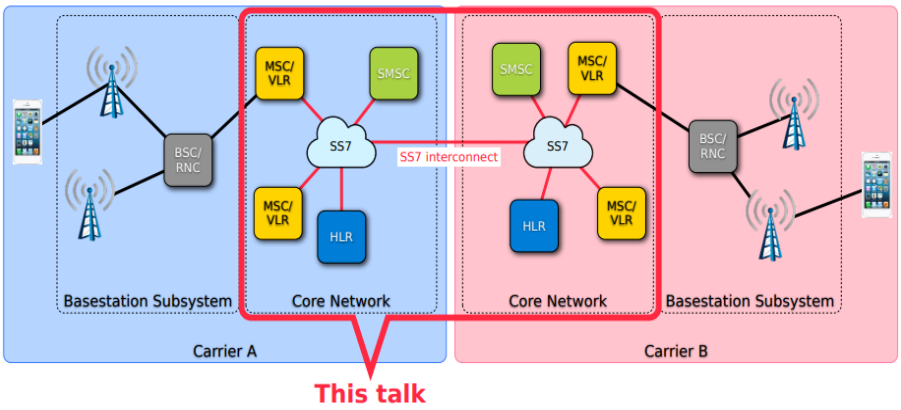
- Invite
- ACK
- Cancel
- Options
- Bye

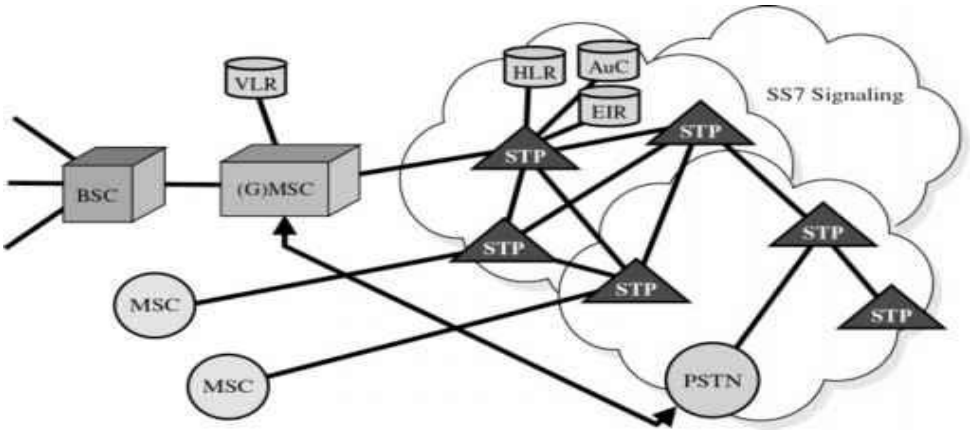
- SIP reply requests require this format:
  - SIP/X.X description
  - Header: XXX
  - URI: The file identification
  - SIP/X.X: SIP version
  - Header: This contains the information about the receiver (To, From, Call-ID are some of the SIP header fields)

- Following are the possible status codes:
  - 1xx: Informational
  - 2xx: Success
  - 3xx: Redirection
  - 4xx: Failure
  - 5xx: Server error
  - 6xx: Global failure

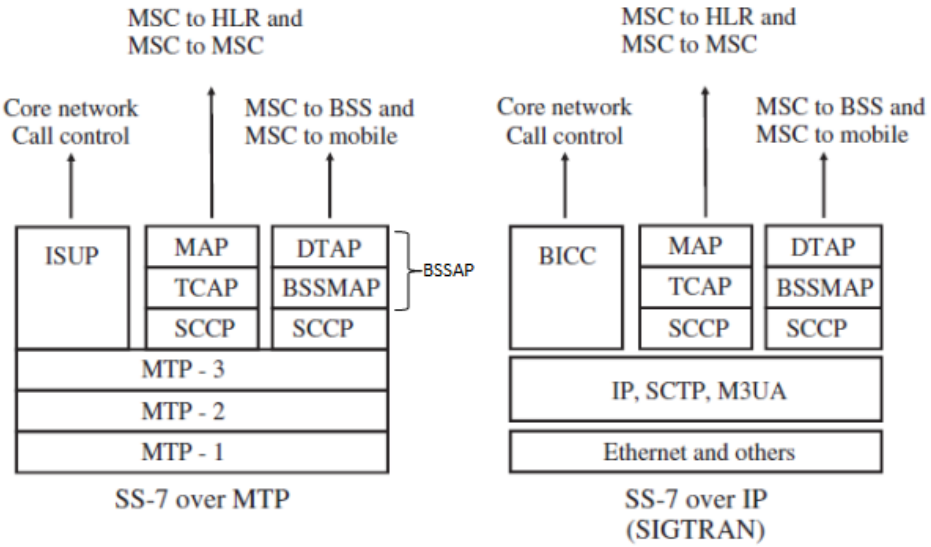
1.4 SS7/Sigtran in GSM network

The MSC is the Central Switching function of the GSM network. The MSC is connected to a SS7 network for the purpose of signaling and performing database queries

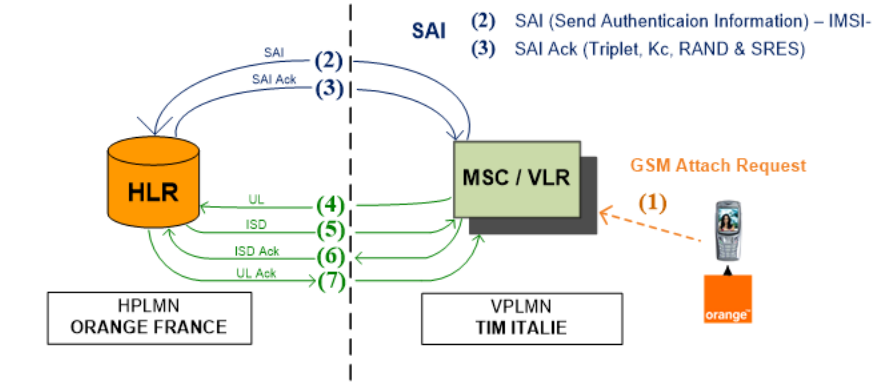




1.4.1 SS7/Sigtran protocol stack in GSM



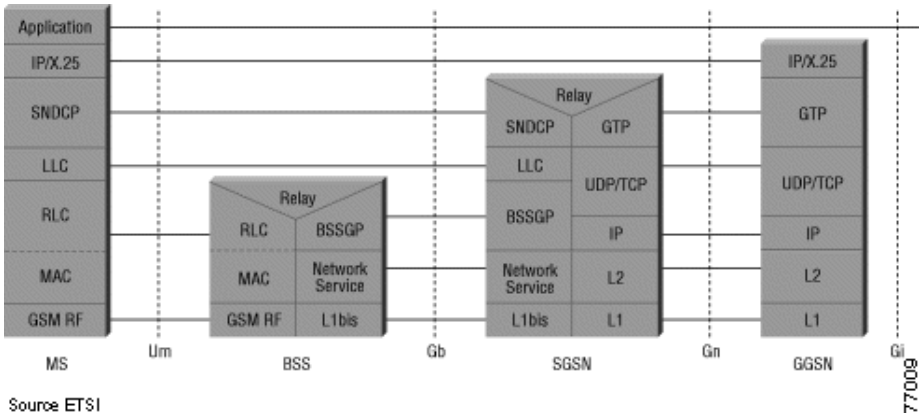
1.4.2 GSM attachment



(4) UL MAPV2 (Update Location) - IMSI, @VLR -
(5) ISD (Insert Subscriber Data) MAPV3 – MSISDN, ODB, CF, SS,...-
(6) ISD Ack
(7) UL Ack

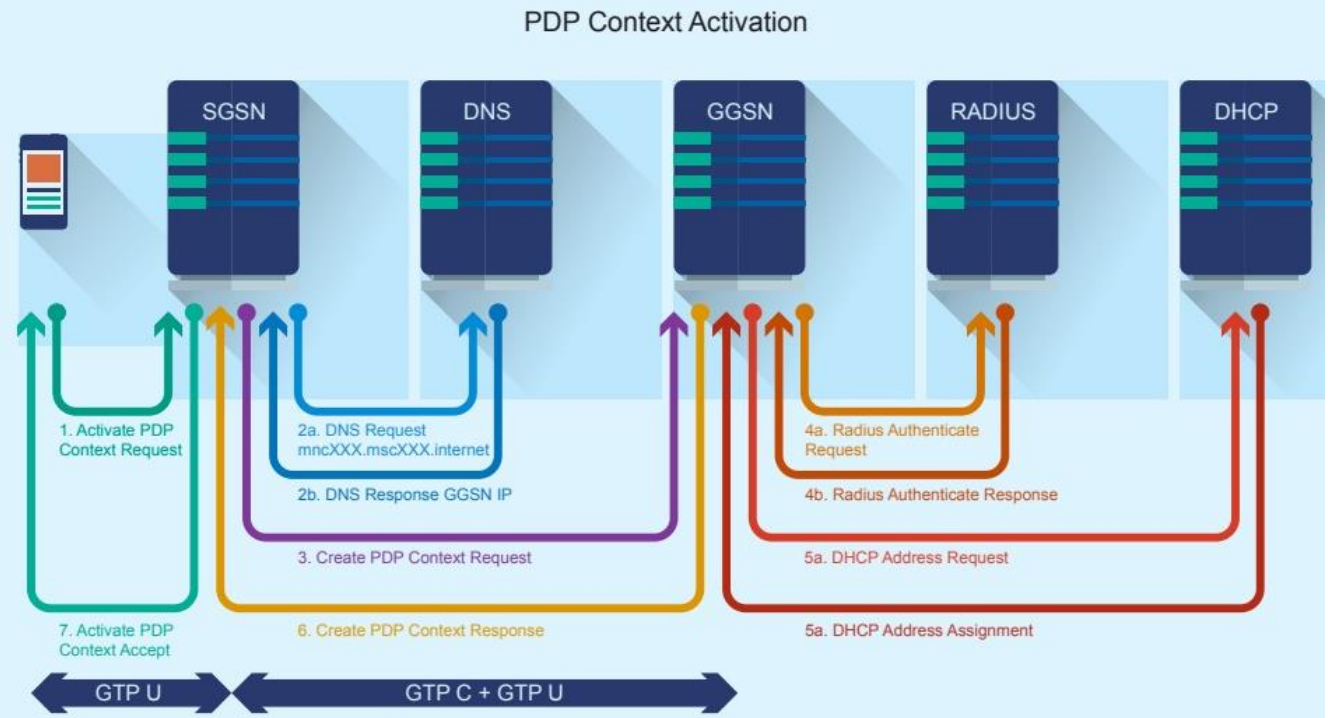
LOCATION UPDATE

1.5 GTP in GPRS network



Source ETSI

- **How communication work between SGSN and GGSN**

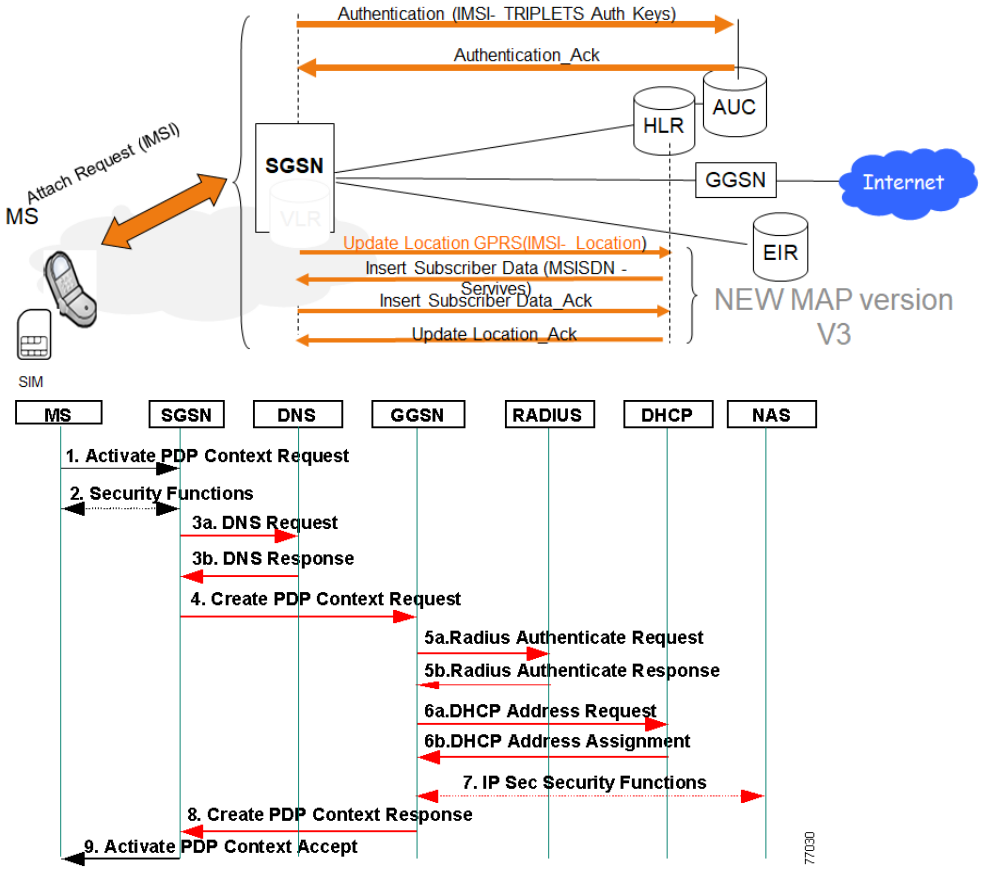


GTP uses tunnels to allow two GPRS support nodes (GSNs) to communicate over a GTP-based interface and to separate traffic into different communication flows.

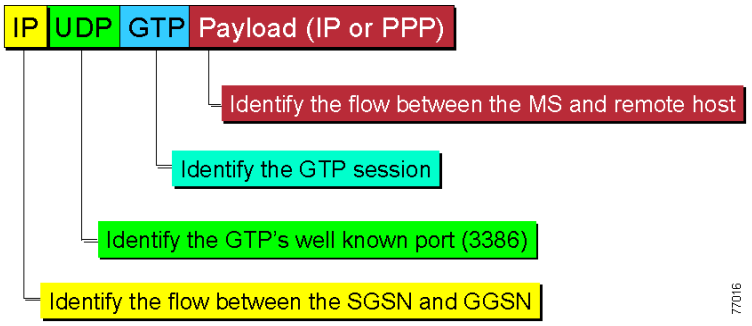
GTP creates, modifies, and deletes tunnels for transporting IP payloads between the user equipment, the GPRS support nodes (GSNs) in the GPRS backbone network and the internet.

GTP comprises three types of traffic—control plane (GTP-C), user plane (GTP-U), and charging (GTP' derived from GTP-C) traffic.

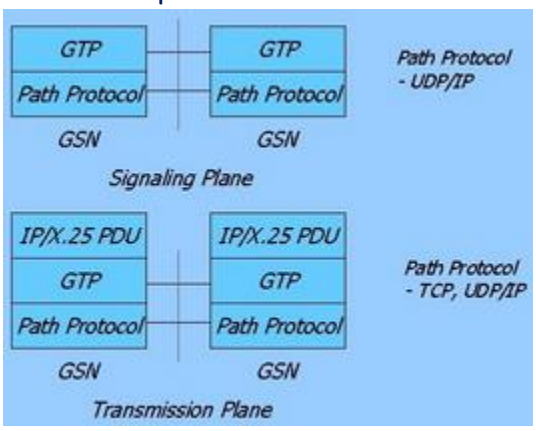
1.5.1 GPRS attachment and Activation



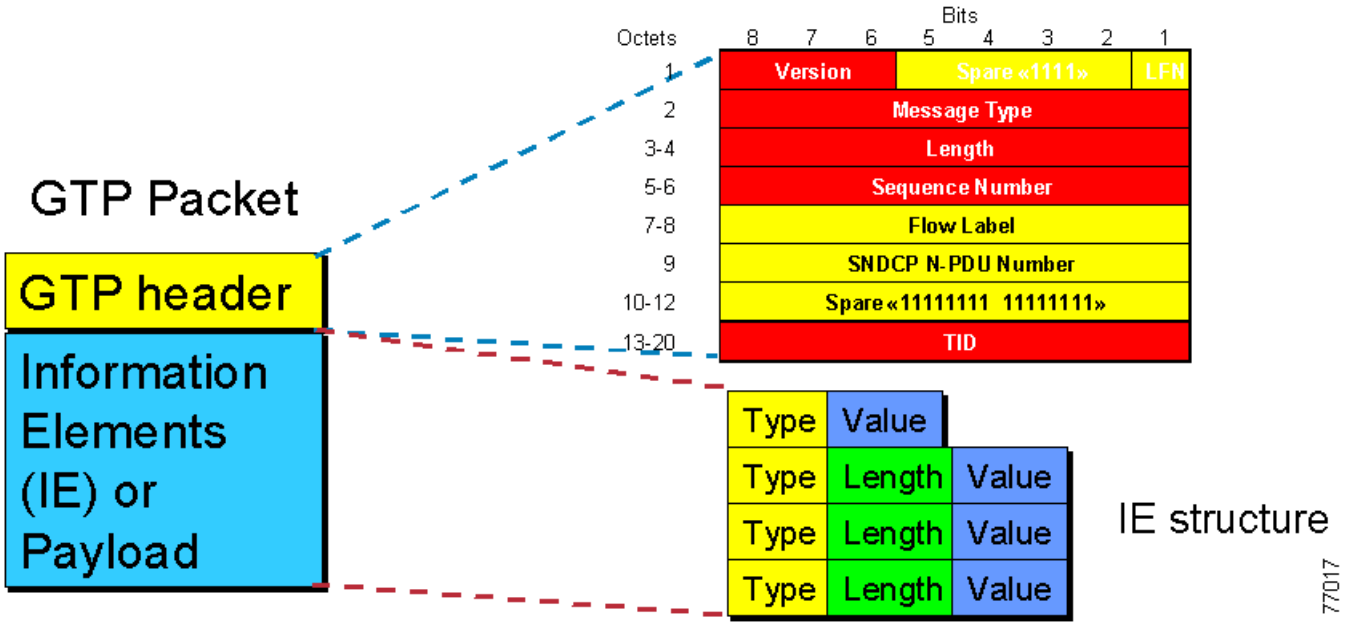
1.5.2 GPRS Tunneling Protocol



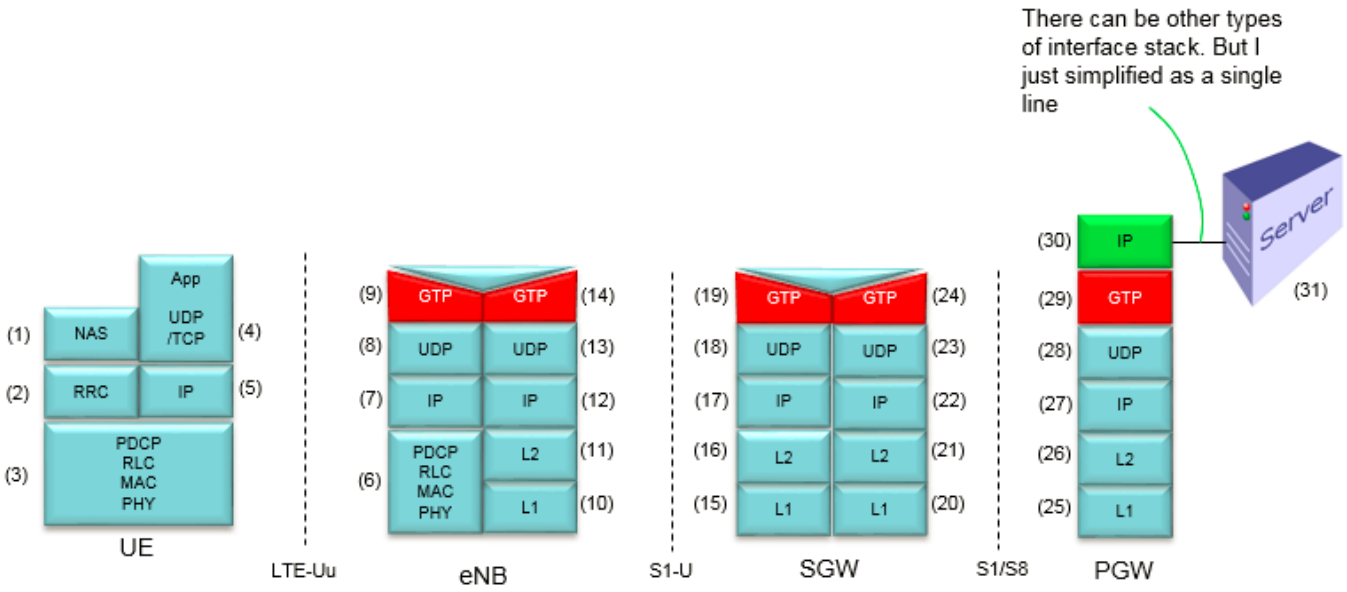
1.5.3 GTP protocol stack



1.5.4 GTP packet header



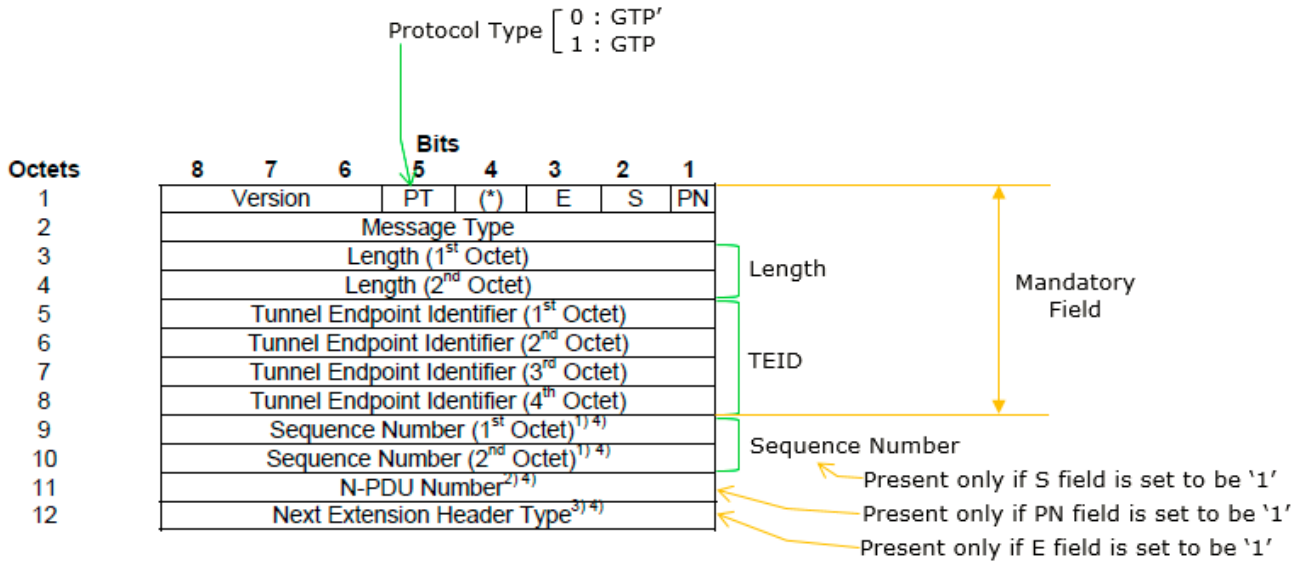
1.6 GTP in LTE network



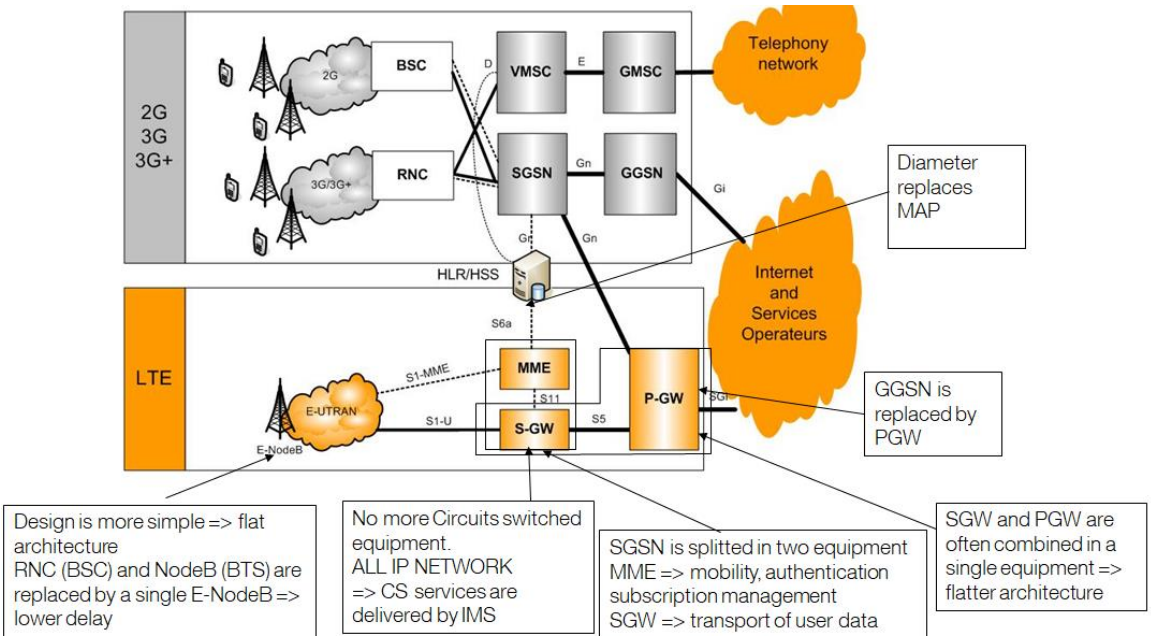
1.6.1 GTP packet header

Following is the GTP Header format. User Data (usually IP data) is encapsulated by a GTP packet following this header as shown in example section.

< 3GPP 29.281 - Figure 5.1-1: Outline of the GTP-U Header >



1.7 Diameter in LTE



- The **Diameter protocol**, standardized by the IETF Authentication, Authorization and Accounting (AAA) working group, is the successor to the RADIUS protocol and was developed to overcome several limitations of RADIUS.

  AAA protocols such as TACACS+ and RADIUS were initially deployed to provide dialup Point-to-Point Protocol (PPP) and terminal server access.

  Over time, with the growth of the Internet and the introduction of new access technologies, including wireless, DSL, Mobile IP, and Ethernet, routers and network access servers (NAS) have increased in complexity and density, putting new demands on AAA protocols.

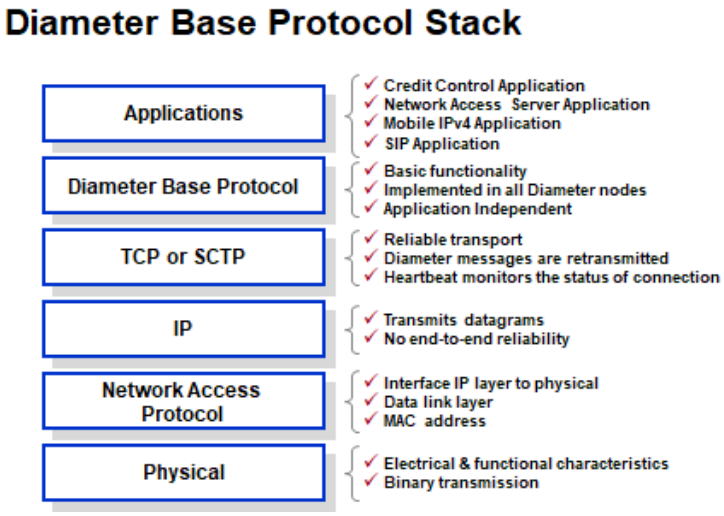
RADIUS	DIAMETER
Connectionless	Connection Oriented
Uses UDP	Uses TCP or SCTP
Unreliable	Reliable
UDP Ports 1812/1813 and 1645/1646	TCP and SCTP port 3868
Hop by Hop Security	Hop by Hop and End to End Security
No Capability Negotiation	Application and Security Level Negotiation
No Server Initiated Message	Server Initiated Message is used
Static Configuration	Static and Dynamic Configuration
Vendor Specific Attributes	Vendor Specific Attributes and Messages

1.7.1 Diameter base

Diameter is composed of **a Base protocol** to which an extension called "Application" must be added.

Diameter Base provides mechanisms for: Reliable transport, Error handling, Accounting, Capabilities negotiation

Diameter Base does not provide the messages necessary for authentication and authorization. They are specific to each application.



1.7.2 Diameter application

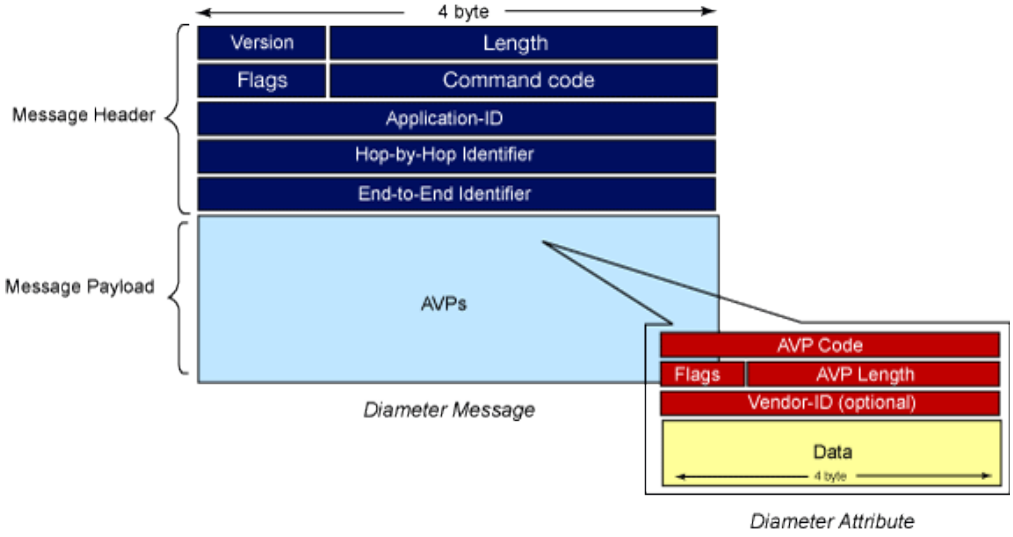
A Diameter application has nothing to do with the "software" meaning of the term. This is a protocol based on Diameter Base.

An application defines a set of commands (requests and responses) and Attribute-Value pairs (AVP) to enable authentication and authorization in a given context.

The applications are described in RFCs and 3GPP specifications. Below, some applications standardized by the IETF:

- RFC 4004 Diameter Mobile IPv4 Application (rfc4004)
- Diameter Network Access Server Application (rfc4005)
- Diameter Extensible Authentication Protocol (EAP) Application (rfc6733)
- Diameter Session Initiation Protocol (SIP) Application (rfc4740)

1.7.3 Diameter message format



- **Length:** Contains the size of the message (header + payload)
- **Flags:** The flags are:

  'R' Used to indicate whether the message is a request or a response.

  'P' Used to indicate whether the message can be forwarded or not.

  'E' Used to indicate whether the message contains an error (valid only for replies).

  'T' Used to indicate whether the message is a forwarded message.
- **Command Code:** Code to uniquely identify each type of request. The response to the request has the same code. However, its 'R' bit allows it to be differentiated from the request.
- **Application-ID**: Each Diameter Application has a unique identifier. This field therefore makes it possible to identify the application concerned by the message. If it is a message defined in Diameter Base then this ID is '0'.
- **Hop-by-Hop identify:** A Diameter architecture can be made up of a client, a server and several intermediate Diameter nodes (cf. Diameter Agents). Each message therefore has a hop-by-hop identifier, in other words, modified by each node crossed by the message.
- **End-to-End identifier:** End-to-end identifier of a message. The response to a request has the same identifier as this request. Thus, a client can detect the response to a request that it has issued.
- **The payload**: is made up of a set of attribute-value pairs (AVP). .

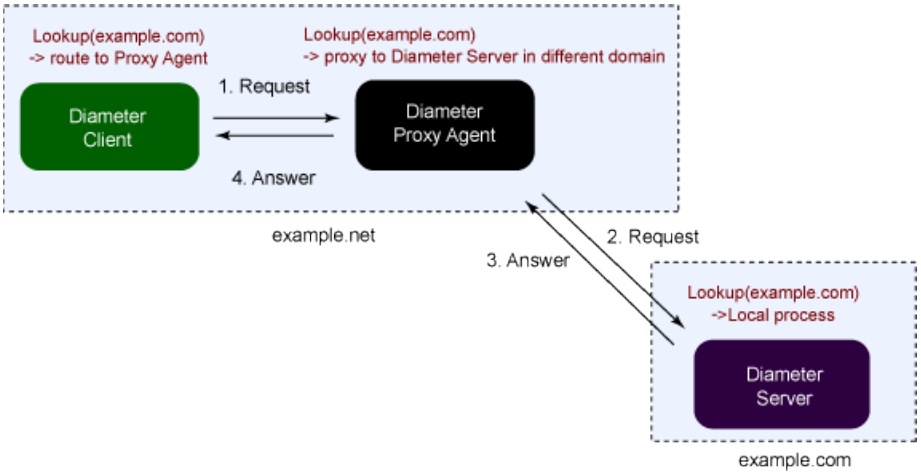
1.7.4 Diameter architecture

A Diameter architecture is made up of different nodes called “Diameter agents”. Each agent has a well-defined role in the protocol specifications. All agents can initiate queries. In this sense, they form a peer-to-peer network. The four types of existing agents are:

- **The relay agent:**
  The role of this agent is to route the messages to the correct destination according to the information contained therein such as the application id or the "Destination-Realm" AVP.

  Typically, the Proxy is placed between the Diameter client and several servers of different applications. Thus, depending on the target application of a request sent by the client, the proxy will be able to transmit it to the right server.

  This avoids configuring the client to take each server into account. In addition, the proxy can modify the messages by adding AVPs. In this context, he can for example modify the access control to certain resources for a given domain

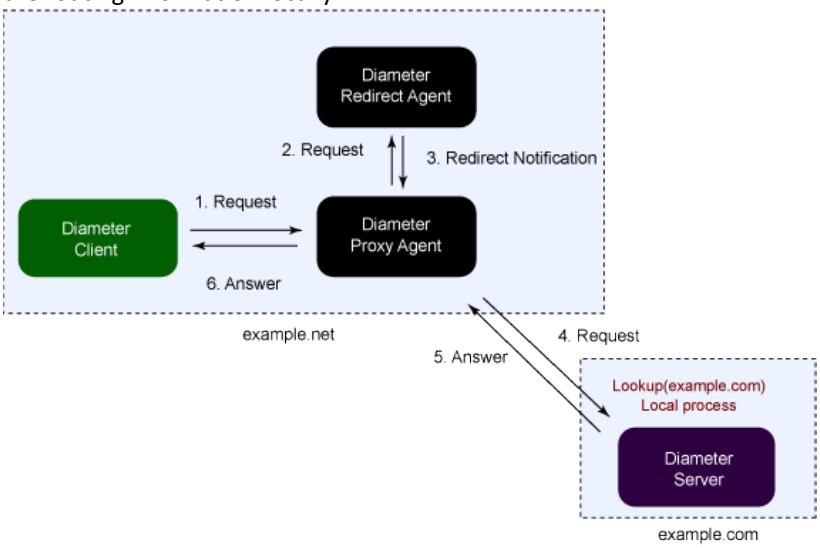


- **The proxy agent**
  The role of a relay is the same as a proxy except that it cannot modify messages.

- **The redirect agent**

  The redirection agent centralizes routing information. It can be queried by any node not knowing where to send a message. The redirect agent then responds with the redirect information.

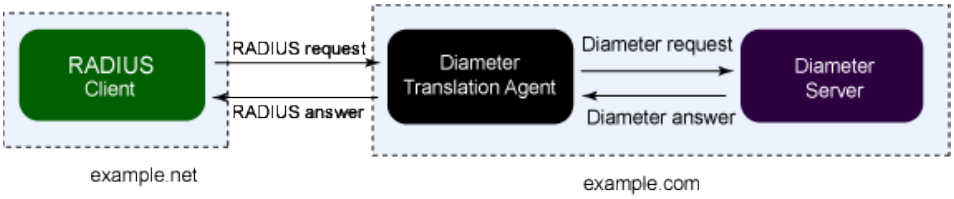
  The use of this agent makes it possible to alleviate the local configurations to the nodes which no longer need to keep the routing information locally.



- **The translation agent**

  The translation agent allows Diameter's interoperability with other AAA protocols. Take the example of RADIUS. The translation agent changes RADIUS messages to their Diameter equivalent (if any).

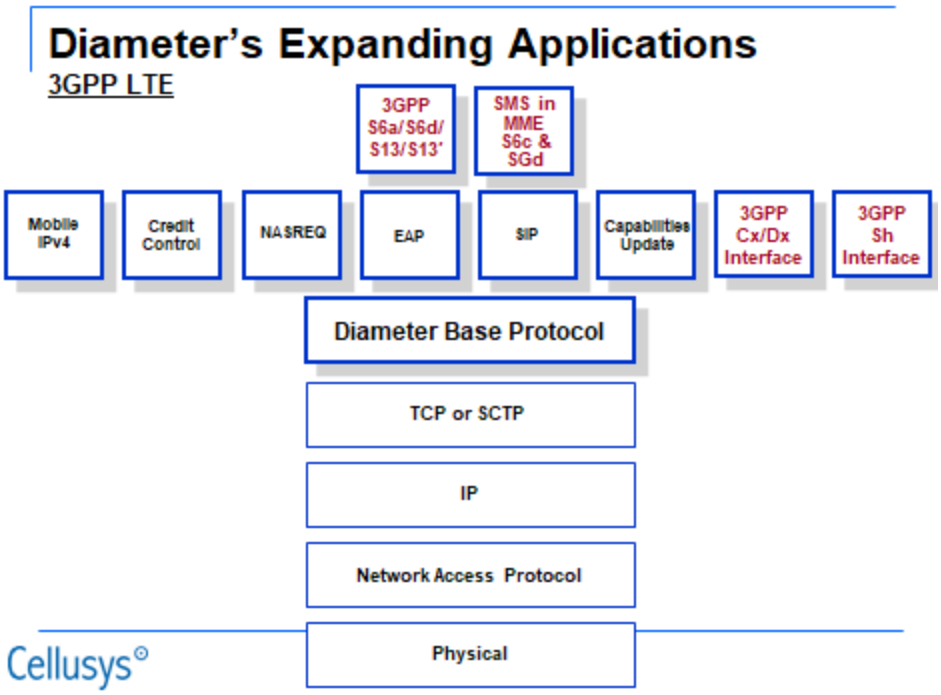
  The benefit may be to ensure a smooth migration to Diameter, retaining RADIUS clients and servers during the transition phase.



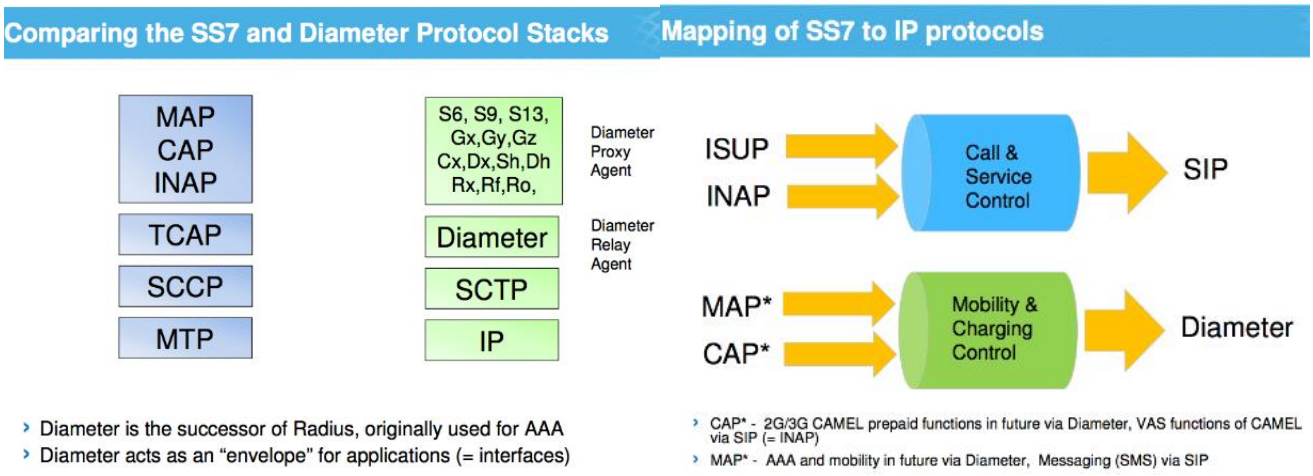
1.7.5 Diameter protocol stack

2 In previous 3GPP releases and architectures, functions like managing the mobile's location, handling subscriber data, authentication, fault recovery and checking the ***Mobile Equipment's*** (ME) identity were all handled using SS7.

3 Now starting with this new architecture in LTE EPS, those functions are handled by Diameter on interfaces called "S6a/S6d/S13/S13." In addition, LTE networks allow an optional architecture called SMS in MME. And, yes that also uses Diameter on interfaces "S6c and SGd." So now our Diameter application tree looks like this:



4



The s6a interface is between MME and HSS in the LTE network and **s6d is between SGSN and HSS**.

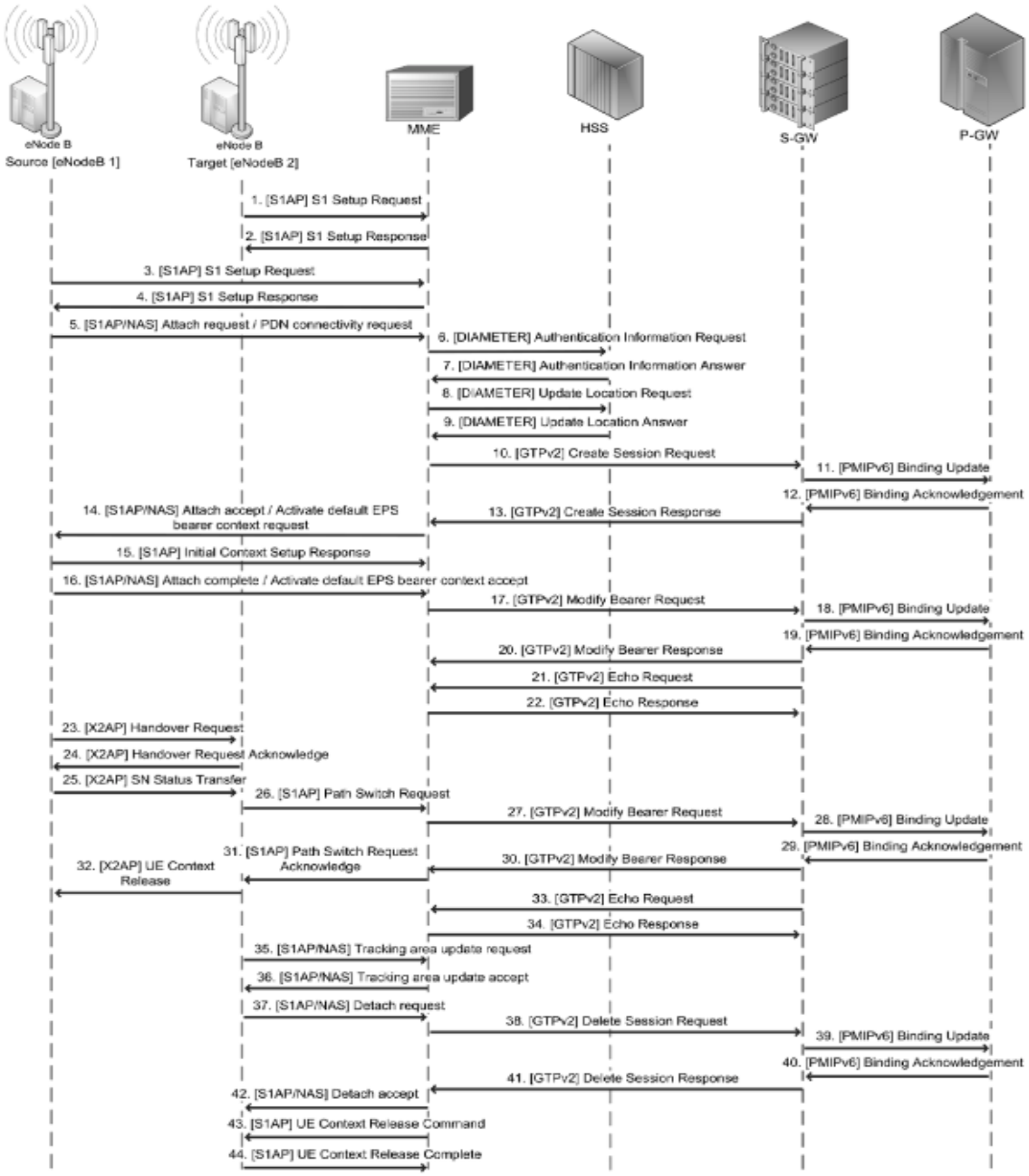
1.7.7 Summary

As a recap, Diameter is an IETF-defined AAA protocol. It is in a *Request/Answer* format. It delivers parameters called ***Attribute Value Pairs*** (AVP). Along with the base protocol, the IETF wrote several applications for Diameter. Additionally, it is used in 3GPP networks in the IMS, in LTE for mobility management and for *Policy and Charging Control* PCC. Though we won't redraw our 3GPP application tree, 3GPP also uses Diameter for other interfaces. These include:

- ***Generic Authentication Architecture*** (GAA)
- 3GPP to ***Wireless LAN*** (WLAN) Interworking
- Location Services (LCS)
- EPS AAA Interfaces

However, the primary ones are used in IMS, LTE and PCC

1.7.8 LTE attachment



2 GSM | GPRS | VOIP | VOLTE | LTE threats attack

2.1 GSM threat attacks

2.1.1 Attacker's profile

An attacker can be a person or a group of people sufficiently qualified to build a node to emulate that of a mobile operator.

To access an SS7 network, attackers can acquire an existing provider's connection on the black (underground) market and obtain authorization to operate as a mobile carrier in countries with lax communications' laws.

In addition, any hacker who happens to work as a technical specialist at a telecommunications operator, would be able to connect their hacking equipment to the company's SS7 network.

In order to perform certain attacks, legitimate functions of the existing communication network equipment must be used.

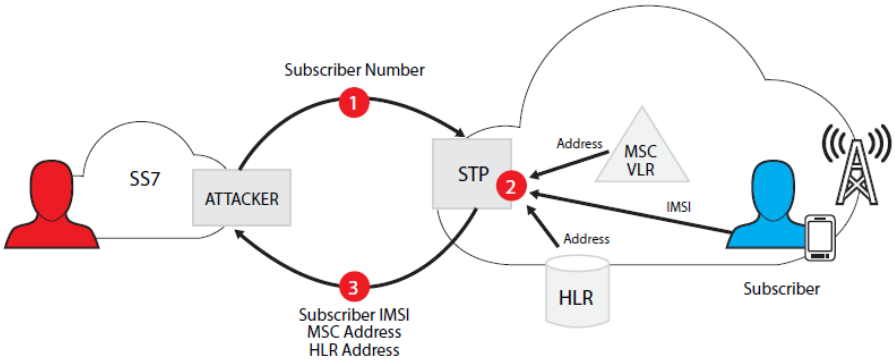
There is also an opportunity to penetrate a provider's network through a cracked edge device (GGSN or a femtocell).

Besides having different ways of accessing an SS7 network, attackers likely also have different motives for doing so including performing fraudulent activities, obtaining a subscriber's confidential data or disrupting service for certain subscribers or the whole network.

2.1.2 IMSI disclosure (Requesting MSC)

Goal: Analyze a service provider's network to obtain subscriber information.

- This attack is based on requesting the Mobile Switching Center (MSC) Visitor Location Register (VLR) address, and the IMSI.
- The request is part of the SMS delivery protocol, which allows the source network to receive information about the subscriber's location for further routing of the message.
- The initial data includes the target subscriber number



Result: In case of successful exploitation, an attacker obtains the following data:

+ Subscriber's IMSI + Servicing MSC/VLR address + Home Location Register (HLR) address where the subscriber's account data is located

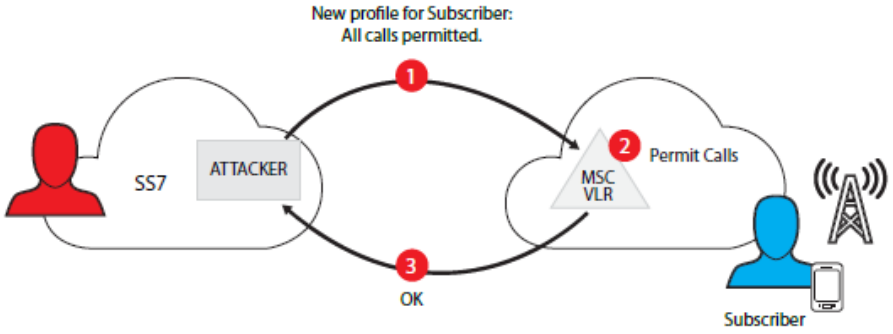
The MSC/VLR address will determine the subscriber's location down to the regional level. Moreover, the intruder can use the obtained data in more complex attacks (as described below).

2.1.3 Subscriber Profile Manipulation (Send fake subscriber profile to VLR)

Goal: Spoof the network with fake subscriber profile data

Description: When a subscriber registers on a switch, his/her profile is copied from the HLR database to the VLR database.

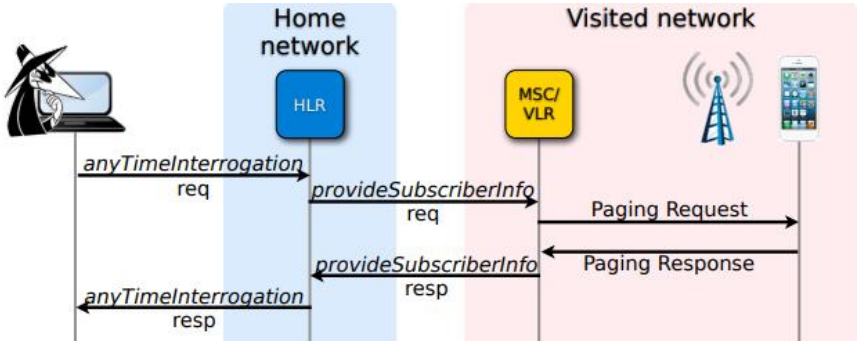
The profile contains information about active and inactive subscriber services, call forwarding parameters, the on-line billing platform address, etc. An attacker can send a fake subscriber profile to the VLR.



Result: A fake profile will fool the MSC/VLR into providing services to the subscriber based on altered and fraudulent parameters. For example, the subscriber will be able to make voice calls that bypass the billing system.

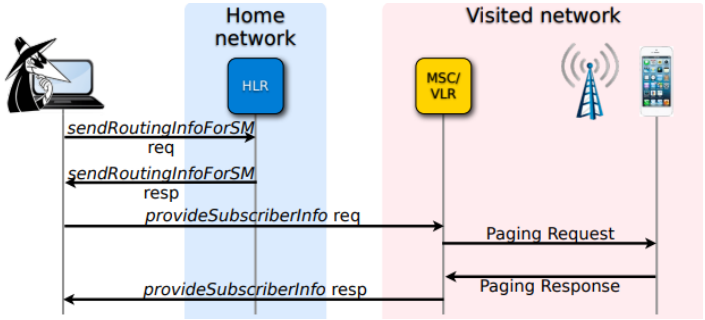
2.1.4 Cell Level Tracking using MAP's anyTimeInterrogation (ATI) service

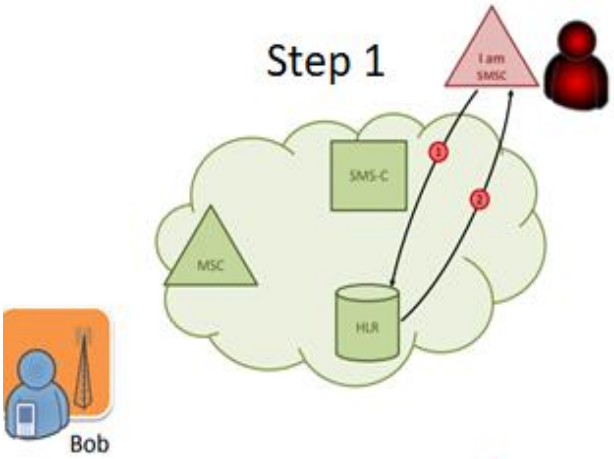
- MAP's anyTimeInterrogation (ATI) service can query the subscriber's HLR for her Cell-Id and IMEI (phone serial number, can be used to look up phone type)



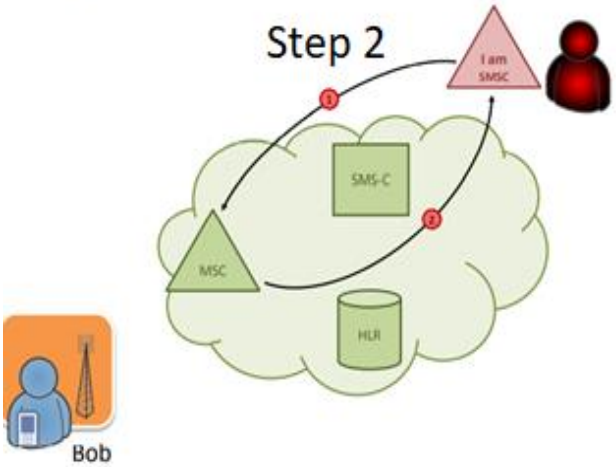
2.1.5 Cell Level Tracking using MAP's SendRoutingInfoForSM (Fake SMSC)

- Instead, query the MSC/VLR directly
- But MSC/VLR use IMSIs (International Mobile Subscriber Identifiers), not phone numbers, to identify subscribers
- ask the HLR for the subscriber's IMSI and Global Title of the current MSC/VLR
- When the attacker knows the IMSI of the subscriber and the Global Title, the MSC/VLR can be asked for the cell id of the subscriber





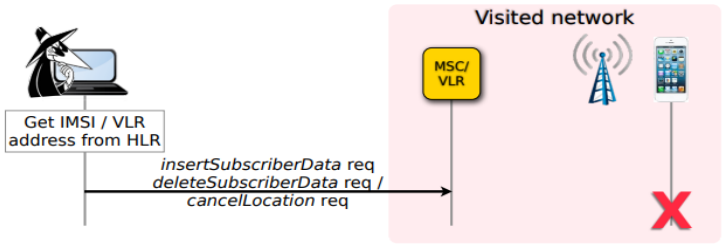
1. Attacker sends request SendRoutingInfoForSM addressing MAP message by MSISDN
2. HLR replies with:
   - own address
   - serving MSC address
   - IMSI

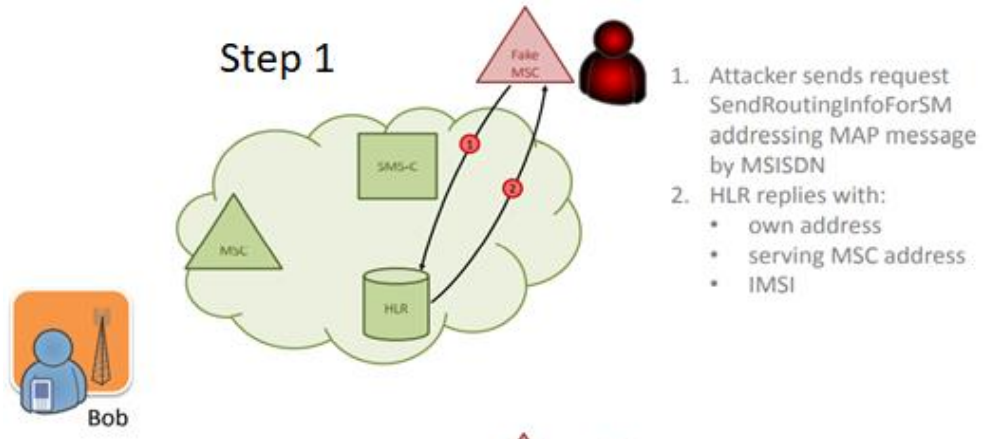


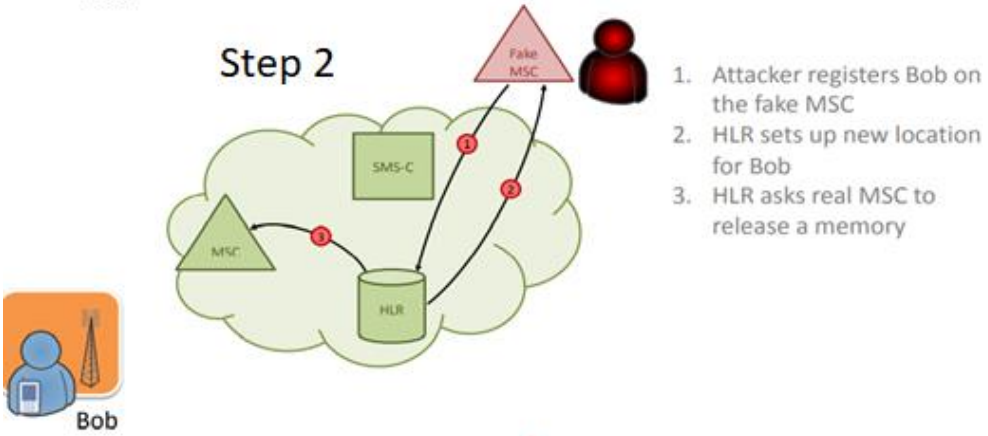
1. Attacker sends request provideSubscriberInfo addressing MAP message by IMSI and asking for subscriber location
2. MSC replies with Cell ID:
   - MCC - 250
   - MNC - 90
   - LAC 4A67
   - CID 673D

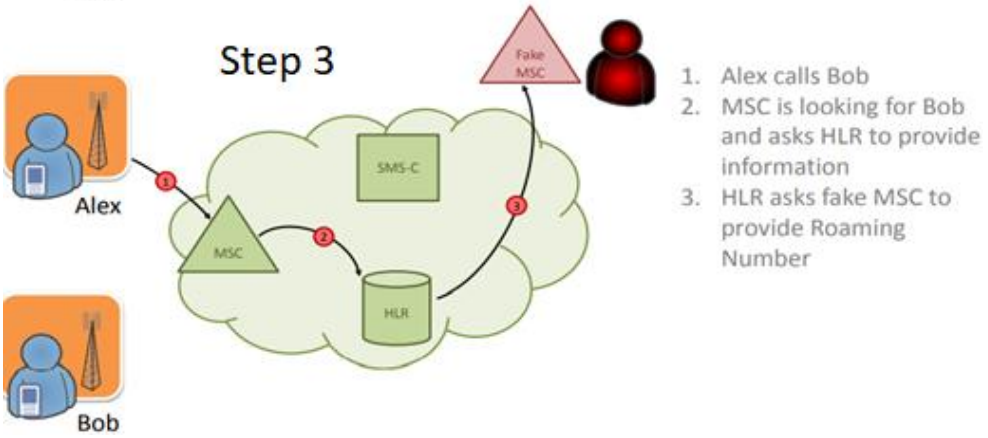
2.1.6 Denial of Service (Fake MSC)

- It is not only possible to read subscriber data - it can also be modified, since most network's VLR/MSC don't do any plausibility checks
- Control every aspect of what a subscriber is allowed to do: enable or disable incoming and/or outgoing calls / SMS or data or delete the subscriber from the VLR altogether









2.1.7 DOS call (using numerous roaming number requests)

Goal: Denial of service for incoming MSC calls

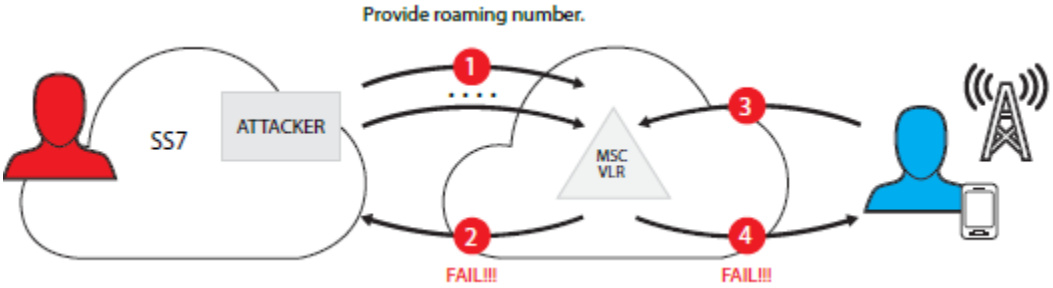
Description: This attack is based on the procedure of assigning a roaming number (MSRN) when receiving a voice call.

When a call is received, the current subscriber's MSC/VLR is identified, after which a voice channel is established to this switch using a temporary roaming number.

Normally, a roaming number lives for a split second. However, the default values of timers responsible for holding a roaming number, which are specified on the equipment, are 30—45 seconds.

If an attacker sends numerous roaming number requests, to a switch using default parameters, then the pool of available numbers will be used up quickly.

As a result, the switch will not be able to process incoming mobile calls.



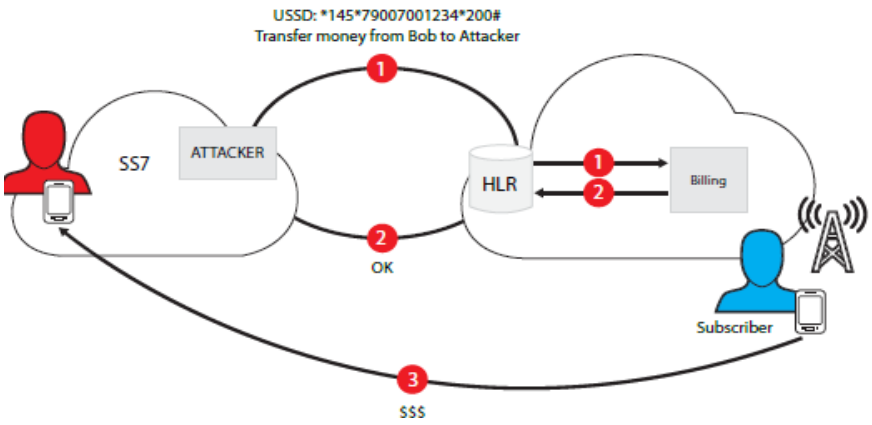
2.1.8 USSD Request Manipulation

Goal: Send USSD requests directly to HLR

Description: This attack is a good example of using a legitimate message with a USSD request sent from VLR to HLR. The initial data is the target subscriber number, the HLR address and the USSD string.

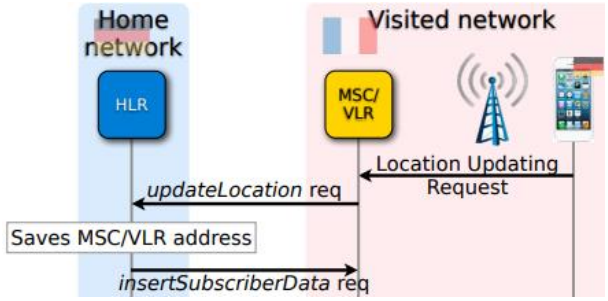
The subscriber number is usually known from the beginning.

The HLR address can be obtained as outlined in 4.1 and USSD requests are described on the service provider's site.

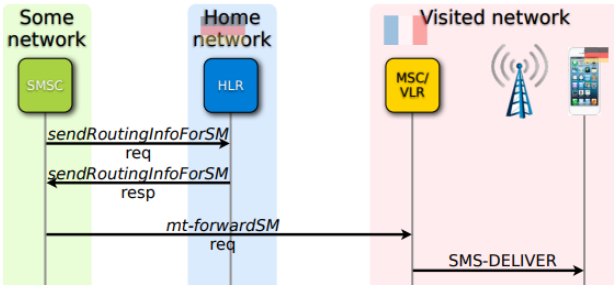


2.1.9 HLR Stealing Subscribers (Roaming scenario)

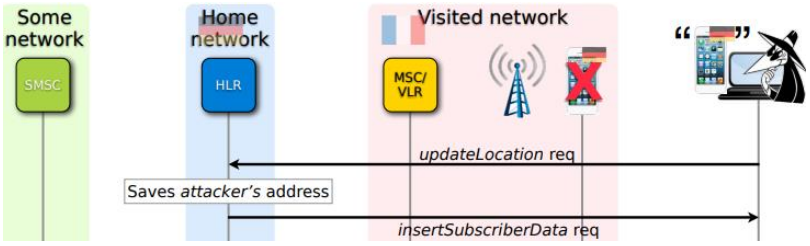
- When a subscriber travels to another region or country, the VLR/MSC sends a MAP updateLocation request to the subscriber's HLR
- The HLR sends a copy of the subscriber's data to the VLR/MSC and saves the address of the VLR/MSC



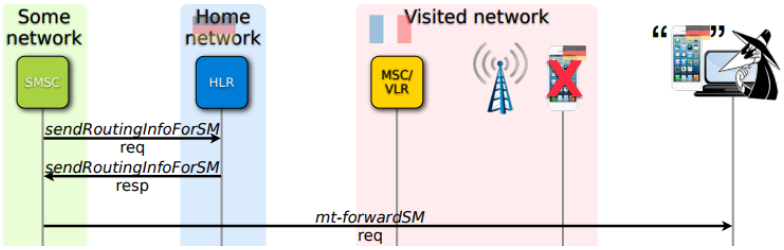
- Now, when somebody wants to call or text the subscriber, the HLR gets asked for routing information (sendRoutingInfo...) and hands out the address of the VLR/MSC



- The updateLocation procedure is also not authenticated
- An attacker can simply pretend that a subscriber is in his "network" by sending the updateLocation with his Global Title to the subscriber's HLR

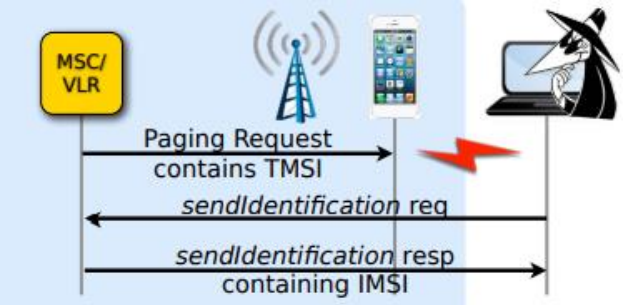


- Now, calls and SMS for that subscriber are routed to the attacker
- Example: Subscriber's bank sends text with mTAN. Attacker intercepts message and transfers money to his own account



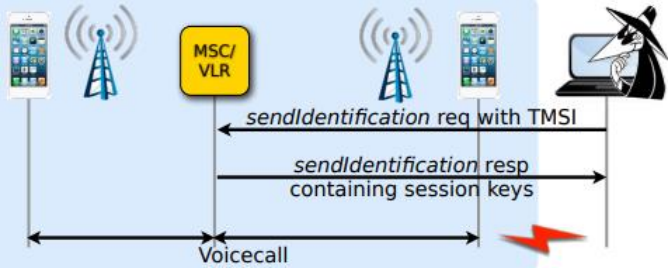
2.1.10 Hybrid Attacks: TMSI De-anonymization

- An attacker can find out the phone numbers of subscribers around him:
  - ▸ Paging of subscribers (e.g. to notify them of an incoming call) has to happen unencrypted
  - ▸ TMSI (Temporary Mobile Subscriber Identifier) is normally used for paging so that the real identity of the subscriber (IMSI) does not have to be sent over the air unencrypted
- Attacker captures TMSI over the air, e.g. with OsmocomBB
- The MSC can be asked to hand out the IMSI if the TMSI is known
- With updateLocation, the attacker can figure out the MSISDN belonging to the IMSI



2.1.10.1 Hybrid Attacks: Intercept Calls

- The MSC can be also be asked for the session key for the subscriber!
- If the attacker captures an encrypted GSM or UMTS call, he can then decrypt it using the session key
- Passive attack, no IMSI catcher necessary



2.1.11 Intercepting outgoing calls

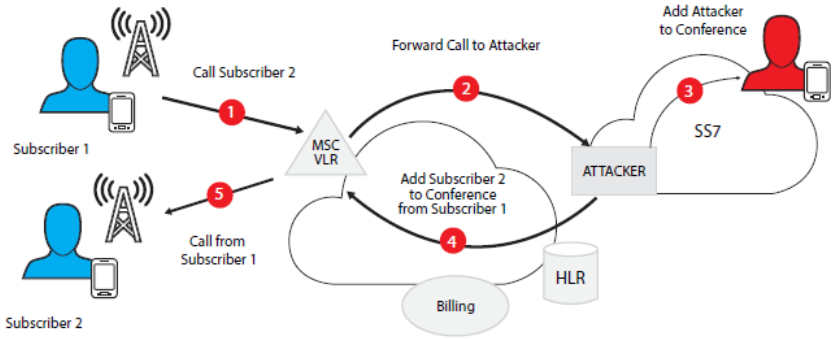
Goal. Redirecting outgoing subscriber voice calls and data messages to an attacker's device.

Description. This attack is an extension of Subscriber Profile Manipulation in VLR attack, described in **1.7 section.**

An attacker substitutes a billing platform address with their equipment address, in the subscriber's profile.

When the subscriber makes a call, the billing request along with the number of the destination subscriber are sent to the attacker's equipment.

The attacker can then redirect the call and create a three-way (destination subscriber, calling subscriber and an attacker) conference call.



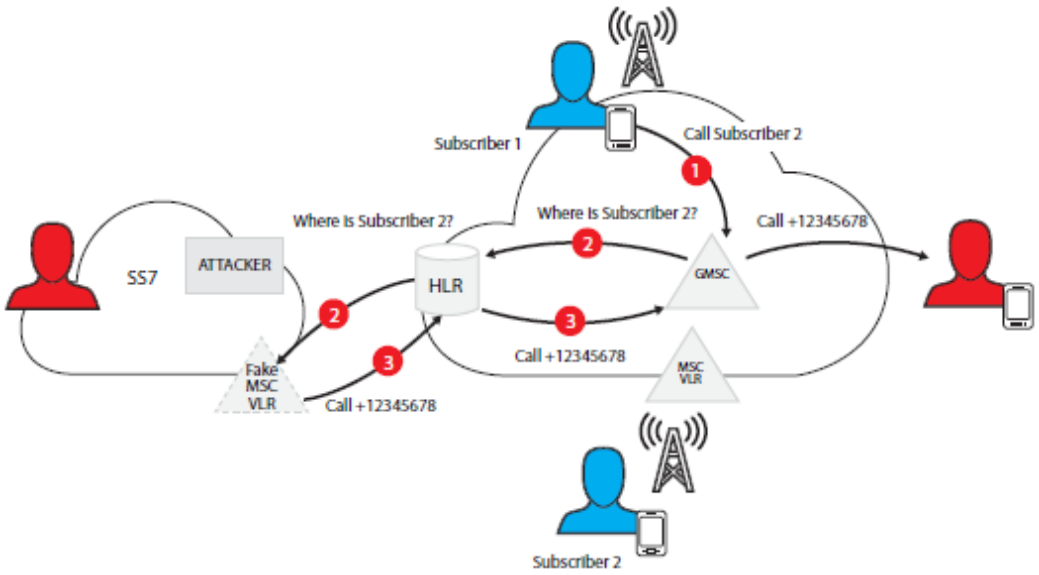
2.1.12 Redirecting incoming calls

Goal: Change voice call routing and redirect incoming calls

Description: This attack is for incoming calls and is an extension of the attack described in **section 1.10**.

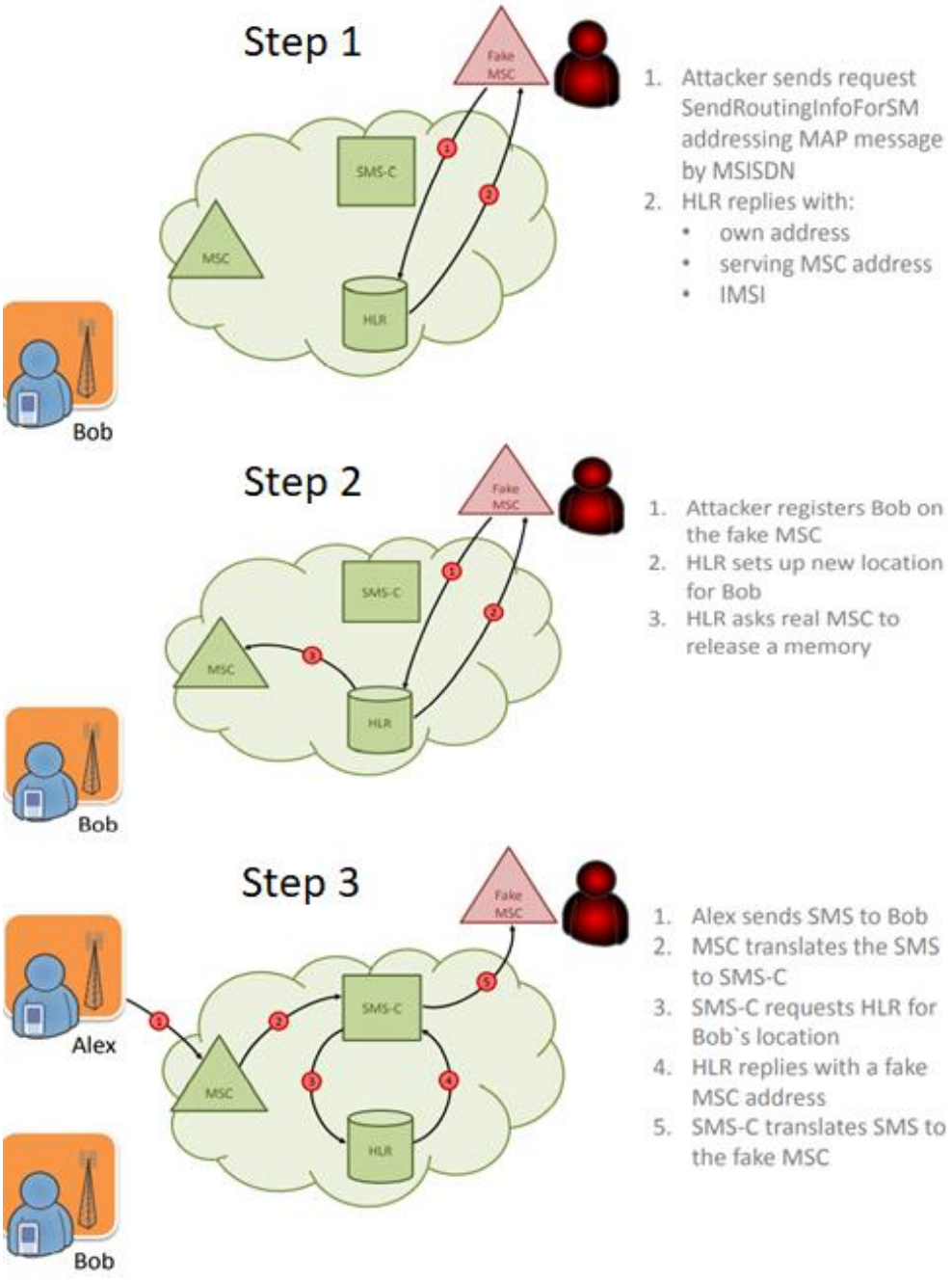
When a call is terminated, the gateway MSC (GMSC) sends a request to the HLR to identify the MSC/VLR that currently serves the subscriber. This data is necessary to route the call to the appropriate switch.

After successfully performing the attack in **section 1.10**, the HLR will redirect the received request to a fake MSC/VLR, which in turn will send the Mobile Station Roaming Number (MSRN) to redirect the call. The HLR transfers this number to the GMSC, which redirects the call to the provided MSRN.



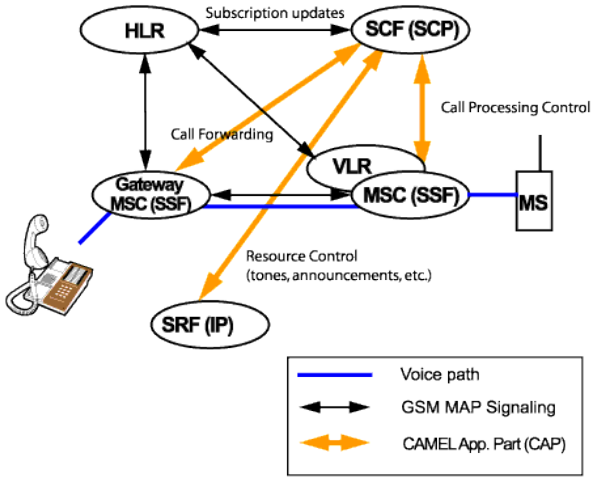
2.1.13 SMS intercept (using Fake MSC)

- A virus on a smartphone – and what if a certain subscriber is a target? How to infect him particularly?
- Reissue SIM? It works only once.
- Radio signal interception (GSM A5/1)? You need to be nearby.
- Via SS7 network is a solution

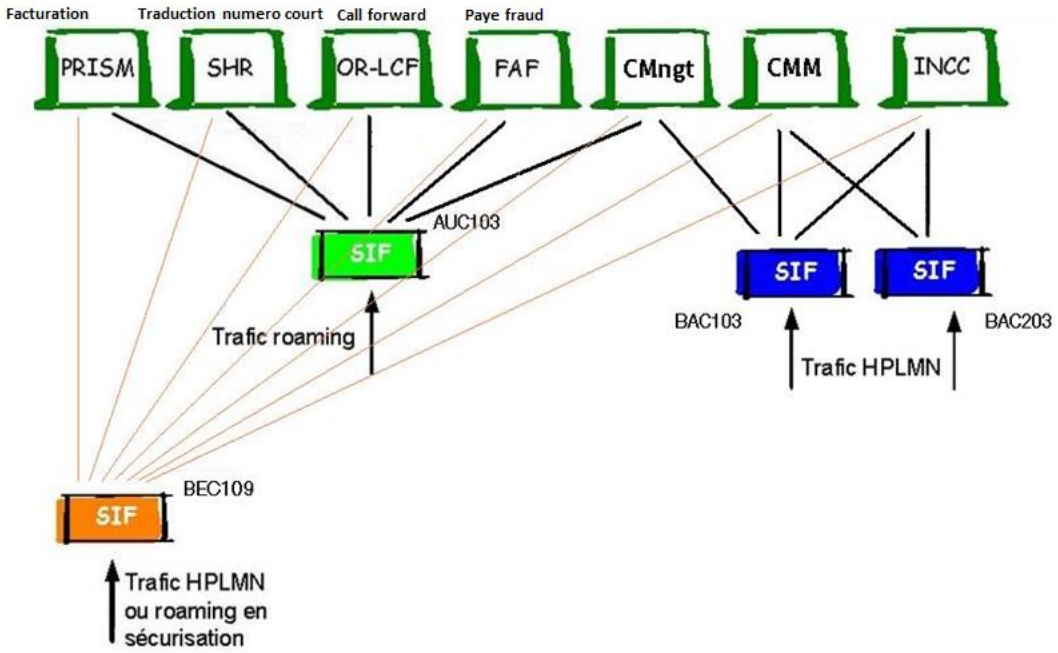


2.1.14 Intercepting calls with CAMEL (Roaming scenario)

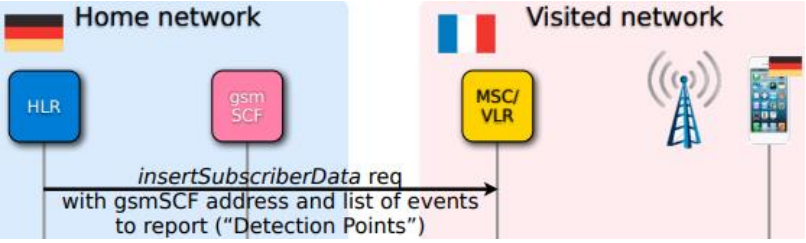
Just for your information CAMEL is a means of adding intelligent applications to mobile (rather than fixed) networks. This is the architecture



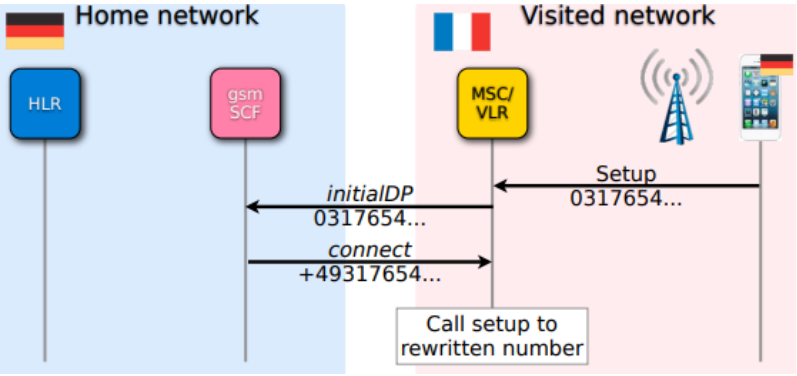
And these are the service offered by CAMEL



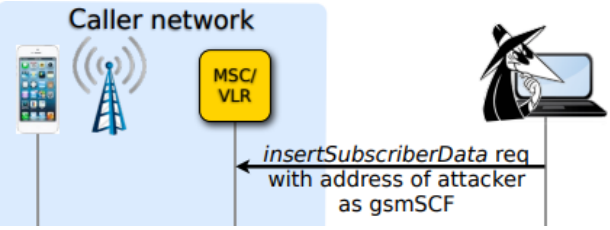
- This how the threat attack come in CAMEL technology:
- German HLR tells French VLR “notify my gsmSCF at address +4917... whenever the subscriber wants to make a call”



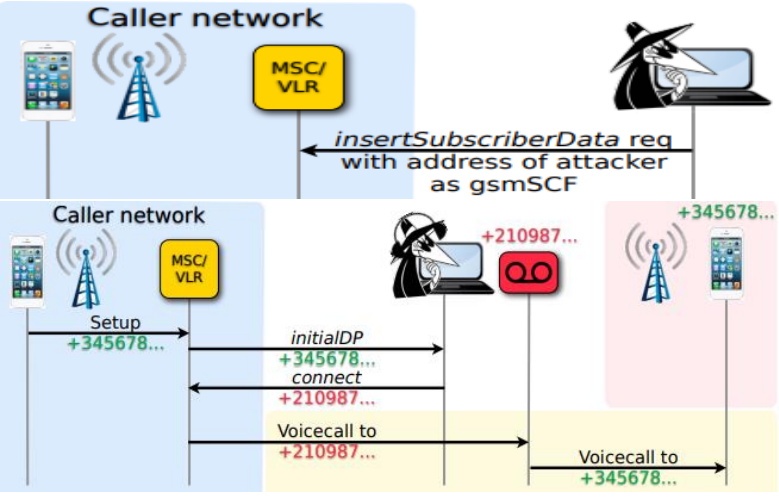
- Subscriber wants to make a phone call, but dials number in German national format (0317654...)
- MSC asks gsmSCF in home network what to do with the call
- gsmSCF rewrites number to international format (+49317654...) and tells MSC to continue with the new number



- Attacker overwrites gsmSCF address in subscriber’s MSC/VLR with its own, “fake gsmSCF” address



- Subscriber wants to call +345678..., but the MSC now contacts the attacker instead of the subscriber’s gsmSCF
- Attacker rewrites number to +210987..., his recording proxy (e.g. an Asterisk PBX)
- MSC sets up call to +210987..., which bridges it to the original +345678...
- Both subscribers can talk to each other, while the attacker records the conversation



2.1.15 SPAM Message in mobile network

Spam refers to unsolicited calls or messages sent in bulk to a telephone line. Spam can be either of a commercial nature or of a fraudulent nature. Fraudulent spam generally falls under one of the following two practices:

- Scam the purpose of which is to extract money from the victim;
- Theft of personal data, the purpose of which is to obtain sensitive information such as credit card numbers or usernames and passwords for connecting to a website.

For this, fraudulent spam responds to several operating modes:

- **Spam SMS**:

- sending fraudulent SMS, which encourages calling a premium rate number, generally 0899,
- sending an SMS to a premium rate number, generally 5 characters,
- Or clicking on a link on a page Internet.

The messages received have a familiar and encouraging nature, such as:

- "Hi, it's me, you didn't call me. I'm waiting for your call back on 0899 (...).
- "Or" Hello, a package has been waiting for you for 10 days and will leave if you don't pick it up by tomorrow. Please call us on 0899 (...).

- **Voice spam**:

- Emission of calls broadcasting a pre-recorded message in order to encourage the called party to call back a surcharged number in 089.

  The messages are familiar, encouraging but can sometimes be very anxiety-provoking such as:
  "Hello, these are the emergencies of XYZ Hospital. Your partner has just had a serious accident. Please call us on 0899 (…). ";

- **Ping call**:

transmission of short calls (one or two maximum rings) without giving the recipient time to pick up the receiver in the hope that the latter will call back the number presented without paying attention or out of curiosity.

While “ping calls” historically called back surcharged “089” numbers directly, this practice has evolved since Arcep banned the use of 089 numbers as a caller ID in 2012 (decision no. -0856).

- **Spoofing**

There were nearly 26 billion scam calls in 2019, according to data collected by YouMail, and scammers are getting smarter. Now they are using a technique called spoofing to make it easier to scam you.

Spoofing is when someone makes your phone number pop up on a caller ID when it really isn’t you that’s making the call.

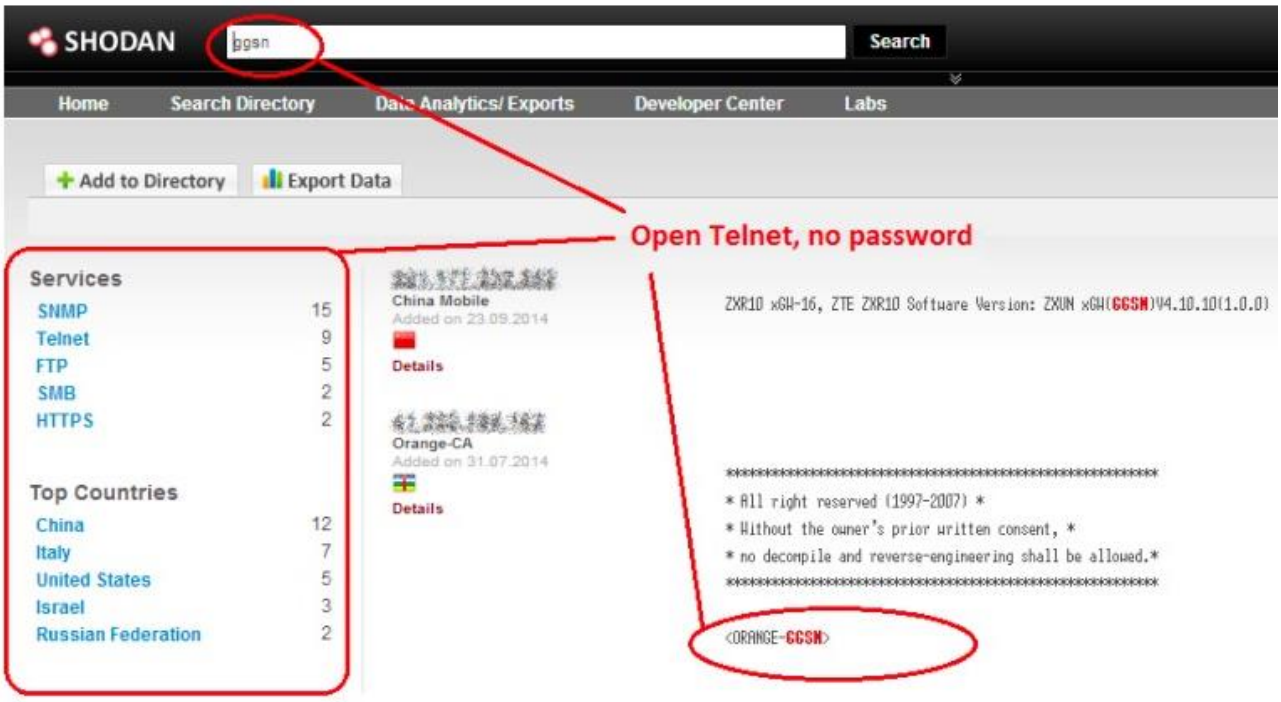
For example, a scammer once spoofed my daughter’s phone number to make me think she was calling me. The goal was to trick me into answering the phone. It worked, because what if it was an emergency and my daughter needed me? When a scammer gets you to pick up, they have the chance to trick you into whatever scheme they’ve come up with, like tricking you into giving them your credit card information.

It doesn’t take much to spoof a phone number. There are apps and websites that allow scammers to simply type in a phone number and make a call. It’s super easy and quick, which makes it appealing to scammers.

2.2 GPRS threats attack

2.2.1 Searching for mobile operator's facilities on the Internet

- We already know that GGSN must be deployed as an edge device. Using Shodan.io search engine for Internet-connected devices, we can find the required devices by their banners.



Search result displays about 40 devices using this abbreviation in their banners.

The screenshot provides a list of some devices that use this abbreviation, including devices with open Telnet and turned off password authentication.

- An attacker can perform an intrusion into the network of the operator in the Central African Republic by connecting to this device and implementing the required settings.

- Having access to the network of any operator, the attacker will automatically get access to the GRX network and other operators of mobile services. One single mistake made by one single operator in the world creates this opportunity for attack to many other mobile networks.

- There are more ways of using the compromised boundary host, for example, DNS spoofing attack (more information about attacks is considered below).

2.2.2 IMSI brute force

Goal: To find a valid IMSI.

Attack vector: An attacker conducts attacks from the GRX network or the operator's network.

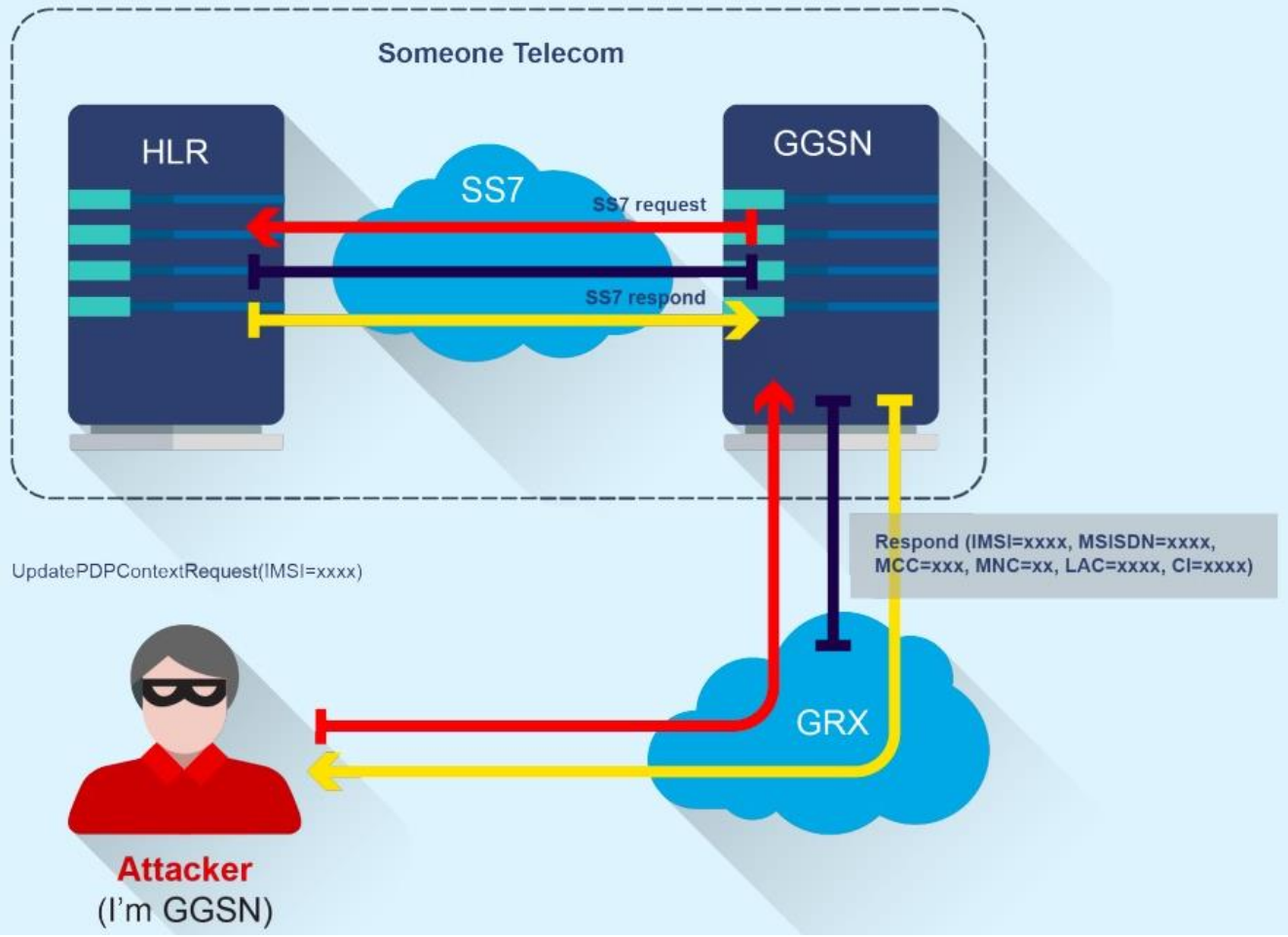
Description: IMSI is the SIM card Number (International Mobile Subscriber ID). It consists of 15 digits, the first three identify the Mobile Country Code (MCC), and the next two digits are the Mobile Network Code (MNC).

- You can choose the required operator on the website www.mcc-mnc.com, enter the MCC and MNC
- And then brute force the remaining 10 digits by sending a «Send Routing Information for GPRS Request» message via GRX.

This message can be sent to any GSN device, which converts the request into an SS7 format (CS core network component) and sends it to HLR where it is processed by SS7 network.

If the subscriber with this IMSI uses the Internet, we can get the SGSN IP address serving the mentioned subscriber. Otherwise, response will be as follows: «Mobile station Not Reachable for GPRS».

Result: Obtaining a list of valid IMSI for further attacks.



2.2.3 The disclosure of subscriber's data via IMSI

Goal: To obtain a phone number, location data, information about the model of a subscriber's mobile device via IMSI.

Attack vector: An attacker conducts attacks from the GRX network or the operator's network.

Description: An attacker can use this vulnerability after the success of the previous attack or if he/she gets a subscriber's IMSI via a viral application for the subscriber's smartphone.

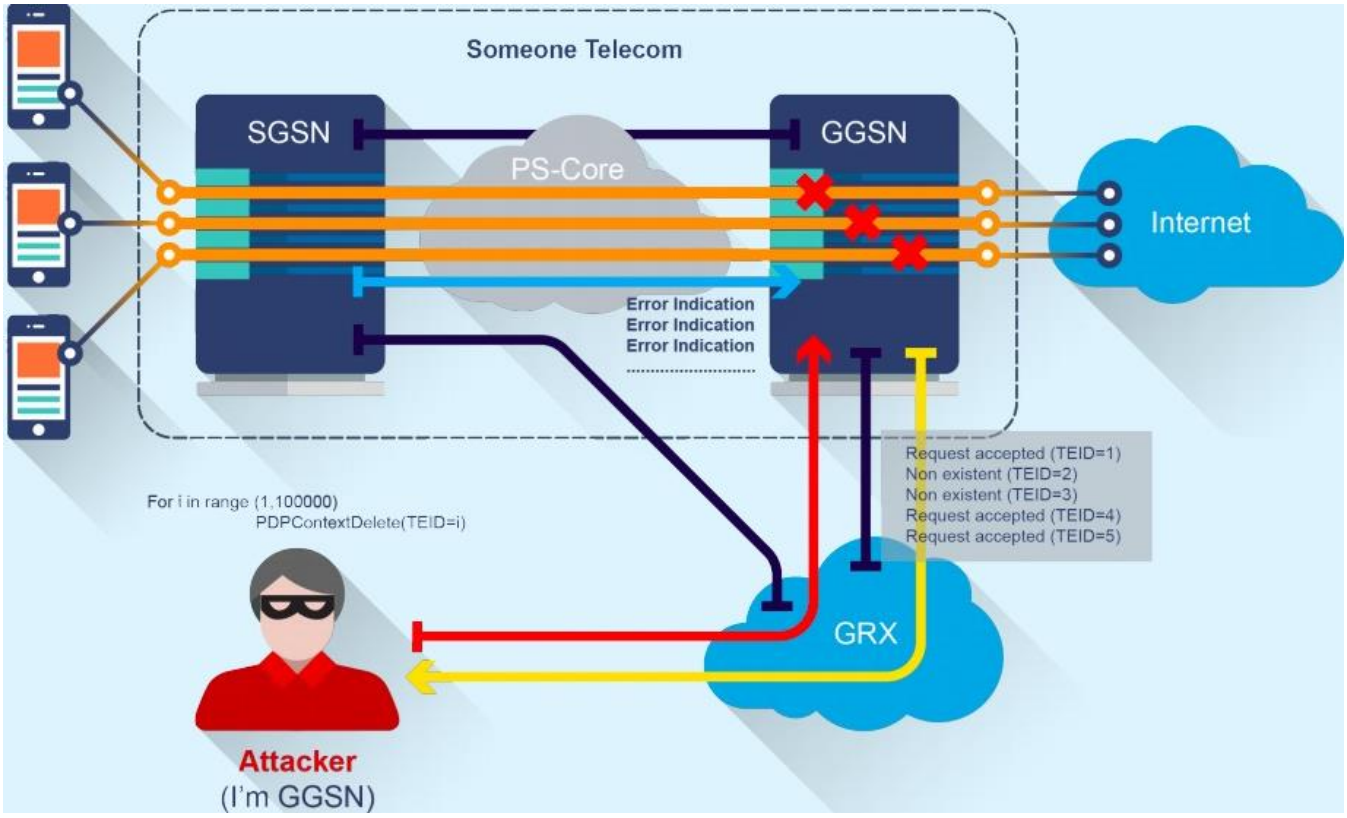
The attacker needs to know the SGSN IP address, garnered from the previous attack.

After that, the attacker sends an Update PDP Context Request to the SGSN IP address requesting the subscriber's location; the GSN Control Plane is spoofed with the attacker's IP address.

The response contains MSISDN (Mobile Subscriber Integrated Services Digital Number), IMEI (International Mobile Equipment Identity, it helps to identify the model of a subscriber's phone) and the current subscriber's mobile radio base tower (MCC, MNC, LAC, CI).

Consequently, the attacker can find the subscriber's location accurate to several hundred meters using the following website: https://xinit.ru/bs/ or http://opencellid.org/.

Result: The required information about the subscriber is obtained.



2.2.4 Disconnection of authorized subscribers from the Internet

Goal: To disconnect the connected subscribers.

Attack vector: An attacker conducts attacks from the GRX network or the operator's network.

Description: The attack is based on sending the «PDP context delete request» packets to the target GGSN with all the TEID listed.

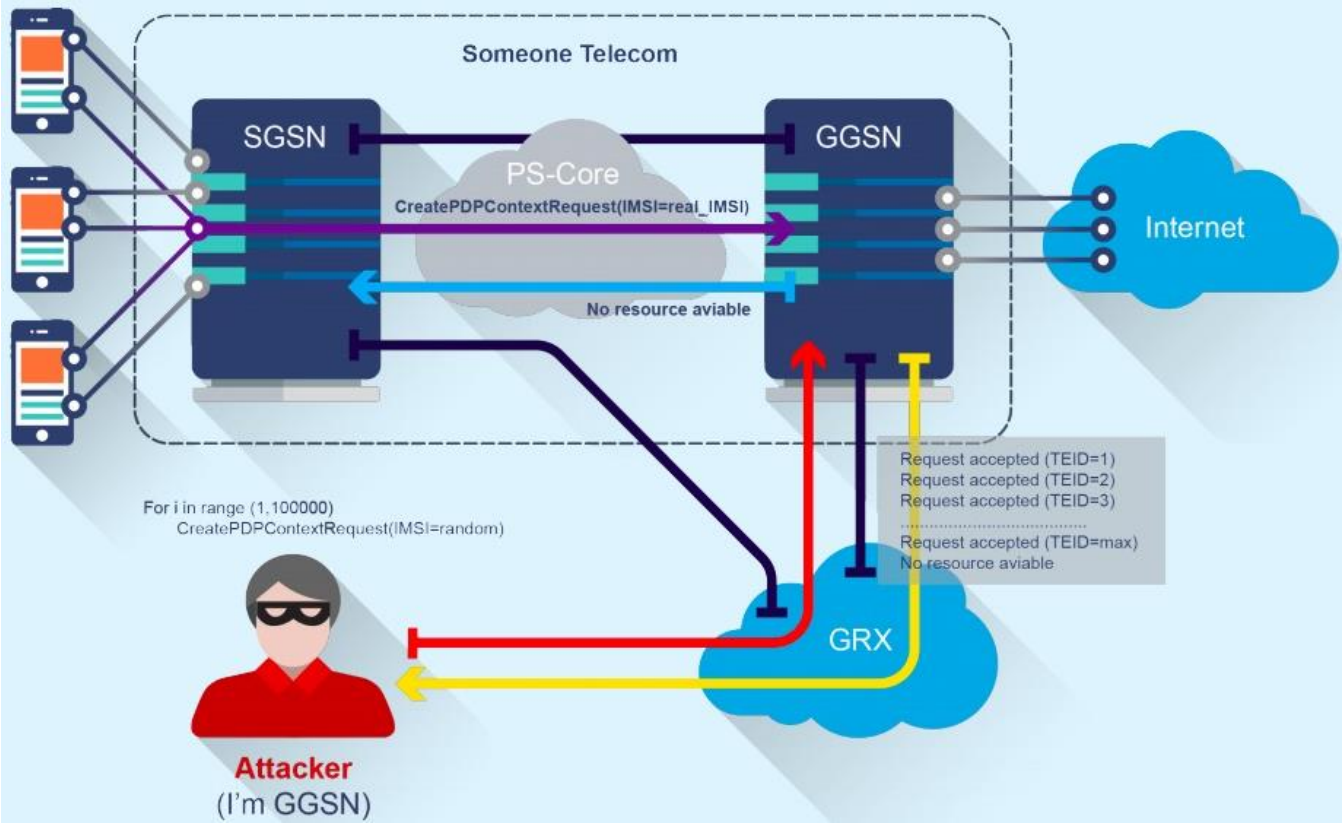
The PDP Context information is deleted, which causes disconnection of authorized subscribers.

At the same time, GGSN unilaterally closes tunnels and sends the responses on this event to the attacker.

A valid SGSN used by the subscriber to set up the connection doesn't have information about closing connections, so tunnels continue to occupy the hardware resources.

The subscriber's Internet stops working, but the connection is displayed as active.

Result: All subscribers connected to this GGSN will be disconnected. The amount of subscribers served by one GGSN is 100,000— 10,000,000.



2.2.5 Blocking the connection to the Internet

Goal: To block the establishment of new connections to the Internet.

Attack vector: An attacker conducts attacks from the GRX network or the operator's network.

Description: The attack is based on sending the «Create PDP context request» packets with IMSI list, thus the exhaustion of the available pool of PDP tunnels occurs.

For example, the maximum number of PDP Context Cisco 7200 with 256 MB of memory is 80,000, with 512 MB — 135,000: it is not difficult to brute force all possible combinations.

Moreover, more and more IP addresses from DHCP pool are issued and they may be exhausted.

It does not matter what will be exhausted first — the DHCP pool or the PDP pool,
After all, GGSN will response with «No resource available» to all valid connection requests.

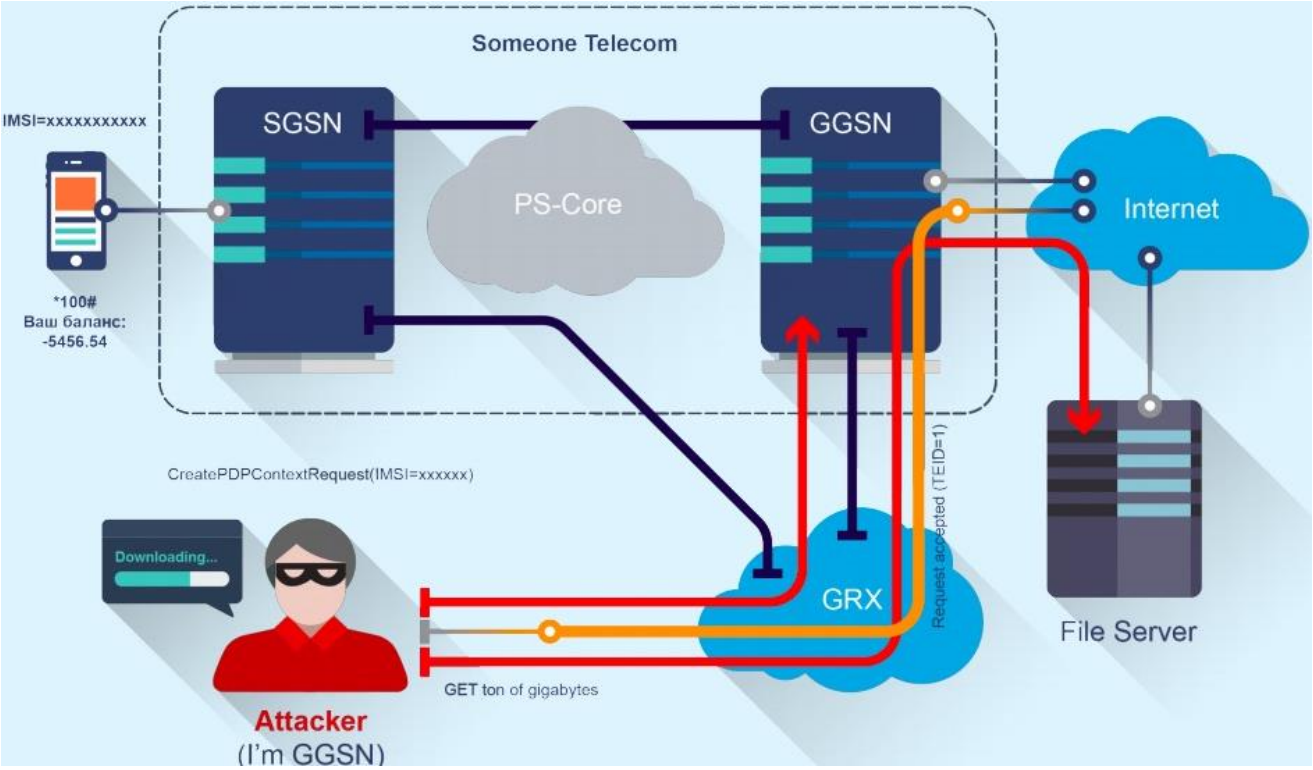
Moreover, GGSN cannot close tunnels, because when you try to close one, GGSN sends an attacker «Delete PDP context request» with the number of the tunnel to be closed.

If there is no response (actually, there isn't any response because an attacker does not want this to happen), GGSN sends such requests over and over again. The resources remain occupied.

In case of successful implementation of this attack, authorized subscribers will not be able to connect to the Internet and those who were connected will be disconnected as GGSN sends these tunnels to the attacker's address.

This attack is an analogue of the DHCP starvation attack at the GTP level.

Result: The subscribers of the attacked GGSN will not be able to connect to the Internet. The amount of subscribers served by one GGSN is 100,000–10,000,000.



2.2.6 Internet at the expense of others

Goal: The exhaustion of the subscriber's account and use of the connection for illegal purposes.

Attack vector: An attacker conducts attacks from the GRX network or the operator's network.

Description: The attack is based on sending the «Create PDP context request» packets with the IMSI of a subscriber known in advance.

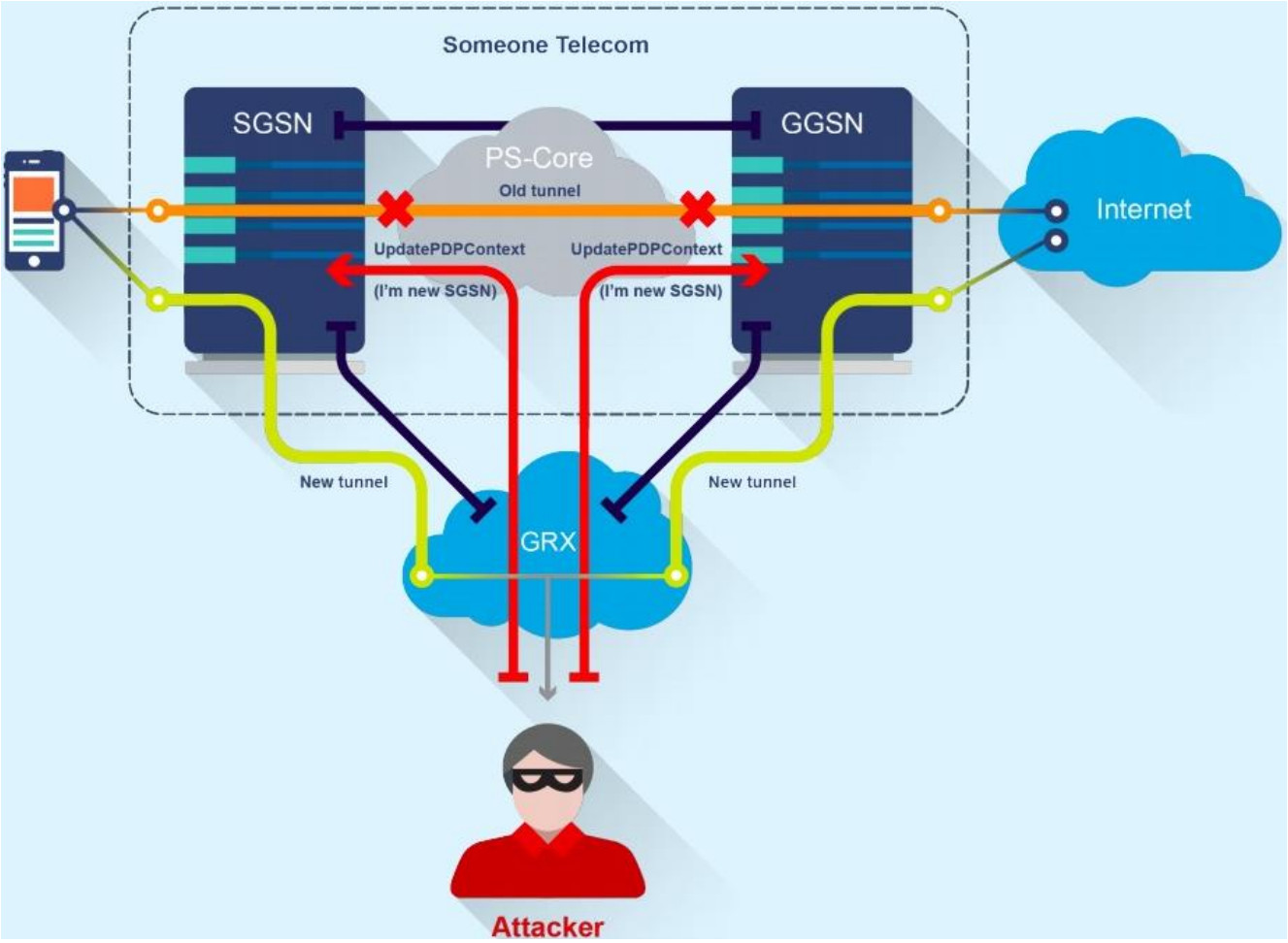
Thus, the subscriber's credentials are used to establish connection.

Unsuspecting subscriber will get a huge bill.

It is possible to establish connection via the IMSI of a non-existent subscriber, as subscriber authorization is performed at the stage of connecting to SGSN and GGSN receives already verified connections.

Since the SGSN is compromised, no verification is carried out.

Result: An attacker can connect to the Internet with the credentials of a legitimate user.



2.2.7 Data interception (Using a spoofed GSN addresses to SGSN and GGSN

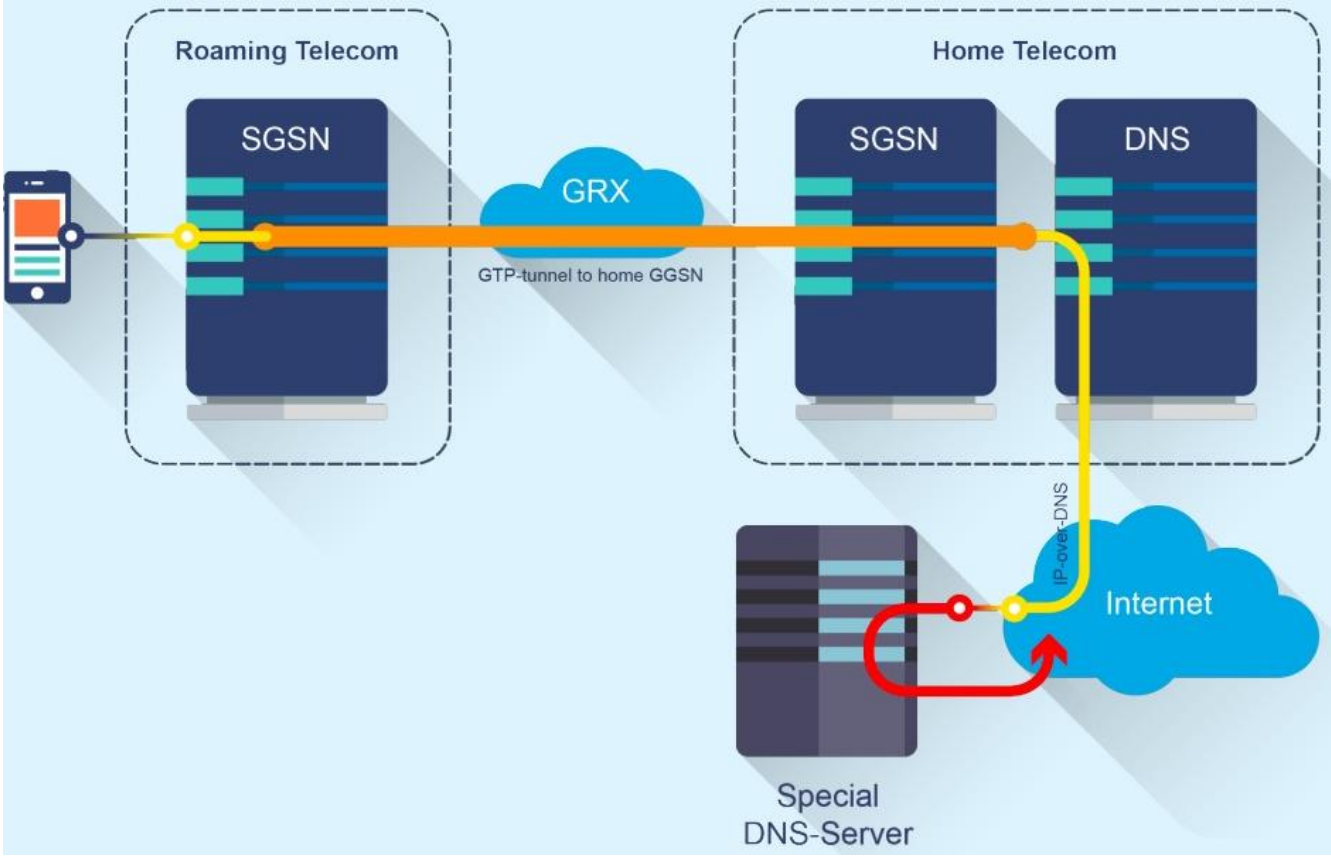
Goal: To listen to the traffic of the victim and conduct a fishing attack.

Attack vector: An attacker conducts attacks from the GRX network or the operator's network.

Description: An attacker can intercept data sent between the subscriber's device and the Internet by sending an «Update PDP Context Request» message with spoofed GSN addresses to SGSN and GGSN.

This attack is an analogue of the ARP Spoofing attack at the GTP level.

Result: Listening to traffic or spoofing traffic from the victim and disclosure of sensitive data.



2.2.8 DNS tunneling

Goal: To get non-paid access to the Internet from the subscriber's mobile station.

Attack vector: The attacker is the subscriber of a mobile phone network and acts through a mobile phone.

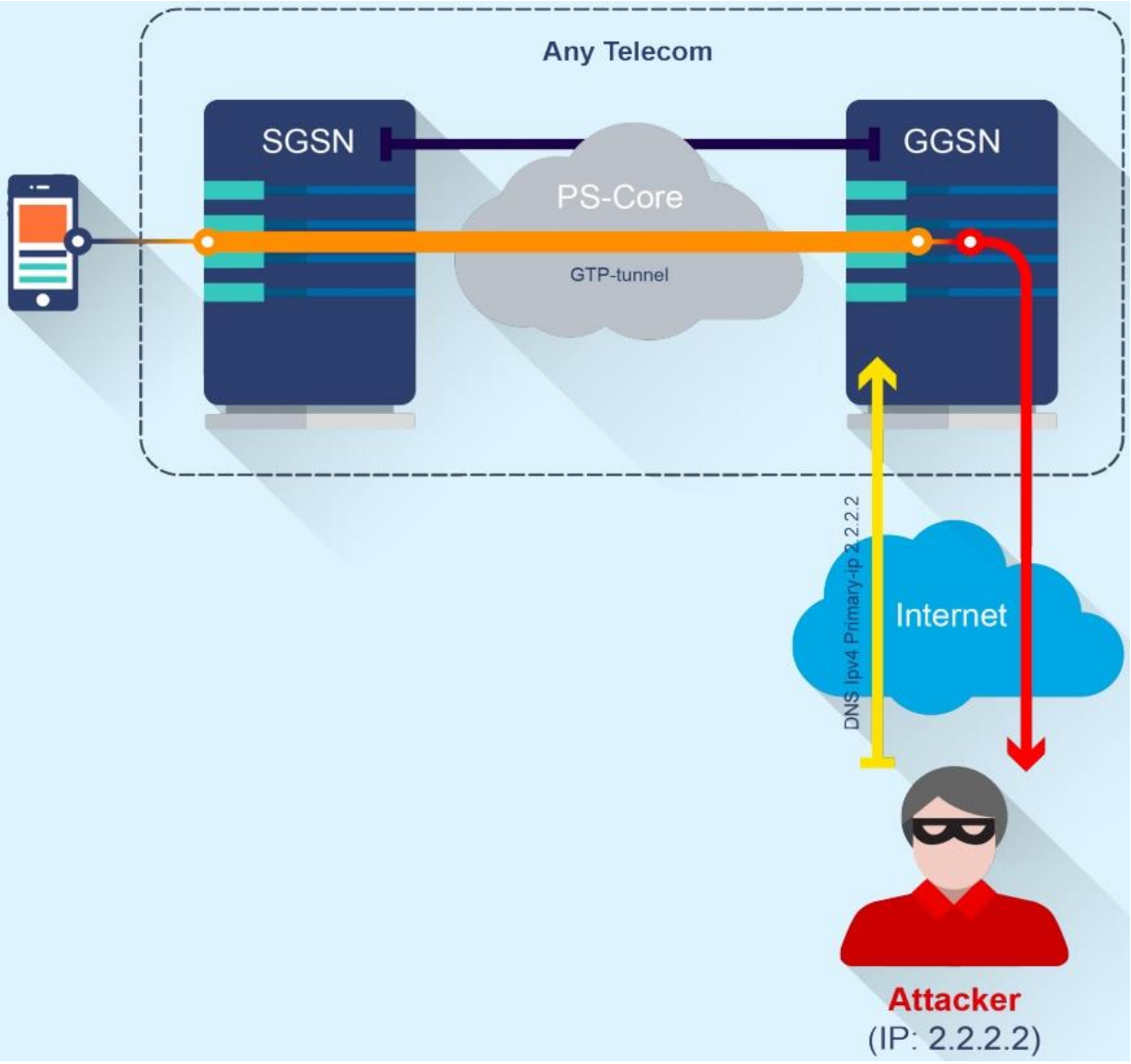
Description: This is a well-known attack vector, rooted in the days of dial-up, but the implementation of low-price and fast dedicated Internet access made it less viable.

However, this attack can be used in mobile networks, for example, in roaming when prices for mobile Internet are unreasonably high and the data transfer speed is not that important (for example, for checking email).

The point of this attack is that some operators do not rate DNS traffic, usually in order to redirect the subscriber to the operator's webpage for charging the balance.

An attacker can use this vulnerability by sending special crafted requests to the DNS server; to get access one needs a specialized host on the Internet.

Result: Getting non-paid access to the Internet at the expense of mobile operator.



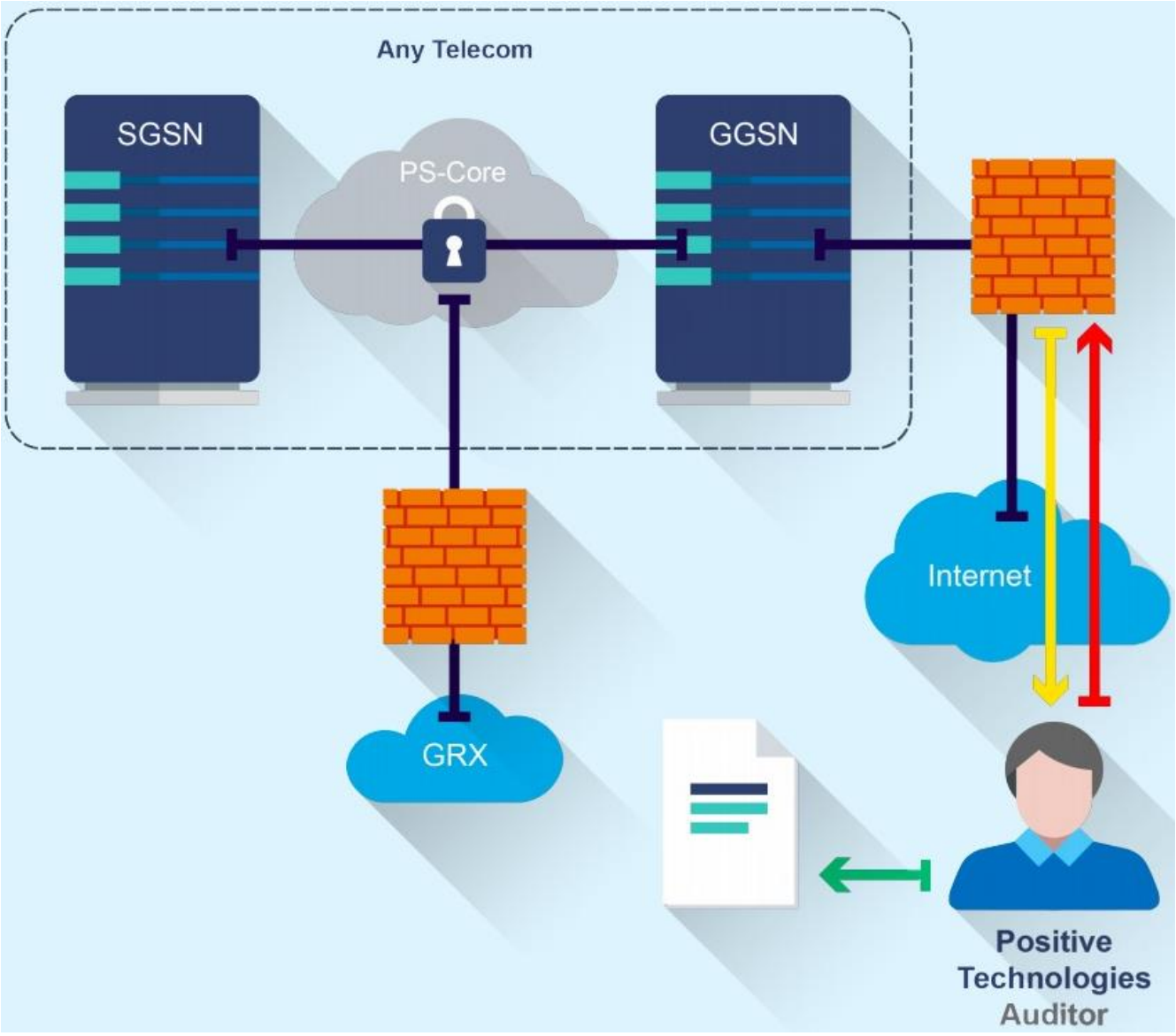
2.2.9 Substitution of DNS for GGSN

Goal: To listen to the traffic of the victim, to conduct a fishing attack.

Attack vector: An attacker acts through the Internet.

Description: If an attacker gets access to GGSN (which is quite possible as we could see), the DNS address can be spoofed with the attacker's address and all the subscriber's traffic will be redirected through the attacker's host. Thus, listening to all the mobile traffic of the subscriber is possible.

Result: An ability to listen to traffic or spoof traffic from all subscribers and then gather confidential data to engage it in fishing attacks.



2.3 VOIP Threats attack

- To perform VoIP information gathering, we need to collect as much useful information as possible about the target. As a start, you can do a simple search online. For example, job announcements could be a valuable source of information. For example, the following job description gives the attacker an idea about the VoIP:

  Job Description

  Position: SBC Voice Engineer / Architect (Sonus SBC 1000)
  Location: Remote
  Job Status: Project Based

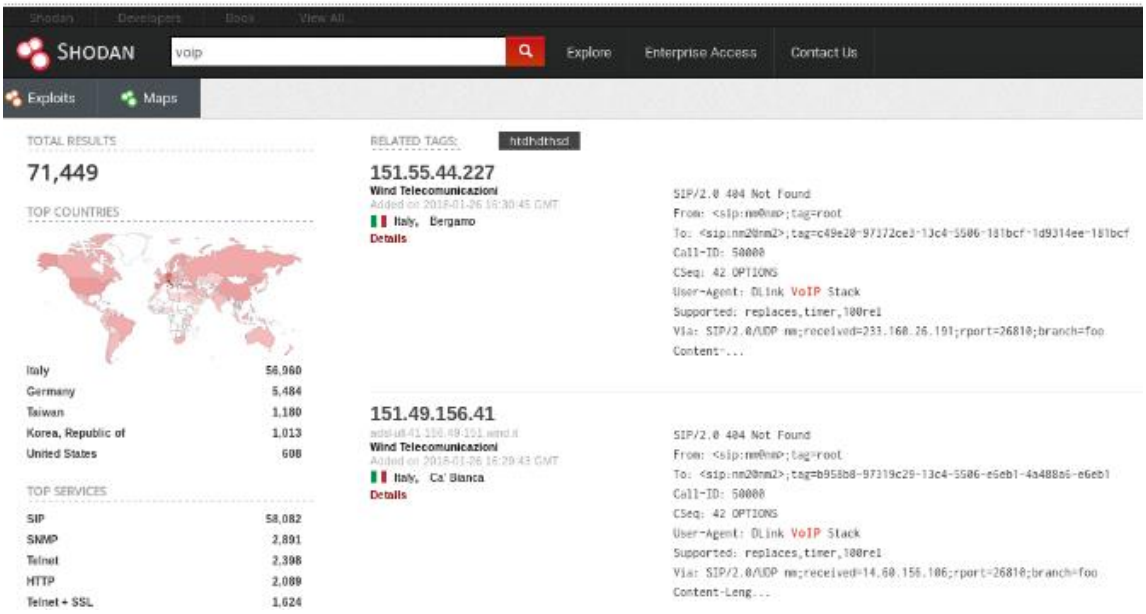
  Later, an attacker could search for vulnerabilities out there to try exploiting that particular system. Searching for phone numbers could also be a smart move, to have an idea of the target based on its voicemail, because each vendor has a default one. If the administrator has not changed it, listening to the voicemail can let you know about your target. If you want to have a look at some of the default voicemails, check http://www.hackingvoip.com/voicemail.html. It is a great resource for learning a great deal about hacking VoIP.

- Google hacking is an amazing technique for searching for information and online portals. We discussed Google hacking using Dorks. The following demonstration is the output of this Google Dork—in URL: Network Configuration Cisco:

  Network Configuration - Cisco Systems, Inc.
  222.249.148.238/NetworkConfiguration
  DHCP Server, 255.255.255.255. BOOTP Server, No. MAC Address, 0012008FA2DB. Host Name, SEP0012008FA2DB. Domain Name. IP Address, 222.249.148.238. Subnet Mask, 255.255.255.128. TFTP Server 1, 211.153.8.90. Default Router 1, 222.249.148.254. Default Router 2. Default Router 3. Default Router 4.

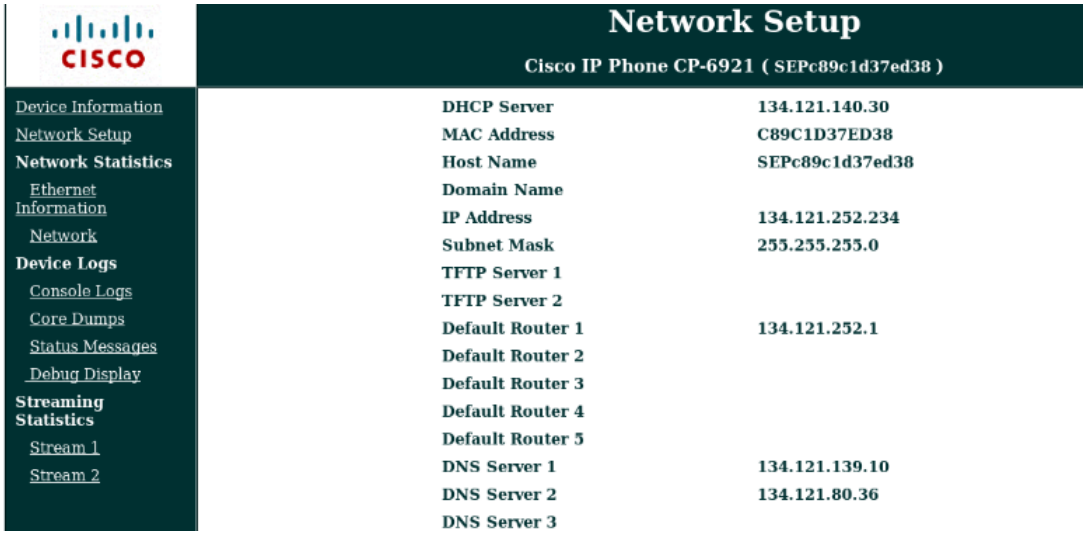
  b

- You can find connected VoIP devices using the Shodan.io search engine:



- VoIP devices are generally connected to the internet. Thus, they can be reached by an outsider. They can be exposed via their web interfaces; that is why, sometimes leaving installation files exposed could be dangerous, because using a search engine can lead to indexing the portal. The following screenshot is taken from an online Asterisk management portal:

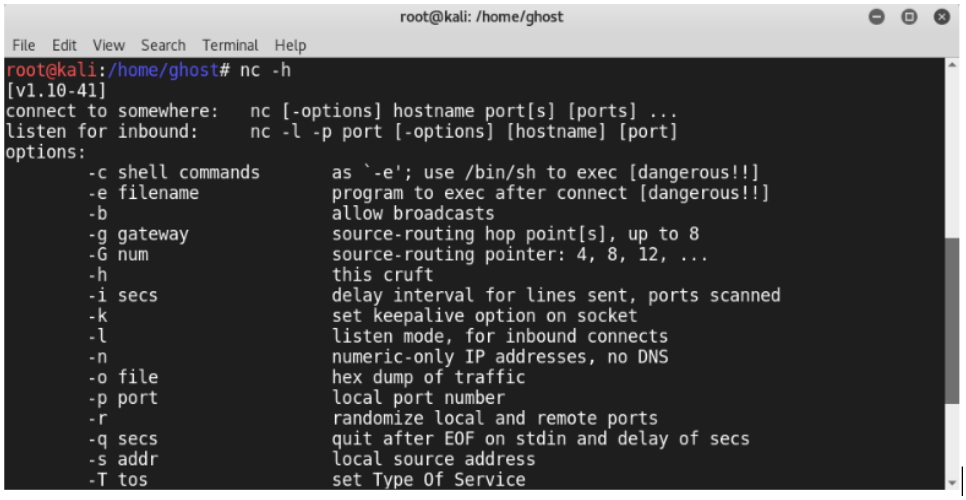
And this screenshot is taken from a configuration page of an exposed website, using a simple search engine query:



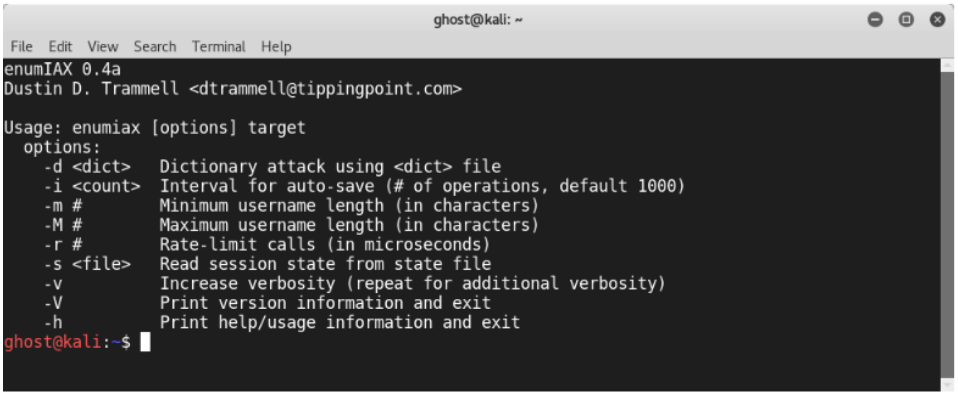
- After collecting juicy information about the target, from an attacker perspective, we usually should perform scanning.

  Banner grabbing is a well-known technique in enumeration, and the first step to enumerate a VoIP infrastructure is by starting a banner grabbing move.

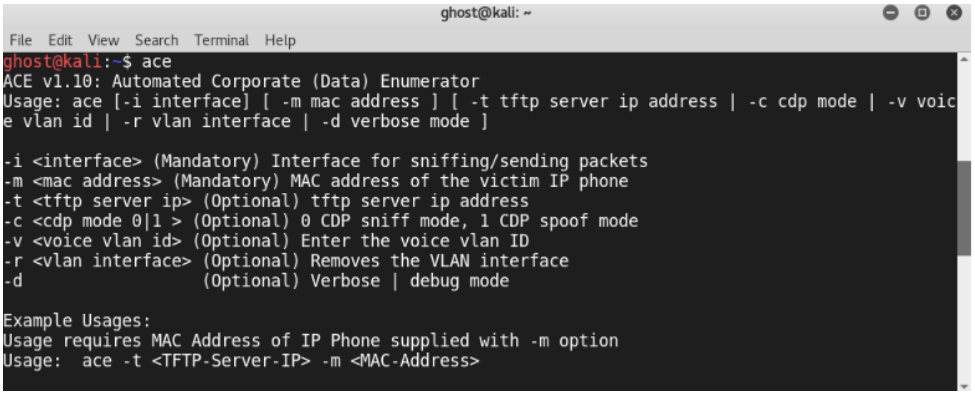
  In order to do that, using the Netcat utility would help you grab the banner easily, or you can simply use the Nmap script named banner: nmap -sV --script=banner <target>



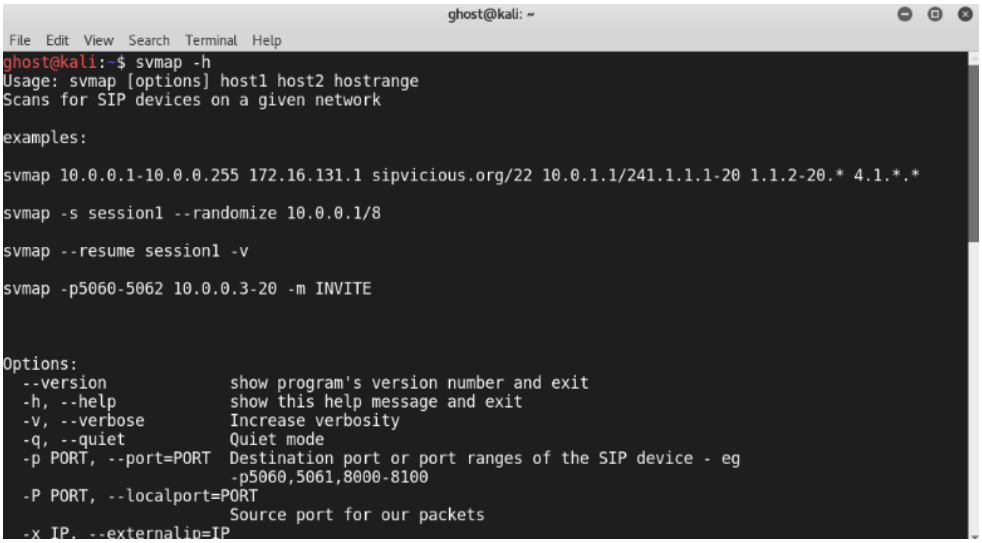
- For a specific vendor, there are a lot of enumeration tools you can use; **EnumIAX** is one of them. It is a built-in enumeration tool in Kali Linux to brute force Inter-Asterisk Exchange protocol usernames:



- **Automated Corporate Enumerator** (**ACE**) is another built-in enumeration tool in Kali Linux:

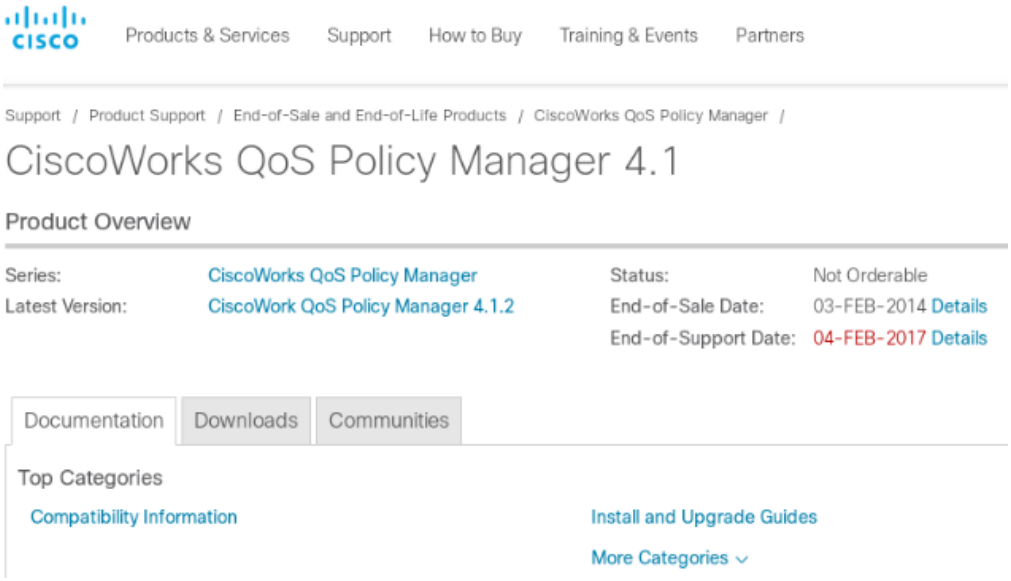


- svmap is an open source built-in tool in Kali Linux for identifying SIP devices. Type `svmap -h` and you will get all the available options for this amazing tool:

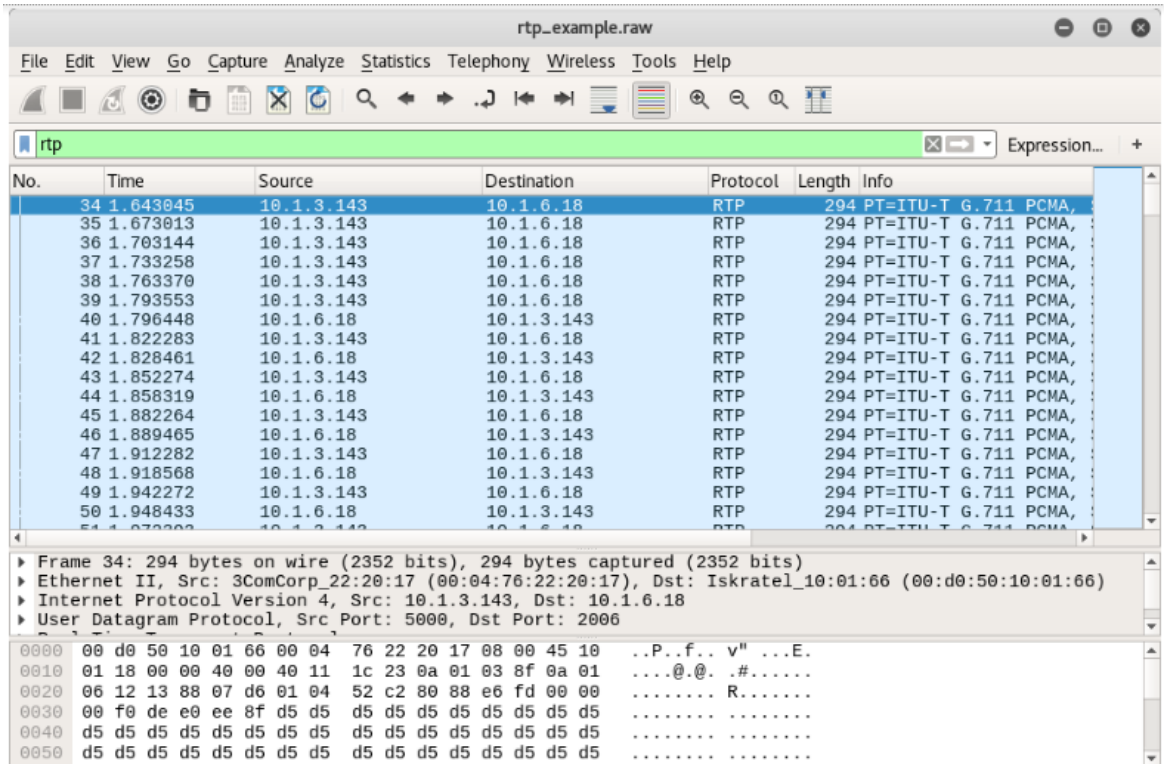


2.3.1 VOIP attack: DOS

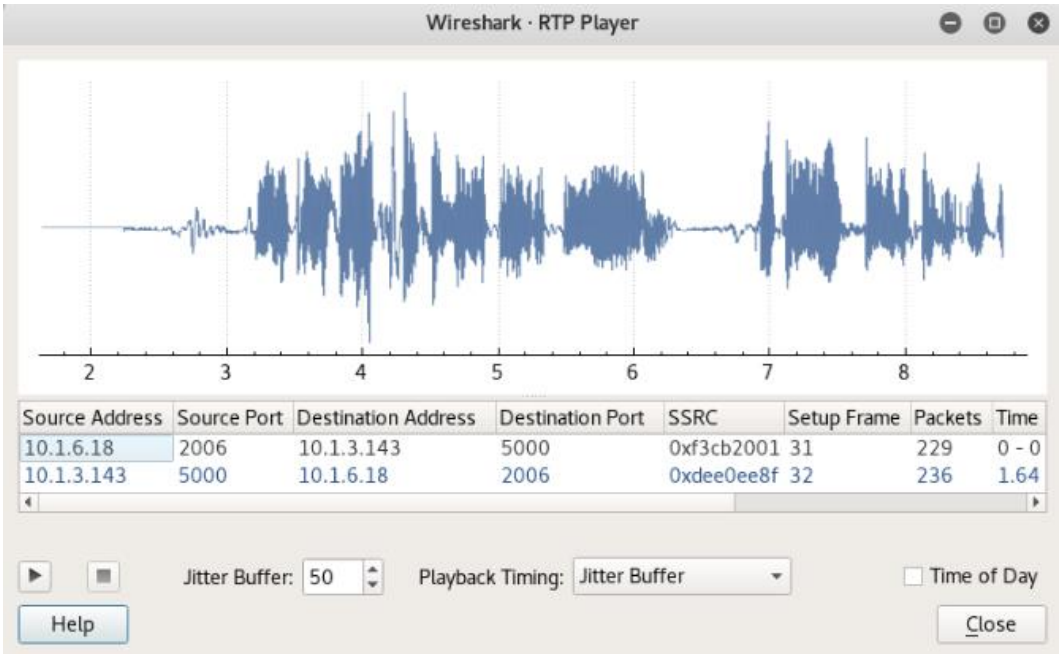
Denial-of-Service (**DoS**) is a threat to the availability of a network. DoS could be dangerous too for VoIP, as ensuring the availability of calls is vital in modern organizations. Not only the availability but also the clearness of calls is a necessity nowadays. To monitor the QoS of VoIP, you can use many tools that are out there; one of them is CiscoWorks QoS Policy Manager 4.1:



To measure the quality of VoIP, there are some scoring systems, such as the **Mean Opinion Score** (**MOS**) or the R-value based on several parameters (jitter, latency, and packet loss). Scores of the mean opinion score range from 1 to 5 (bad to very clear) and scores of R-value range from 1 to 100 (bad to very clear). The following screenshot is taken from an analysis of an RTP packet downloaded from the Wireshark website:



You can also analyze the RTP jitter graph:

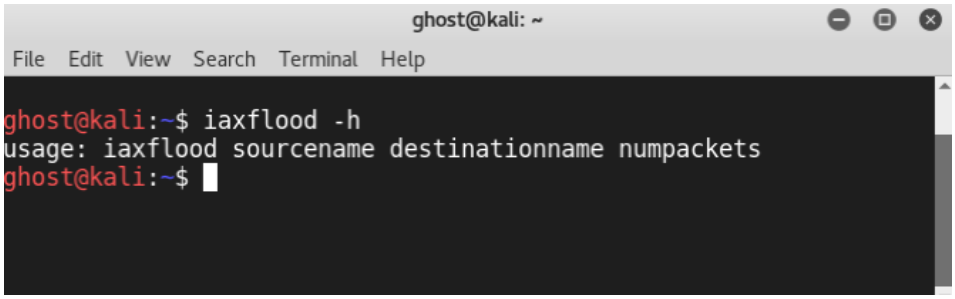


VoIP infrastructure can be attacked by the classic DoS attacks. We saw some of them previously:

- Smurf flooding attack
- TCP SYN flood attack
- UDP flooding attack

One of the DoS attack tools is iaxflood. It is available in Kali Linux to perform DoS attacks. **IAX** stands for **Inter-Asterisk Exchange**.

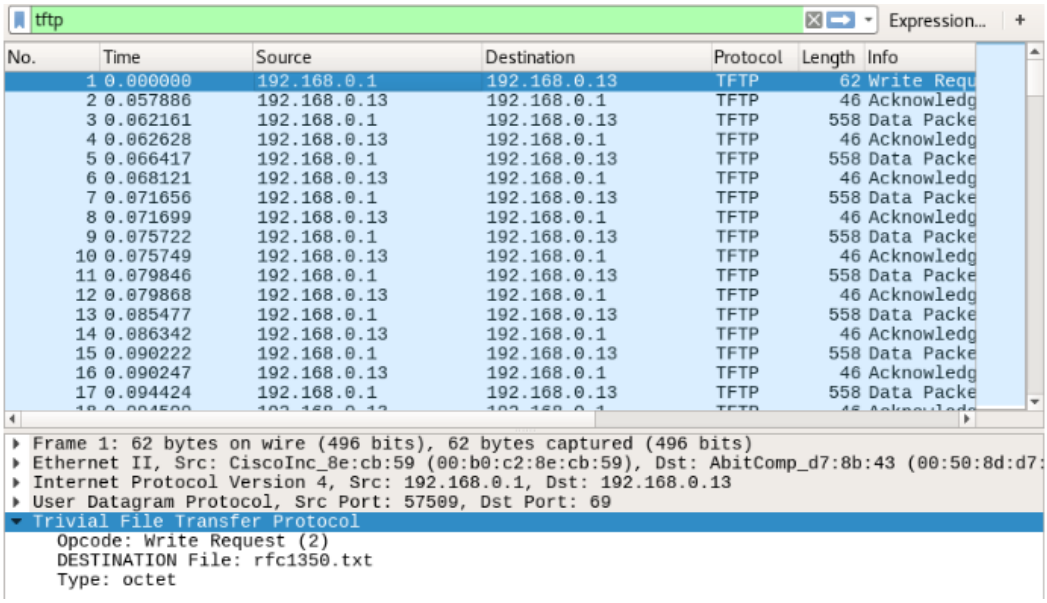
Open a Kali terminal and type iaxflood <Source IP> <Destination IP> <Number of packets>:



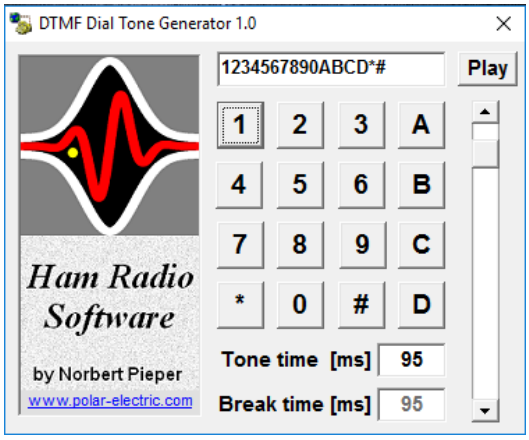
The VoIP infrastructure can not only be attacked by the previous attacks attackers can perform packet Fragmentation and Malformed Packets to attack the infrastructure, using fuzzing tools.

2.3.2 VOIP attack: Eavesdropping

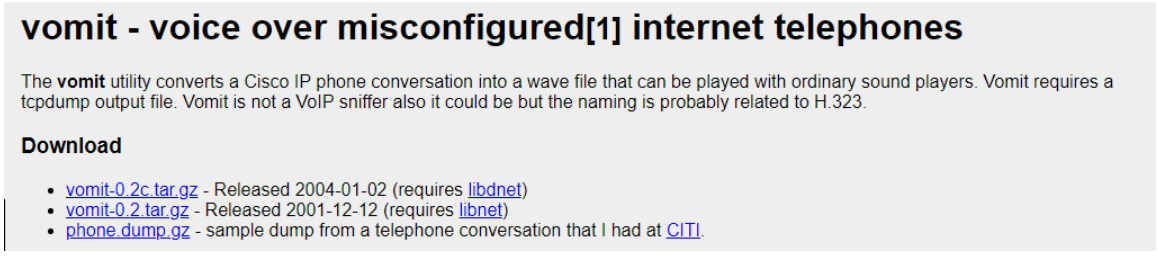
Eavesdropping is one of the most serious VoIP attacks. It lets attackers take over your privacy, including your calls. There are many eavesdropping techniques; for example, an attacker can sniff the network for TFTP configuration files while they contain a password. The following screenshot describes an analysis of a **TFTP** capture:



Also, an attacker can harvest phone numbers and build a valid phone numbers databases, after recording all the outgoing and ongoing calls. Eavesdropping does not stop there, attackers can record your calls and even know what you are typing using the **Dual-Tone Multi-Frequency** (**DTMF**). You can use the DTMF decoder/encoder from this link http://www.polar-electric.com/DTMF/:



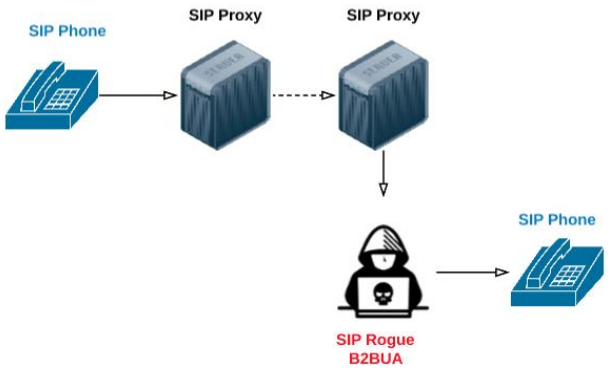
Voice Over Misconfigured Internet Telephones (**VOMIT**) is a great utility to convert Cisco IP Phone conversations into WAV files. You can download it from its official website http://vomit.xtdnet.nl/:



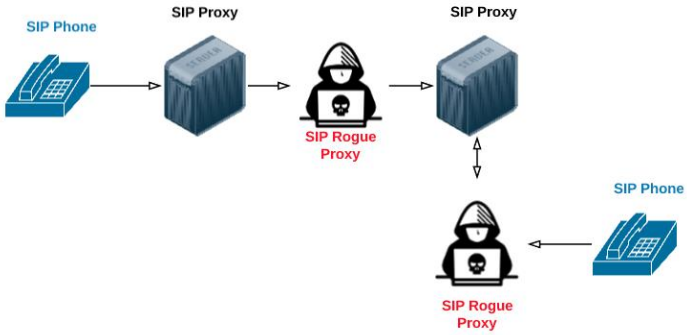
2.3.3 VOIP attack: SIP attacks

Another attacking technique is SIP rogues. We can perform two types of SIP rogues. From an attacker's perspective, we can implement the following:

Rogue SIP B2BUA: In this attacking technique, the attacker mimics SIP B2BUA:



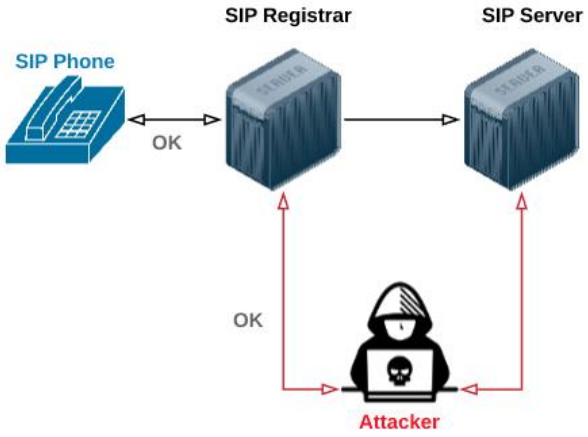
SIP rogue as a proxy: Here, the attacker mimics a SIP proxy:



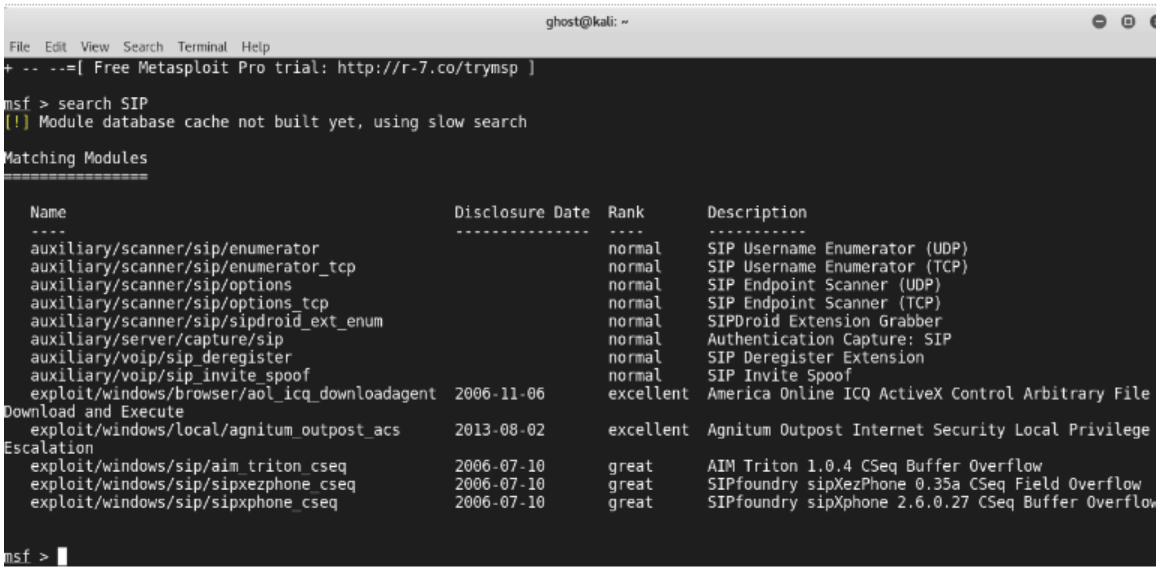
2.3.4 VOIP attack: SIP registration hijacking

SIP registration hijacking is a serious VoIP security problem. Previously, we saw that before establishing a SIP session, there is a registration step. Registration can be hijacked by attackers.

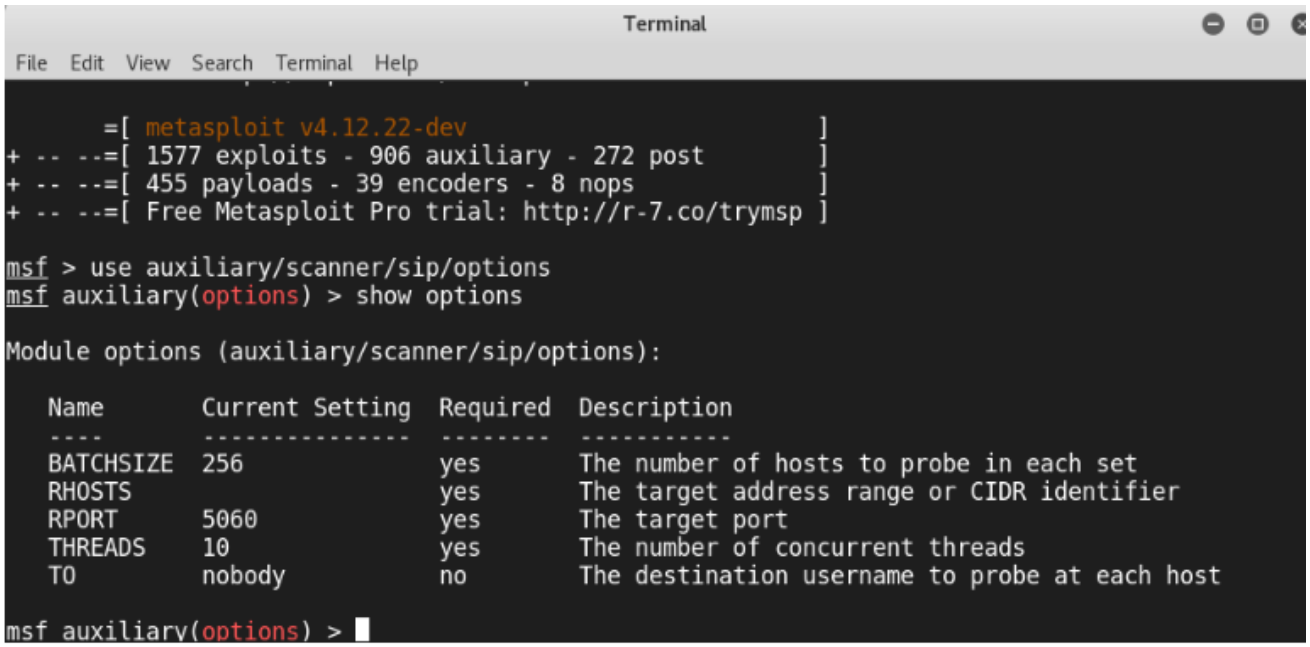
During a SIP registration hijacking attack, the attacker disables a normal user by a Denial of Service, for example, and simply sends a registration request with his own IP address instead of that users because, in SIP, messages are transferred clearly, so SIP does not ensure the integrity of signalling messages:



If you are a Metasploit enthusiast, you can try many other SIP modules. Open a Metasploit console by typing msfconsole and search SIP modules using search SIP:



To use a specific SIP module, simply type `use <module >`. The following interface is an example of SIP module usage



2.3.5 VOIP attack: Spam over Internet Telephony

Spam over Internet Telephony (**SPIT**), sometimes called **Voice spam**, is like email spam, but it affects VoIP. To perform a SPIT attack, you can use a generation tool called **spitter**.

2.3.6 VOIP attack: Embedding malware

Malware is a major threat to VoIP infrastructure. Your insecure VoIP endpoints can be exploited by different types of malware, such as Worms and VoIP Botnets.

Softphones are also a highly probable target for attackers. Compromising your softphone could be very dangerous because if an attacker exploits it, they can compromise your VoIP network. Malware is not the only threat against VoIP endpoints. VoIP firmware is a potential attack vector for hackers. Firmware hacking can lead to phones being compromised.

2.3.7 VOIP attack: Viproy test kit

Viproy VoIP penetration testing kit (v4) is a VoIP and unified communications services pentesting tool presented at Black Hat Arsenal USA 2014 by Fatih Ozavci:

VIPROY

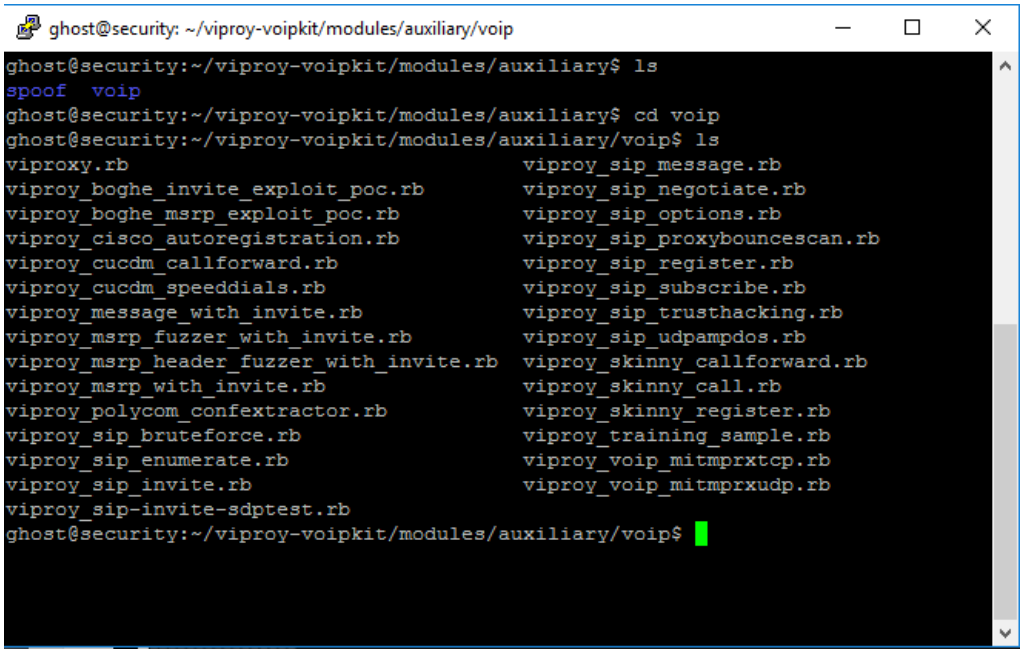
Viproy VoIP Penetration Testing and Exploitation Kit (v4.1)

Project Page : http://www.github.com/fozavci/viproy-voipkit
Download : Viproy 4.1
Author : Fatih Ozavci

To download this project, clone it from its official repository, https://github.com/fozavci/viproy-voipkit:

git clone https://github.com/fozavci/viproy-voipkit.

The following project contains many modules to test SIP and Skinny protocols:



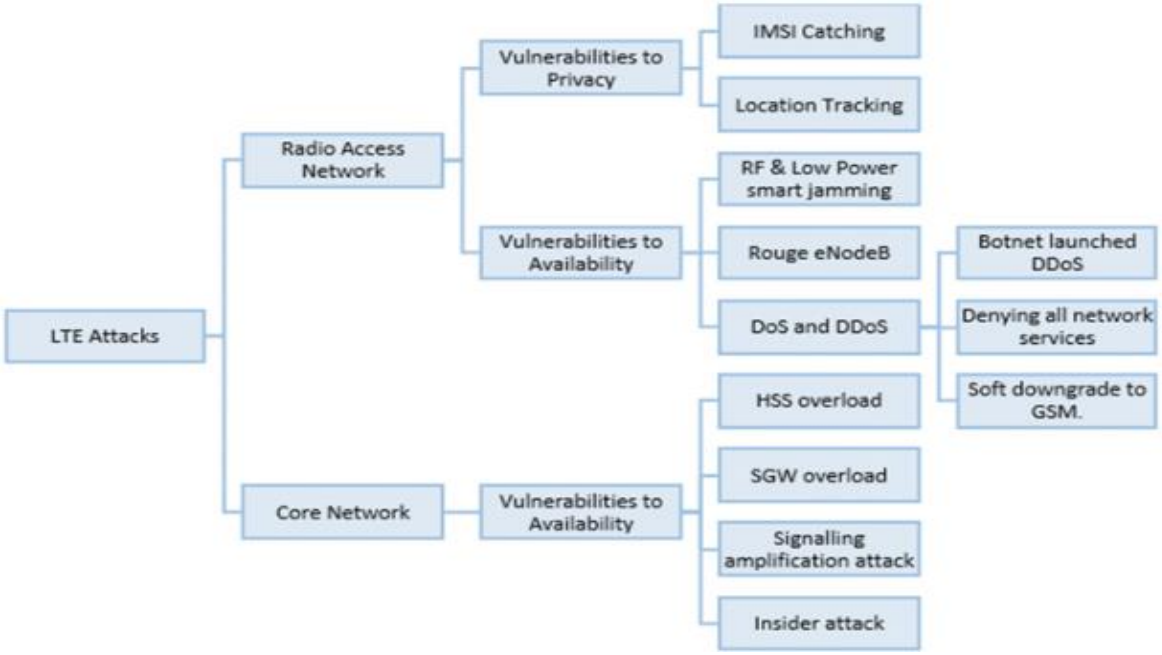
To use them, copy the lib, modules, and data folders to a Metasploit folder in your system.

Thus, in this article, we demonstrated how to exploit the VoIP infrastructure. We explored the major VoIP attacks and how to defend against them, in addition to the tools and utilities most commonly used by penetration testers.

If you've enjoyed reading this, do check out Advanced Infrastructure Penetration Testing to discover post-exploitation tips, tools, and methodologies to help your organization build an intelligent security system.

2.4 LTE threats attack

The vulnerabilities towards radio access network and core network respectively is depicted in the figure down below



2.4.1 IMSI Catching active and passive attack

- **When a mobile device is switched on and attempts a connection to the network:**

- The only way to go through the authentication process is: the IMSI.
- Secondly, during some event when network has never retrieved a Temporary Mobile Subscriber Identity (TMSI) or it is lost, for regular LTE operations, the mobile device has to disclose in the clear its IMSI in such circumstances.

✓ Since the IMSI is transmitted in the system information messages before the actual authentication and encryption takes place, this is precisely where the IMSI catcher exploits the network.

- **An IMSI catching can happen in two separate ways:**

- **Passive way** is to simply observe/eavesdrop the wireless traffic over the air interface and store all the observed IMSIs by decoding system information messages.
- **A semi-passive** adversary is, in addition to passive monitoring, able to trigger signaling messages to subscribers using interfaces and actions that are legitimately available in LTE or in higher layer systems. For example, a semi-passive adversary can trigger paging messages to subscribers by sending a message via a social network or initiating a call. The adversary is assumed to be aware of social identities of subscribers.
  For example, these identities can be a Facebook profile or a mobile phone number of the subscriber. A semi-passive adversary is analogous to the 'honest-but-curious' or 'semi-honest' adversary model used for cryptographic protocols
- **And Active way**, a rogue eNodeB set-up can be accomplished, which will impersonate the real network and simply command each mobile device in its vicinity to identify itself. These commands are forced to disclose user identity: the IMSI.

  Disclosure of the IMSI can compromise user information, location information, and even conversation information.

2.4.2 Location tracking

- **Method 1: Passive attacks** can retrieve IMSI or GUTI to decide whether the target user is present in that area.

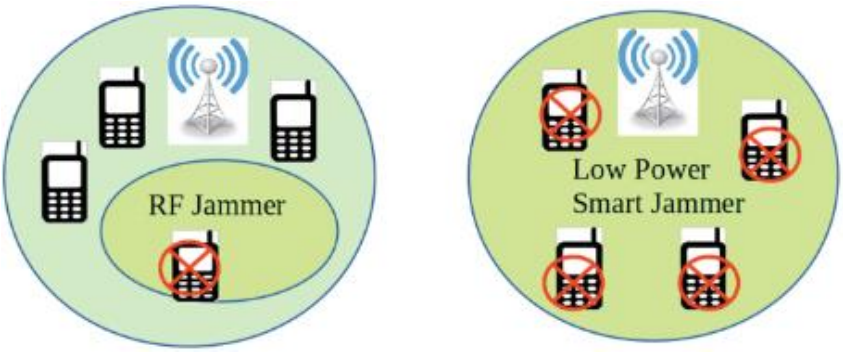
- The Cell Random Network Temporary Identifier (C-RNTI) is a PHY layer identifier, uniquely defined per device within a given cell.
- The figure down below show the C-RNTI assignment during RACH procedure in radio access network. By examining pull notifications in social media applications or silent text messages, an attacker can identify the current C-RNTI of the mobile user's UE during RACH procedure.
- Now, a passive eavesdropper can look for the probable user location in the obtained Tracking Area (TA).

- **Method 2: Also with a passive attack** can take place during handover process; knowing that handovers are network triggered in LTE.
- **Method 3: with semi-passive attacks** (For precise tracking of user location) can produce signaling messages through Volte calls or social media applications like WhatsApp or Facebook and confirm a particular cell within the Tracking Area (TA) for retrieval of paging information.
- **Method 4: with active attacks**: an attacker deploys a rogue eNodeB in the network and reprimands the vulnerabilities present in RRC protocol stack for a more fine-grained location tracking.
- The attacker's main concern is to take advantage of Measurement Report (MR) or Radio Link Failure (RLF) report messages which provides signal strengths, even GPS coordinates under some circumstances of the victim UE.
- The distance between the victim UE and the rogue eNodeB can be easily calculated using trilateration technique or directly from GPS coordinates.

2.4.3 RF and Low Power Smart Jamming.

RF **and** smart jamming are commonly present attacks at PHY layer and to great extent can make network appear unresponsive

- **RF jammers** deliberately disrupt radio communications by decreasing the Signal-to-Noise Ratio (SNR) of the received signal without raising any alerts, causing Denial of Service (DoS).
- The first instance of RF jamming occurred in GSM networks and it was proposed that a local jamming of GSM base station can be accomplished with floods of text messages.
- In legacy GSM networks, text messages share resources with control signaling channels which make jamming quiet feasible.



- Why **Smart Jamming in LTE?**, However, in LTE standards, text message traffic does not share resources with control signaling channels, thus making text based flooding attack on RAN impossible.

- Smart jamming can be performed by saturating one or more of the control channels in both downlink and uplink that are necessary for the UE to access the spectrum.

- Instead of saturating the entire control channel, the attacker will target narrower control channels leading to less power consumption. The Physical Control Format Indicator Channel (PCFICH) is distinctly a sparse channel, turning it to be more vulnerable to sophisticated jamming techniques.

  The PCFICH essentially carries all control information's required to decode Physical Downlink Control Channel (PDCCH) for the UE. According to LTE specifications, since the radio resource allocation of broadcast and downlink synchronization channels (PBCH, PDCH, PSS and SSS) is known beforehand, smart jamming is an easy improvement over basic RF jamming.

  ✓ An attacker would block downlink reception of one or more of the aforementioned control channels; by simply tuning a commercially available off-the-shelf (OTS) radio jammer at the targeted center frequency of the LTE band and transmission bandwidth of at least 1.08 MHz.

  ✓ Similar attack would be possible in uplink control channels too and; given the fact that an attacker is challenging lower-power UEs, the required power to accomplish the jamming will be relatively low.

2.4.4 Rogue eNodeB

- **A rogue eNodeB is a fake base station, illegitimately setup in the LTE network and controlled by an attacker through various widely available open-source software platforms.**

- Under normal conditions, UE always try to scan the nearby eNodeBs and prefers to establish connection with the eNodeB having the highest signal strength.

- **How to build a Rogue eNodeB? Before answer the question we need to understand the difference between Rogue GSM and LTE.**

  ✓ During GSM attacks, the rogue BTS was operated with signal power higher than the neighboring base stations.

  ✓ However, in LTE this procedure will not sustain, as per LTE specifications, when the UE is quite nearer to a serving eNodeB, it apparently avoids scanning neighboring eNodeBs to reduce power utilization.

- **But great news is that a new feature referred as "absolute priority based cell reselection" in LTE can be exploited** to overcome the aforementioned limitation and establish a rogue eNodeB.

  ✓ In this feature, the UE in IDLE state should periodically scan and attach to eNodeB operating with the highest priority frequency.

- Hence, during active attacks, the UE will forcibly attach to rogue eNodeB operating on a frequency having highest cell reselection priority, even when it is closely located to a real eNodeB.

- This cell reselection priority list is present in system information messages broadcasted by a real eNodeB.

- By means of passive or semi-passive attacks, these cell reselection priority list can be sniffed and apparently used to configure the rogue eNodeB.

- **Once the rogue eNodeB is established** with necessary network parameters of a real eNodeB, an attacker can now launch potentially severe threats like :
  - Man-In-The-Middle (MITM) attacks,
  - DoS,
  - adding malicious messages in attach process,
  - deny mobile services to the UEs and
  - downgrade to a non-LTE network

2.4.5 DoS and DDoS Attacks

Denial of Service (DoS) and Distributed Denial of Service (DDoS) both can potentially disrupt the LTE network.

- Traditionally, the **DoS can be performed by sending floods of messages to a target** network and exhaust its bandwidth resources, eventually making the target network unavailable to legitimate UEs.
- Apart from this, other DoS attacks that can lead to network failure are downgrading to a non-LTE network or denying all network service.
- In **DDoS attacks, an attacker can produce heavy volume traffic by launching a botnet** managed via Command and Control Centers or hacked UEs that are well synchronized. Prominent attacks based on DoS and DDoS are discussed here.

2.4.5.1 Botnet Launched DDoS Attack

- **The DDoS attacks can potentially be launched by a botnet of mobile devices or a high volume of malicious traffic** against the LTE network.

  The key idea behind such attack is, many botmasters activate all botnet nodes simultaneously and create bandwidth congestion on both downlink and uplink; resulting into temporary large-scale saturation of the LTE network.

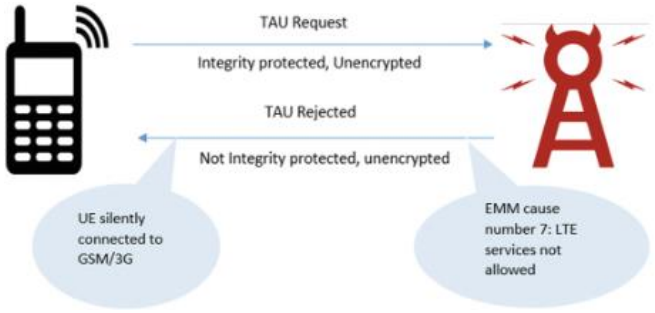
  The severity of such DoS/DDoS attacks is of large extend in the core network (EPC) as compared to radio access network (RAN).

- **The upsurge in the occurrence of mobile malware and affluent virus spread** have improved the likelihood of placing various DoS attacks. Henceforth, a mobile device based botnet introduces a powerful attack vector against LTE network by means of high volume malicious traffic or signalling messages in the network; resulting in a new way of DDoS attack.

2.4.5.2 Soft Downgrade to Non-LTE Services

- An attacker can establish a rogue eNodeB that will reject the UE from accessing LTE services:

- **Method 1:** by abusing the Reject causes messages like “TAU Reject”, which are transmitted without any integrity protection.
  - ✓ Since, no security keys is required for the transmission of “TAU Reject” messages, the rogue eNodeB could target any LTE mobile user within its vicinity for temporary DoS.



Downgrade to non-LTE services

- **Method 2:** A similar threat is also feasible with “Service Reject/Attach” messages.
  - ✓ When the UE transmits “TAU Request” message to a rogue eNodeB, it is still attached to the real network, hence under the NAS security context, this message is integrity protected but not encrypted. This would turn out as an advantage to the attacker, who could easily decode it and responds with a “TAU Reject” message (EMM cause number 7: LTE services not allowed) which as per LTE specifications, does not require integrity protection.
  - ✓ Upon the reception of “TAU Reject” message, the UE will accept the reject cause and proceeds to act further, by removing all existing services associated with the real network.
  - ✓ As a result, the UE considers itself invalid for any LTE services unless a rebooting or USIM re-insertion happens.
  - ✓ Furthermore, the UE will not search for or send TAU Attach request to any nearby legitimate LTE network, triggering temporary Denial of Service (DoS) and is of less impact.

- **However, in context to gaining network services, the UE may search for GSM or 3 G network**.

- By downgrading to non-LTE networks like 2G or 3G, a DoS threat can be triggered by an attacker; which would not only open doors to attacks like a full man in the middle attack, active eavesdropping to phone calls or text messages, but also make complete loss of LTE services.

- As long as the UE does not loose connectivity to non-LTE networks, the user might not even realize it is connected by GSM or 3G network.

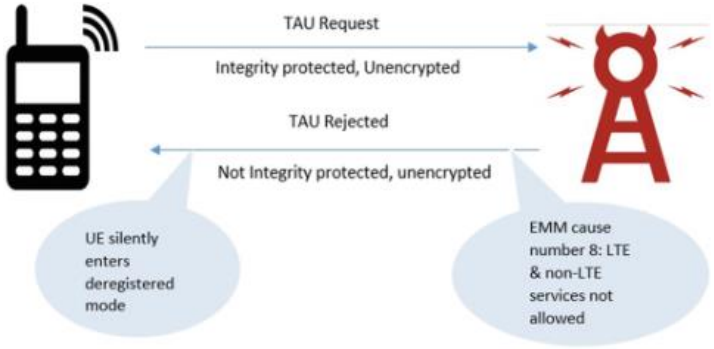
2.4.5.3 Denying All Network Services

- **A similar threat can be accomplished by placing "TAU Reject" message with EMM cause number 8: "LTE and non-LTE services not allowed".**

- In such scenario, the UE again considers itself invalid for any LTE services unless a rebooting or USIM re-insertion happens.

- The UE further enters into a state of "EMM-DEREGISTERED", which makes it unknown to MME, consequently causing a persistent Denial of Service (DoS).

- The UE will never attempt to connect GSM, 3G or LTE networks despite being available.



Deny both LTE and non-LTE services

2.4.6 HSS Overload

- **Therefore, a DoS/DDoS attack achieved by means of HSS overload can severely obstruct the network operation and demote network services.**

- The HSS overload can be happen when an attacker disguising a legitimate UE, constantly transmit fake IMSIs to HSS.

- Consequently, the HSS has to generate excessive authentication vectors for the UE to complete authentication process.

- This repetitive generation of authentication vectors will force HSS to consume excessive computational resources, definitely leading to HSS overload.

2.4.7 SGW Saturation

- Serving Gateway (SGW) may face saturation due to threats like:
- **flood of "bearer setup messages" to the SGW**
- And a **programmable mobile phone incessantly triggering Tracking Area Update (TAU)** procedure in a short period of time.
  - ✓ During the signaling procedure, MME implements the necessary security mechanism for Tracking Area Update (TAU) that includes the authentication and integrity check of context request message. This guarantees correct signaling and legitimate users, but signaling integrity takes place before user authentication in LTE network.
  - ✓ Therefore, MME may unknowingly presume the initiating device as an already validated user and will not undergo authentication process unless signaling integrity is broken.
  - ✓ This opens the door to DoS/DDoS threats against MME through illegitimate users. The MME sends Create Bearer Request to the new Serving Gateway (SGW) whenever the UE move towards a new TA.
- **Now, If the MME is compromised by an attacker**, it will sends floods of bearer setup messages to the new SGW in a very short span of time, apparently leading to SGW overload.
- Also, if a programmable mobile phone incessantly triggers TAU procedure through a stored program in a short span of time, these requests are forwarded to SGW, which again can lead to SGW overload.

2.4.8 Signaling Amplification Attacks

Practically, mobile networks are deployed to sustain peak traffic hours and does not have adequate radio resources available for each user at the same time.

- The scarce spectrum gives rise to advanced wireless techniques, which can lead to more efficient reuse of idle resources. For example, the RRC layer in the LTE network is responsible for reassigning radio resources from an established UE to another UE, when the connection of the former goes idle for a short span of time.
- This reassignment of radio resources takes place when an inactivity timer expires, thus triggering a disconnection of radio bearer between the established UE and the core network.
- ✓ A significant amount of control plane messages are exchanged among many nodes within the core network during each instance of radio bearer setup and disconnection.
- ✓ And these signaling overhead, if not efficiently monitored, can lead to large-scale saturation of core network, which subsequently can get exploited in context of DDoS.
- **So how this attack can be happen?**
- **Method 1:** Signaling amplification attack like launching a **botnet** of mobile device could force each UE to constantly establish and release connection with the core network.
- **Method 2:** Another threat to EPC in reference to DDoS can also take place, when a piece of **malware** can trigger all UEs to reboot simultaneously, thereby overloading the EPC with registration requests.
- ✓ It is also necessary to consider that, HSS too is involved in a significant number of signaling processes at the EPC; thus can as well suffer from signaling amplification attack.

2.4.9 Insider Attack

The threats to availability of wireless communication networks must take in account the problems associated with insider threat, which are often assumed unlikely to occur.

Also, with the current security threat landscape and impact of new player like Advanced Persistent Threat (APT), insider attacks have become highly relevant.

A very well-funded attacker can maliciously persuade an insider who has privileges to access core network elements, to remotely or physically shut down a particular network node in the core network. This would eventually pose an attack against availability by obstructing the normal radio communication in the LTE core network.

- **The insider attacks at RAN can be :**

- Shutdown of eNodeB,
- Purposefully jamming LTE
- Downgrade to non-LTE networks
- Launching botnet of mobile devices.

- **At core network, the insider attacks could:**

- Open doors to potentially global breakdowns like shutdown of HSS, SGW saturation or any node damage.
- The impact of the insider attacks would not be global unless the affected node is HSS or SGW.

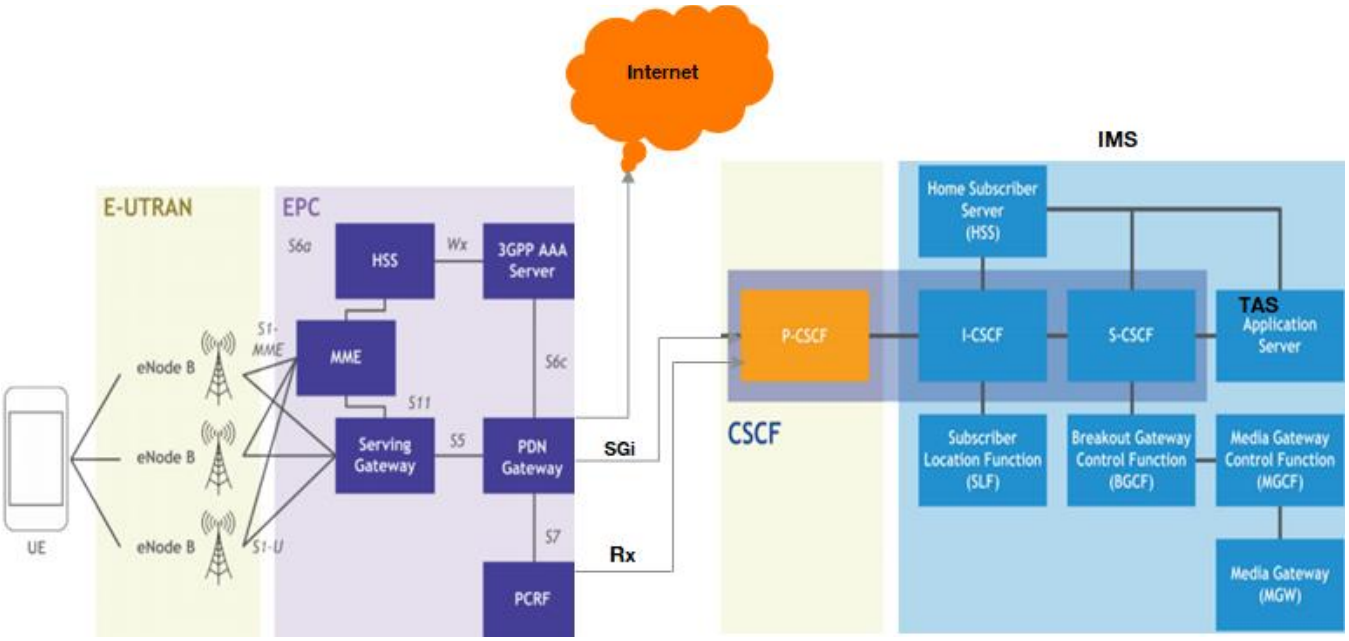
2.5 VOLTE threats attack

Voice over LTE (VoLTE) carries voice over 4G networks by the help of IMS core network, without IMS there is no VOLTE, Its call quality is higher than the other VoIP variants, in addition to providing better coverage.
VoLTE, as well as the other voice technologies, faces various threats from attackers.

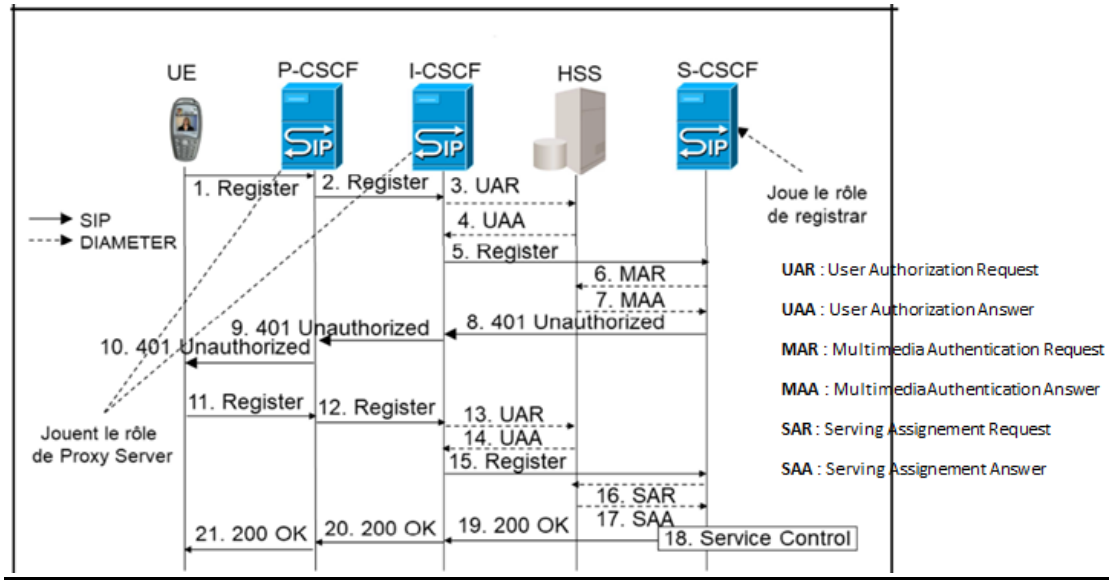
- Sniffing VoLTE interfaces
- Exposed keys in GSM SIM
- User location manipulation
- Roaming information manipulation
- Side channel attack

In another way to hack volte catch LTE

2.5.1 VOLTE architecture



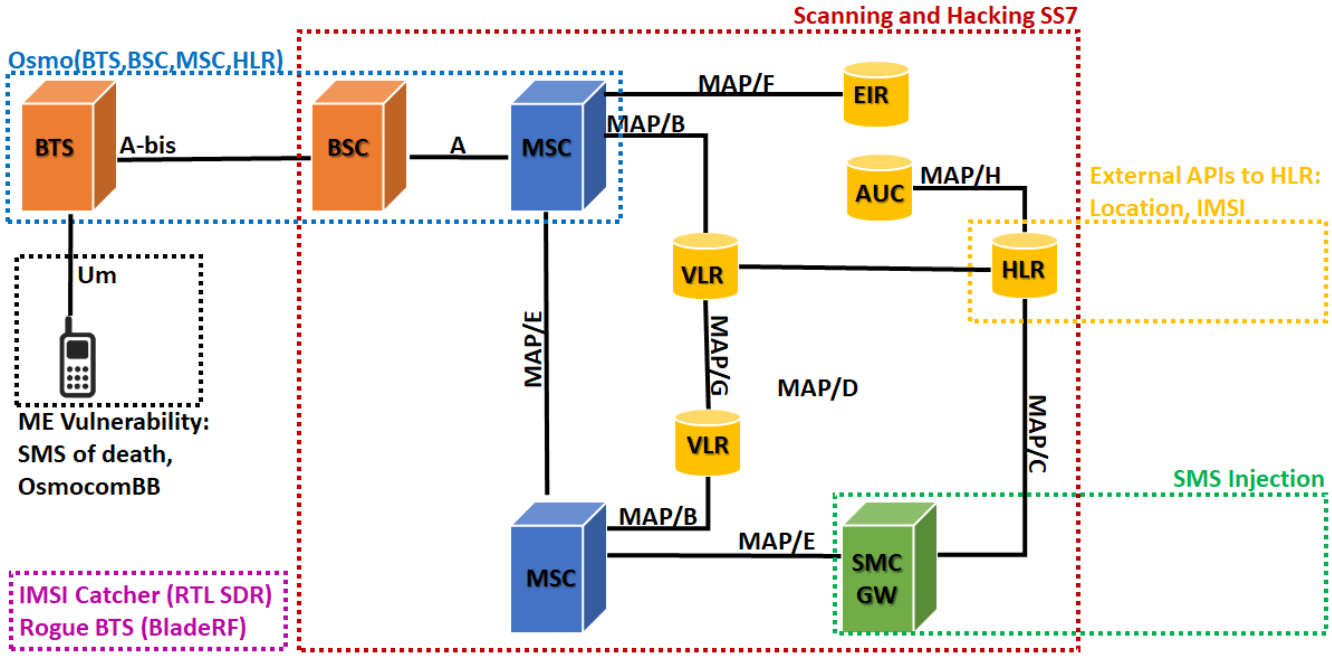
2.5.2 VOLTE attachement



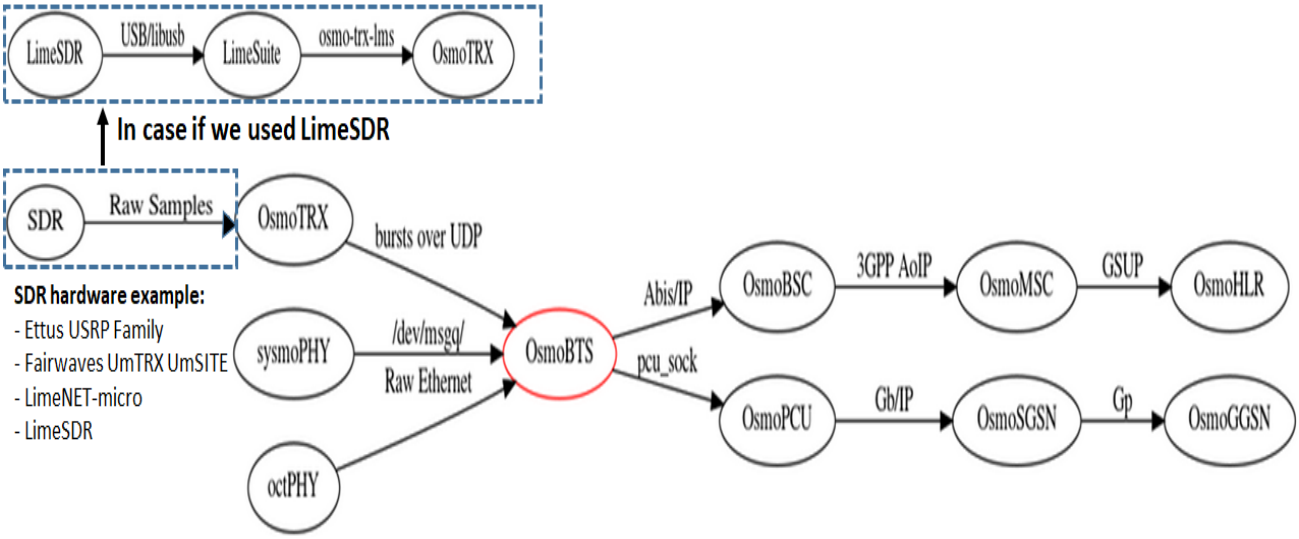
Here is a typical IMS SIP registration call flow:

1. The IMS client attempts to register by sending a **REGISTER** request to the P-CSCF.
2. The P-CSCF forwards the REGISTER request to the I-CSCF.
3. The I-CSCF polls the HSS for data used to decide which S-CSCF should manage the REGISTER request. The I-CSCF then makes that decision.
4. The I-CSCF forwards the REGISTER request to the appropriate S-CSCF.
5. The S-CSCF typically sends the P-CSCF a 401 (UNAUTHORIZED) response as well as a challenge string in the form of a "number used once" or "nonce".
6. The P-CSCF forwards the 401 – UNAUTHORIZED response to the UE.
7. Both the UE and the network have stored some Shared Secret Data (SSD), the UE in its ISIM or USIM and the network on the HSS. The UE uses an algorithm per RFC 33101 (e.g. AKAv2-MD5) to hash the SSD and the nonce."
8. The UE sends a REGISTER request to the P-CSCF. This time the request includes the result of the hashed nonce and SSD.
9. The P-CSCF forwards the new REGISTER request to the I-CSCF.
10. The I-CSCF forwards the new REGISTER request to the S-CSCF.
11. The S-CSCF polls the HSS (via the I-CSCF) for the SSD, hashes it against the nonce and determines whether the UE should be allowed to register. Assuming the hashed values match, the S-CSCF sends 200 – OK response to the P-CSCF. At this point an IPSec security association is established by the P-CSCF.
12. The P-CSCF forwards the 200 – OK response to the UE.

3 How to get in GSM Network

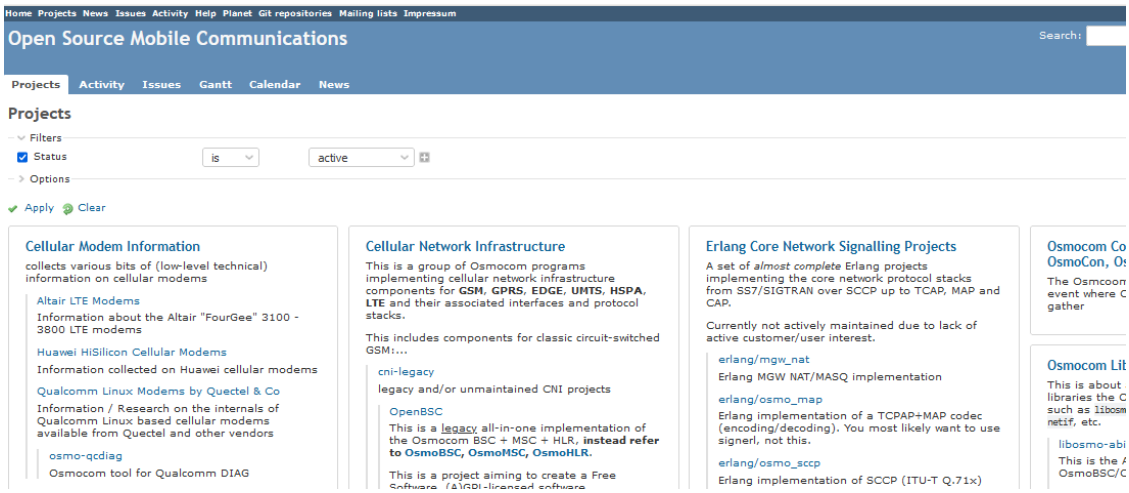


- **With the help of Osmocom we can build this system for our hack: and then we can apply all hacking technique we saw it in chapter 2**

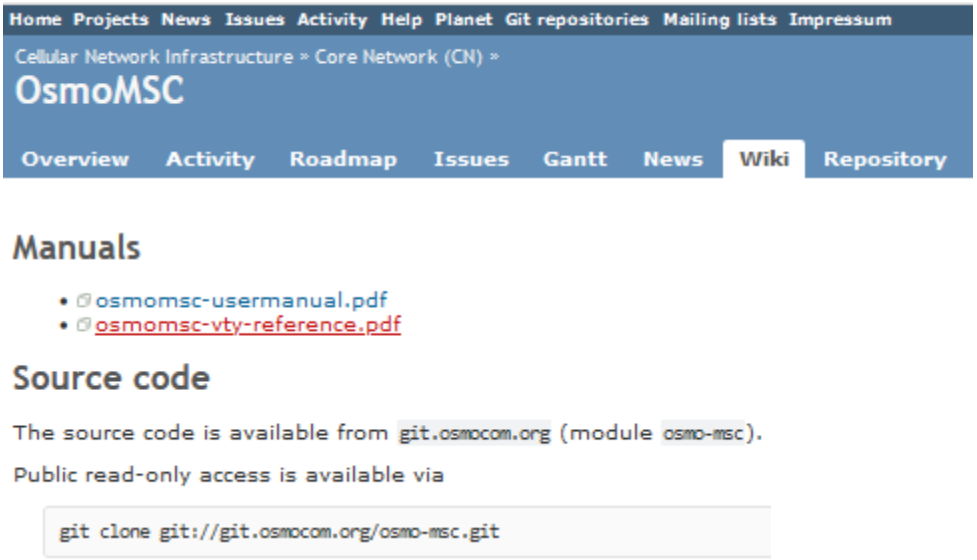


So what is Osmocom? The Osmocom project is an umbrella project regarding Open source mobile communications. This includes software and tools implementing a variety of mobile communication standards, including GSM, DECT, TETRA and others.

- Under each umbrella projects there is a lot of projects, and in our resume we will focus on these umbrella project:
  - Cellular Network Infrastructure we will focus on : OsmoBTS, OsmoBSC, OsmoMSC, OsmoTRX, OsmoHLR
  - SDR (Software Defined Radio)
  - OsmocomBB



- In each projects for example OsmoMSC there is 2 things that we need to take care about it in the “wiki” page:
  - Source code: this we will need it for the installation of the fake OsmoMSC
  - Manuals : this we will use it to understand and to configure the OsmoMSC

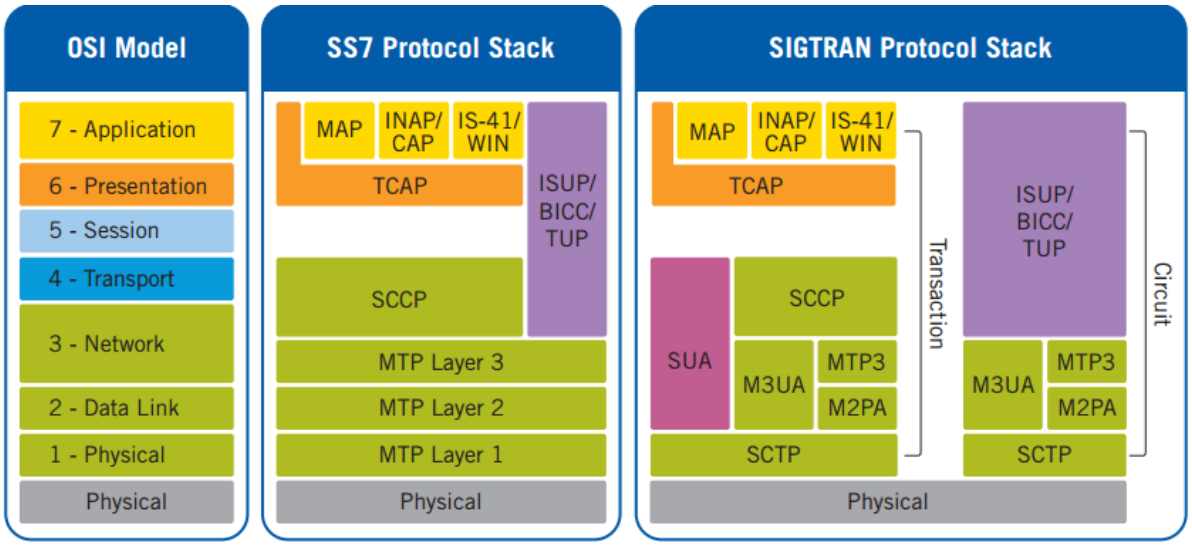


- As you can imagine a lot of things to read in the manual for setup the Osmo… But this dude “nickvsnetworking.com” make it easy for us to configure directly the specified equipment OsmoMSC etc….

  But I just want to mention that in case you faced any problem or you want to get more hand control on OsmoMSC, OsmoBTS etc… the manuals will be your best friend

3.1 Scanning and Hacking SS7

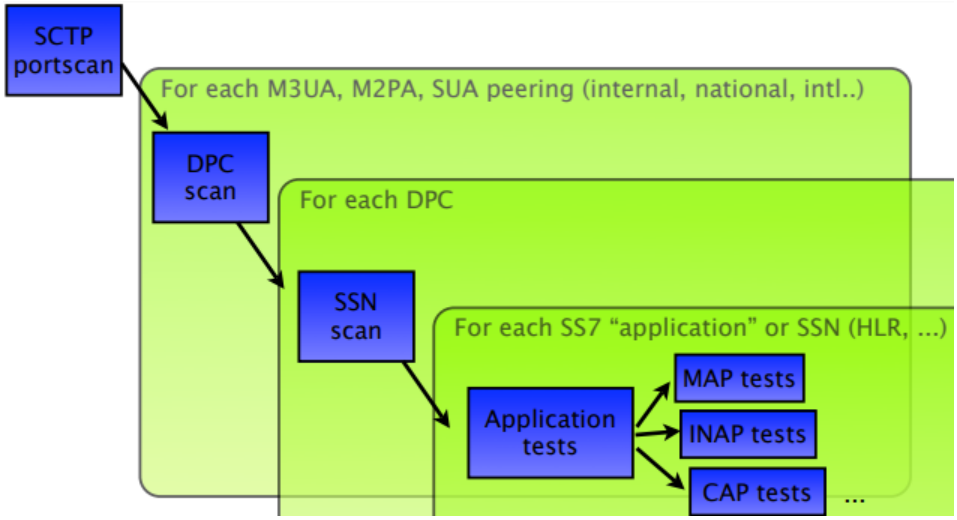
First let's compare SS7 with the OSI layers



3.1.1 SS7/SIGTRAN vs TCP

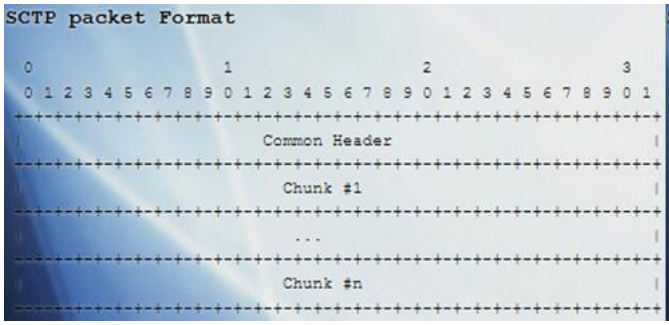
TCP/IP	SS7
IPsec endpoint scan, MPLS label scan, VLAN tag scan	SCTP endpoint scan
Arp or Ping scan	MTP3 or M3UA scanning
Ping scan using TCP SYN	SCCP DPC scanning
TCP SYN or UDP port/service scanning	SCCP SSN (SubSystem Number) scanning
Application (*AP) traffic injection (e.g. MAP, INAP, CAP, OMAP...)	Service-specific attacks and abuses (e.g. attacks over HTTP, SMB, RPC, ...)

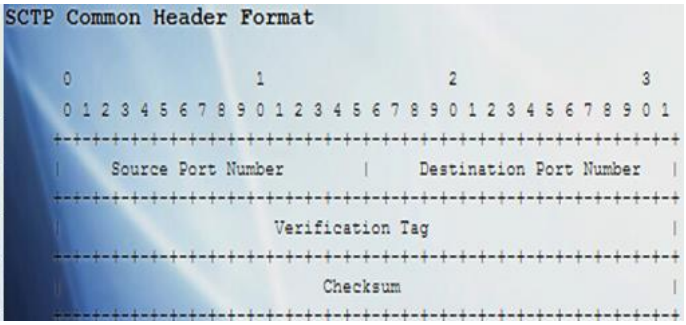
3.1.2 SS7/SIGTRAN audit strategy



3.1.3 SCTP scaning: mapping SIGTRAN

SCTP is part of the SIGTRAN protocol family that is used to carry SS7 over TCP/IP among other things. Using an SCTP scan allows you to find entry points to Telecom networks.





- **SCTP Chunk types**

ID Value	Chunk Type
0	- Payload Data (DATA)
1	- Initiation (INIT)
2	- Initiation Acknowledgement (INIT ACK)
3	- Selective Acknowledgement (SACK)
4	- Heartbeat Request (HEARTBEAT)
5	- Heartbeat Acknowledgement (HEARTBEAT ACK)
6	- Abort (ABORT)
7	- Shutdown (SHUTDOWN)
8	- Shutdown Acknowledgement (SHUTDOWN ACK)
9	- Operation Error (ERROR)
10	- State Cookie (COOKIE ECHO)
11	- Cookie Acknowledgement (COOKIE ACK)
12	- Reserved for Explicit Congestion Notification Echo (ECNE)
13	- Reserved for Congestion Window Reduced (CWR)
14	- Shutdown Complete (SHUTDOWN COMPLETE)

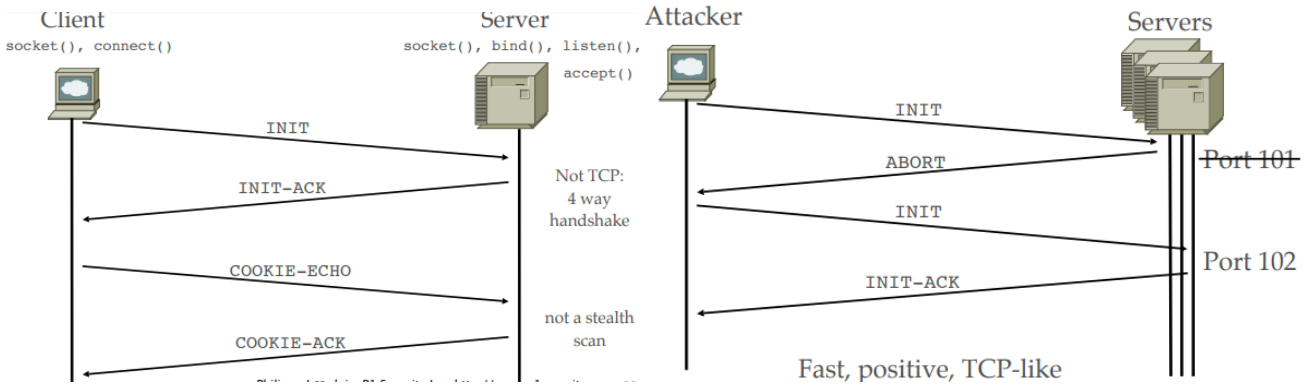
3.1.3.1 SCTP INIT Scan

SCTP is an alternative to the TCP and UDP protocols, combining most characteristics of TCP and UDP, and also adding new features like multi-homing and multi-streaming.

It is the SCTP equivalent of a TCP SYN scan. It is able to scan thousands of ports per second on a fast network not hampered by restrictive firewalls.

Like SYN scan, INIT scan is relatively unobtrusive and stealthy, since it never completes SCTP associations. It also allows clear, reliable differentiation between the open, closed, and filtered states.

Since you don't open a full SCTP association, this technique is known as half-open scanning. You send an INIT chunk as if you are going to open a real association and then wait for a response.



SCTP 4-way handshake **stealth scan**

https://www.p1sec.com/corp/research/tools/sctpscan/ git clone git@github.com:philpraxis/sctpscan.git

```
root@gate:~/sctp# ./sctpscan --scan --autoportscan
-r 203.151.1
Netscanning with Crc32 checksumed packet
203.151.1.4 SCTP present on port 2905
203.151.1.4 SCTP present on port 7551
203.151.1.4 SCTP present on port 7701
203.151.1.4 SCTP present on port 8001
203.151.1.4 SCTP present on port 2905
root@gate:~/sctp#
```

```
root@gate:~/sctp# ./sctpscan-v11 --scan --autoportscan -r
 203.151.1
Netscanning with Crc32 checksumed packet
203.151.1.4 SCTP present on port 2905
203.151.1.4 SCTP present on port 7102
203.151.1.4 SCTP present on port 7103
203.151.1.4 SCTP present on port 7105
203.151.1.4 SCTP present on port 7551
203.151.1.4 SCTP present on port 7701
203.151.1.4 SCTP present on port 7800
203.151.1.4 SCTP present on port 8001
203.151.1.4 SCTP present on port 2905
root@gate:~/sctp#
```

3.1.3.2 SCTP scan with nmap

You can use SCTP packets to determine if a host is online by sending SCTP INIT packets and looking for ABORT or INIT ACK responses. This technique is named SCTP INIT ping scan that is implemented by Nmap.

However, to scan a port with SCTP, first, open your terminal and run: nmap -sn -PY <target>

```
nmap -sn -PY scanme.nmap.org

Nmap scan report for scanme.nmap.org (45.33.32.156)

Host is up (0.15s latency).

Other addresses for scanme.nmap.org (not scanned):

2600:3c01::f03c:91ff:fe18:bb2f

Nmap done: 1 IP address (1 host up) scanned in 4.31 seconds
```

3.1.4 More scanning tools

3.1.4.1 GTScan

GTScan relies on using empty TCAP layers as probes to detect listening subsystem numbers (i.e application port numbers like 80 for http, 443 for https but for telecom nodes) on the respective global titles. With this way will be able to map the network and use the results to conduct targeted direct attacks to the respective nodes.

GTScan includes Message handling: Return message on error in the SCCP layer to determine from the response what is the scanned node. If a TCAP abort message is returned with an error p-abortCause: unrecognizedMessageType (0) thus the destination nodes is listening on the SSN that was scanned, else then the scanner continues scanning on other SSNs

You can provide GTscan a range of global titles to be scanned, a comma-separated or a single GT to be scanned, along with other parameters

- **Installation in this link:** https://github.com/SigPloiter/GTScan

3.1.4.2 SigPloit-ss7

SiGploit a signaling security testing framework dedicated to Telecom Security professionals and reasearchers to pentest and exploit vulnerabilites in the signaling protocols used in mobile operators regardless of the geneartion being in use. SiGploit aims to cover all used protocols used in the operators interconnects SS7, GTP (3G), Diameter (4G) or even SIP for IMS and VoLTE infrastructures used in the access layer and SS7 message encapsulation into SIP-T. Recommendations for each vulnerability will be provided to guide the tester and the operator the steps that should be done to enhance their security posture

SiGploit is developed on several versions. Note: In order to test SS7 attacks, you need to have an SS7 access or you can test in the virtual lab with the provided server sides of the attacks, the used values are provided.

Version 1: SS7

SiGploit will initially start with SS7 vulnerabilities providing the messages used to test the below attacking scenarios A- Location Tracking B- Call and SMS Interception C- Fraud

Version 2: GTP

This Version will focus on the data roaming attacks that occur on the IPX/GRX interconnects.

Version 3: Diameter

This Version will focus on the attacks occurring on the LTE roaming interconnects using Diameter as the signaling protocol.

Version 4: SIP

This is Version will be concerned with SIP as the signaling protocol used in the access layer for voice over LTE(VoLTE) and IMS infrastructure. Also, SIP will be used to encapsulate SS7 messages (ISUP) to be relayed over VoIP providers to SS7 networks taking advantage of SIP-T protocol, a protocol extension for SIP to provide intercompatability between VoIP and SS7 networks

Installation in this link: https://github.com/ethicalhackeragnidhra/SigPloit-ss7
https://github.com/alex14324/ss7

3.1.4.3 M3UA scan

M3UA scan is simple scanner that aims to help pentesters to identify nodes that has sctp ports opened with m3ua on top of it.

Detecting a node with m3ua is an indication that this is a node core node in a telecom infrastructure that provides signaling. This scanner could be helpful to identify signaling nodes exposed on the internet that could be compromised and used as a gate to the SS7 network.

One benefit could be testing if telecom nodes are hardened and only forming sctp associations with the nodes that supposed to connect to only, testing if there is some filtering done on the nodes to prevent anyone to perform sctp associations with it thus connect to the network.

Installation in this link: https://github.com/SigPloiter/M3UAScan

3.1.4.4 HLR-Lookups

This script is used to automate hlr-lookup process using the api from hlr-lookups and extract important data from it like the IMSI of the subscriber, which country is he/she roaming in that is provided in the current MSC GT along with the HLR GT, these information could be useful to conduct further attacks... using sigploit.

Must be noted that those public services like HLR-Lookups are using one variant of SS7 messages that is most probably SendRoutingInformationForSM(SRISM) that is used to locate the target location before sending SMS, some operators implement an SMS Firewall/proxy that scrambles the IMSI and/or HLR and returns back a fake MSC.

To overcome this kind of protection it's always recommended to perform TCAP scanning on the range of GT as the implementation of such scrambling is very weak and predictable. Thus it's recommended to run the script twice for each msisdns to make sure that the returned information is the same and is not changing per request

Installation in this link: https://github.com/SigPloiter/HLR-Lookups

3.1.4.5 GTping

Like ping(8), but uses GTP ping requests to ping GGSNs and anything else that will answer them.

Installation in this link: https://github.com/ThomasHabets/gtping

HandBook: https://www.slideshare.net/naotomatsumoto/gtp-56187797
https://www.slideshare.net/kentaroebisawa/gtping-how-to

3.1.4.6 ss7MAPer

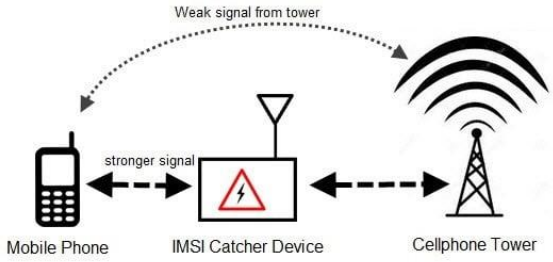
SS7 MAP (pen-)testing toolkit.

DISCONTINUED REPO, please use: https://github.com/0xc0decafe/ss7MAPer/

Installation in this link: https://github.com/ernw/ss7MAPer

3.2 IMSI catcher (RTL-SDR)

IMS Catchers are devices that act like fake cell towers, which trick a target's device to connect to them and then relay the communication to an actual cell tower of the network carrier. The target's communications in the form of calls, text messages, internet traffic etc. go through the IMSI Catcher, which can read messages, listen to the calls and so on



- **Hardware we need:** Any of the hardware below can be used for practical purposes.
- RTL-SDR
- Hackrf
- USRP
- Blade-RF

- **Software we need:** The following software tools are required for practical purposes.
- **GR-GSM:** A python module, which is used for receiving information transmitted by GSM.
- **Wireshark:** Capturing the wireless traffic.
- **IMSI-Catcher:** This program shows the IMSI number, country, brand and operator of cellphones.
- **GQRX:** Software defined radio receiver.
- **RTL-SDR Tools:** Get the information of the RTL SDR dongle.
- **Kailbrate:** Determine the signal strength

- **Installation of Wireshark, GQRX, GR-GSM, rtl-sdr**

```
sudo apt-get update

sudo apt-get install gnuradio gnuradio-dev git cmake autoconf libtool pkg-config g++ gcc make libc6
libc6-dev libcppunit-1.14-0 libcppunit-dev swig doxygen liblog4cpp5v5 liblog4cpp5-dev python3-scipy
gr-osmosdr libosmocore libosmocore-dev rtl-sdr osmo-sdr libosmosdr-dev libboost-all-dev libgmp-dev
liborc-dev libboost-regex-dev python3-docutils build-essential automake librtlsdr-dev libfftw3-dev
gqrx wireshark tshark
git clone -b maint-3.8 https://github.com/velichkov/gr-gsm.git

cd gr-gsm
mkdir build
cd build
cmake ..
make
sudo make install
sudo ldconfig
export PYTHONPATH=/usr/local/lib/python3/dist-packages/:$PYTHONPATH
```

- **Installation of Kalibrate**

```
sudo apt-get update
git clone https://github.com/steve-m/kalibrate-rtl
cd kalibrate-rtl
./bootstrap && CXXFLAGS='-W -Wall -O3'
./configure
make
sudo make install
```

- **Installation of IMSI Catcher**

```
sudo apt install python-numpy python-scipy python-scapy
git clone https://github.com/Oros42/IMSI-catcher.git
```

3.2.1 Capturing the GSM traffic

- For this practical, the RTL-SDR dongle was used. Open the terminal and run the below command to check the dongle has been plugged in successfully.

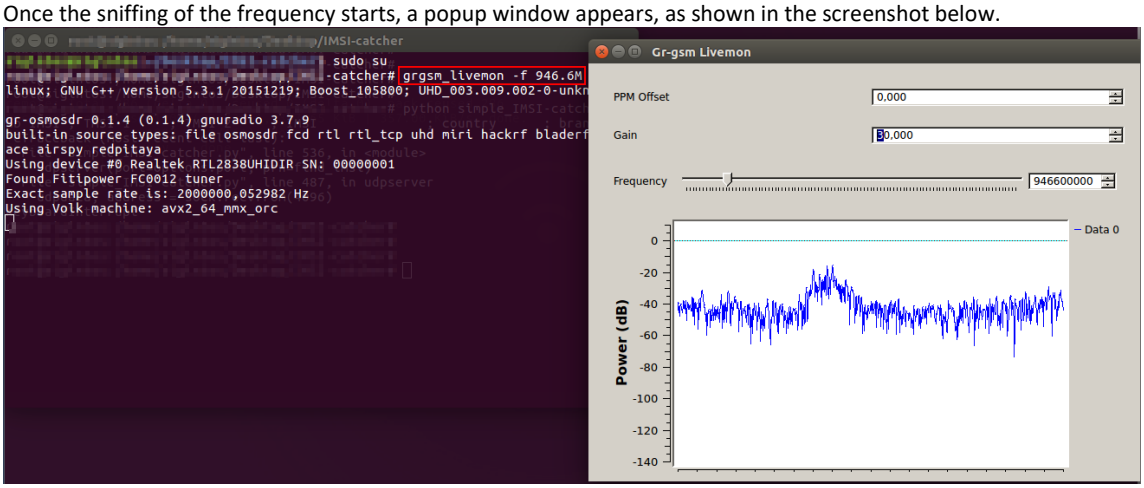
```
# lsusb
Bus 001 Device 004: ID 0bda:2838 Realtek Semiconductor Corp. RTL2838 DVB-T
```

- In France Mobile GSM networks work on 900MHz and 1800MHz frequency bands (Uplink and Downlink).
  The help guide of the "grgsm scanner" tool. Search for nearby GSM base stations using "Kalibrate" or "grgsm_scanner" tools.

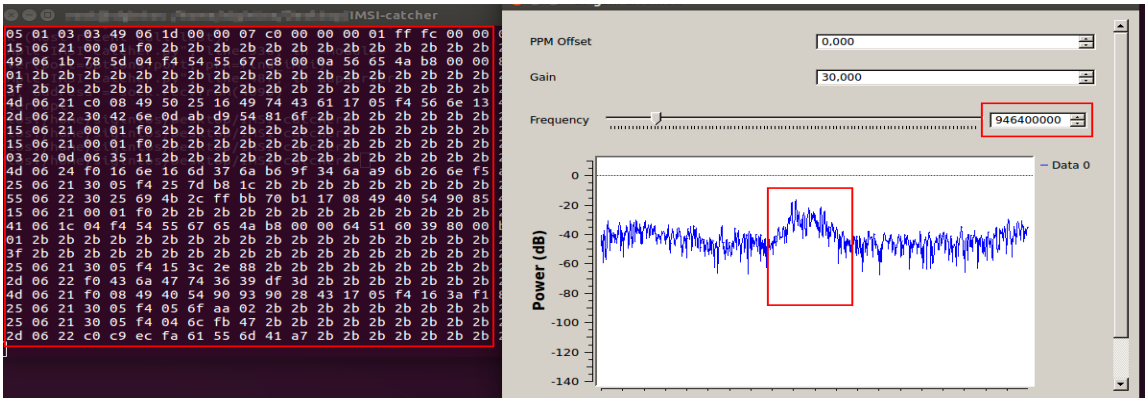
```
/IMSI-catcher# grgsm_scanner -v -b GSM900
linux; GNU C++ version 5.3.1 20151219; Boost_105800; UHD_003.009.002-0-unknown


ARFCN:   52, Freq:  945.4M, CID: 308[illegible], LAC: 218[illegible], MCC: 40[illegible], MNC:  45, Pwr: -44
  |---- Configuration: 1 CCCH, not combined
  |---- Cell ARFCNs:
  |---- Neighbour Cells:

ARFCN:   53, Freq:  945.6M, CID:     0, LAC: 218[illegible], MCC: 40[illegible], MNC:  45, Pwr: -45
  |---- Configuration: Unknown
  |---- Cell ARFCNs:
  |---- Neighbour Cells:

ARFCN:   57, Freq:  946.4M, CID:     0, LAC: 218[illegible], MCC: 40[illegible], MNC:  45, Pwr: -27
  |---- Configuration: Unknown
  |---- Cell ARFCNs:
  |---- Neighbour Cells:
```

Three base stations were found. The signal mentioned above was relatively strong with a frequency of 945.4MHz and 945.6MHz.

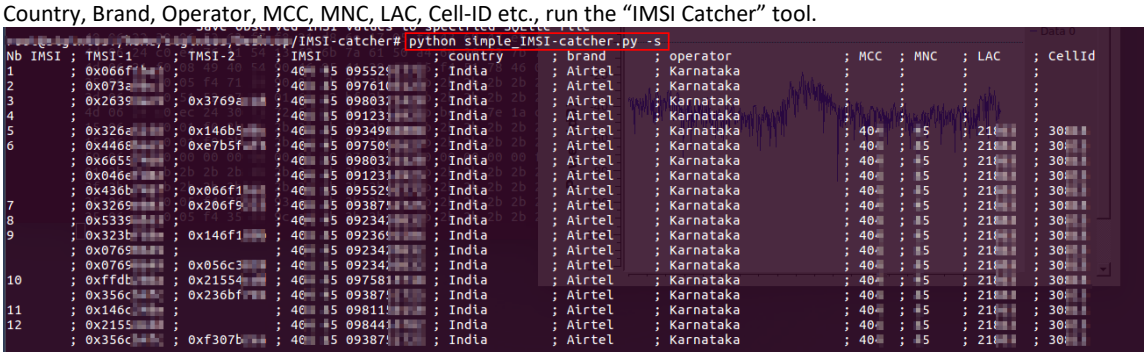
- In the above manner, we obtained some parameter information of the base station, such as: center frequency, channel, ARFCN value, LAC, MCC, MNC value, etc.
- With the above details, we want to sniff the base station frequency. For that the program called "**grgsm_livemon**" will be used.
- Run the "**Wireshark**" before running the "**grgsm_livemon**" tool to capture the packets. Select any interface to capture all the data.
- Once the sniffing of the frequency starts, a popup window appears, as shown in the screenshot below.



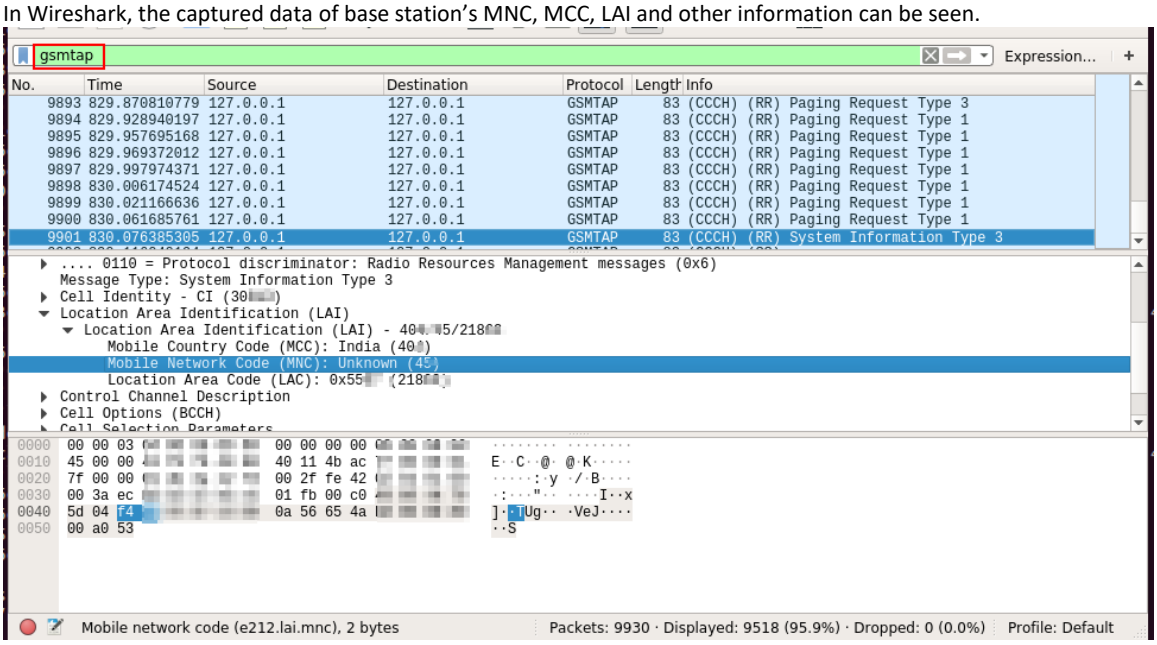
- The frequency button needs to be moved in order to capture the frequency. Once data capture starts it will look like the screenshot below.



- Now we need to capture the IMSI details with the help of an “IMSI Catcher” tool. To capture the IMSI and other details like TMSI, Country, Brand, Operator, MCC, MNC, LAC, Cell-ID etc., run the “IMSI Catcher” tool.



- In Wireshark, the captured data of base station’s MNC, MCC, LAI and other information can be seen.



3.3 Rogue BTS (BladeRF2.0 + YateBTS software)

Rogue BTS is the use of a software-defined radio (SDR) to create a fake cell tower and a software implementation of a GSM/GPRS radio access network. The software typically used to power rogue BTS' is **YateBTS**, which supports GSM850, EGSM900, DCS1800, PCS1900 GSM bands.

The instructions in this resume are for the installation and setup of the BladeRF 2.0 Micro. The setup uses the 2.0 Micro (A9) model. The BladeRF X40, the predecessor to the BladeRF 2.0 Micro supported 300 MHz to 3.8 GHz while the 2.0 Micro supports 47 MHz to 6 GHz.

Note: the author and the maker Nick used an Ubuntu 18.04 as a system.

3.3.1 Setup

- **Step 1: Update/upgrade your fresh installation of your linux system.**

```
$ apt update ; apt upgrade
```

- **Step 2: Add BladeRF PPA and install BladeRF tools and libbladeRF**

```
$ add-apt-repository papa:nuand/bladerf
$ apt update
$ apt install libbladerf-dev
```

- **Step 3: Add user/group permissions for non-root user**

```
$ addgroup yate
$ usermod -a -G yate alissaknight
$ apt install libusb-1.0-0-dev
```

- **Step 4: Download the custom Yate distro created by Nuand**

```
$ wget https://www.nuand.com/downloads/yate-rc.tar
$ tar xvf yate-rc.tar
$ mv yate /usr/src
$ mv yatebts /usr/src
$ mv *.rbf /usr/share/nuand/bladeRF
$ apt install autoconf gcc g++ make
```

- **Step 5: Compile Yate**

```
$ cd /usr/src/yate
$ ./autogen.sh ; ./configure --prefix=/usr/local ; make ; make install-noapi ; ldconfig
```

- **Step 6: Compile YateBTS**

```
$ cd /usr/src/yatebts
$ ./autogen.sh
$ ./configure --prefix=/usr/local
$ make
$ sudo make install
$ sudo ldconfig
```

- **Step 7: Set permissions**

```
$ touch /usr/local/etc/yate/snmp data.conf /usr/local/etc/yate/tmsidata.conf
$ chown root:yate /usr/local/etc/yate/*.conf
$ chmod g+w /usr/local/etc/yate/*.conf
```

- **Step 8: Set transceiver scheduling**

```
$ vi /usr/local/etc/ybts.conf
# Add the values below to the ybts.conf file
radio read priority=highest
radio_send_priority=high
```

- **Step 9: Install Apache2 and PHP**

```
$ apt install apache2
$ add-apt-repository ppa:ondrej/php
$ apt update
$ apt install php5.6
```

- **Step 10: Install Network-in-a-PC**

```
$ cd /var/www/html
$ ln -s /usr/local/etc/yate/nipc_web nipc
$ chmod -R a+rw /usr/local/etc/yate
$ /etc/init.d/apache2 start
```

- **Step 11: Connect to NIPC with your web browser and configure MCC, MNC, and Band for your BTS.**
  NOTE: To determine what values to use here, select a wireless network to act as a decoy (E.g. AT&T Wireless).
  If your mobile phone is connected to the wireless network you want to imitate a BTS for, you can place your phone into field test mode.

- **Here is the code you need to dial for an iPhone and Android:**
  1. Push the call button to make a phone call
  2. Dial *3001#12345#*
  3. Push the Call button
  4. Push Serving Cell Info
- **freq_band_ind (Frequency Band Indicator):**

  ✓ **4G :**

  700 MHz Lower B/C, Band 12/17 (LTE). 850 MHz Cellular, Band 5 (LTE).
  1700/ 2100 MHz AWS, Band 4 (LTE).
  1900 MHz PCS, Band 2 (LTE).
  2300 MHz WCS, Band 30 (LTE).

  ✓ **5G:**

  850 MHz, 24 GHz, 39 GHz (Band n260).

- In our case, the freq_band_ind is 2. Because we using AT&T Wireless, the frequency for Band 2 would be LTE, 1900 MHz PCS. The **sel_plmn_mcc:310, sel_plmn_mnc:410.**

  This matches up with what mcc-mnc.com says for our carrier: the freq_band_ind is currently 2. So based on the above bands for AT&T Wireless, the phone is operating currently at 1900 MHz PCS over 4G/LTE.

30

Not Secure — mcc-mnc.com

or HNI of the world.

Mcc-mnc.com is a service by SMScarrier.EU and powered by interactive digital media GmbH

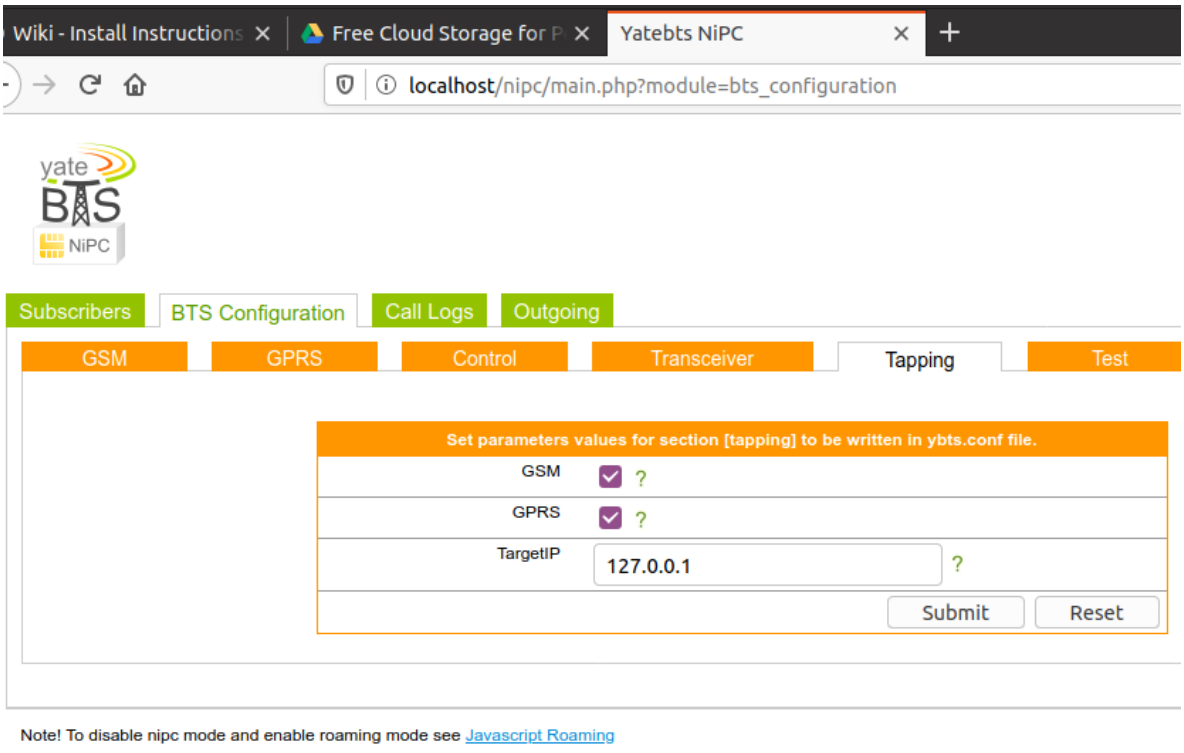
Show 25 entries Search: at&t

MCC	MNC	ISO	Country	Country Code	Network
302	370	ca	Canada	1	FIDO (Rogers AT&T/ Microcell)
302	720	ca	Canada	1	Rogers AT&T Wireless
334	50	mx	Mexico	52	AT&T/IUSACell
334	050	mx	Mexico	52	AT&T/IUSACell
334	040	mx	Mexico	52	AT&T/IUSACell
334	04	mx	Mexico	52	AT&T/IUSACell
310	150	us	United States	1	AT&T Wireless Inc.
310	680	us	United States	1	AT&T Wireless Inc.
310	070	us	United States	1	AT&T Wireless Inc.
310	560	us	United States	1	AT&T Wireless Inc.
310	410	us	United States	1	AT&T Wireless Inc.
310	380	us	United States	1	AT&T Wireless Inc.
310	170	us	United States	1	AT&T Wireless Inc.
310	980	us	United States	1	AT&T Wireless Inc.
310	38	us	United States	1	USA 3650 AT&T

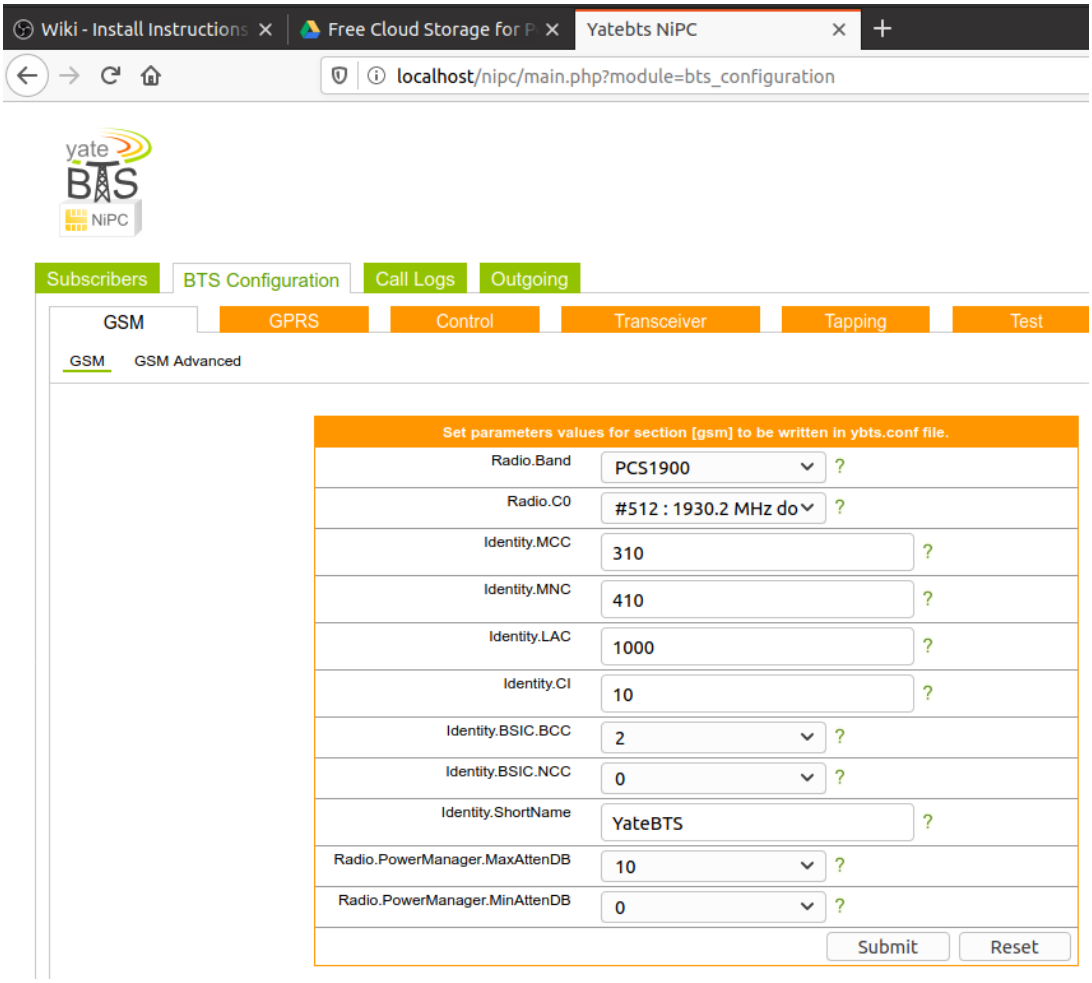
Showing 1 to 15 of 15 entries (filtered from 1,690 total entries)

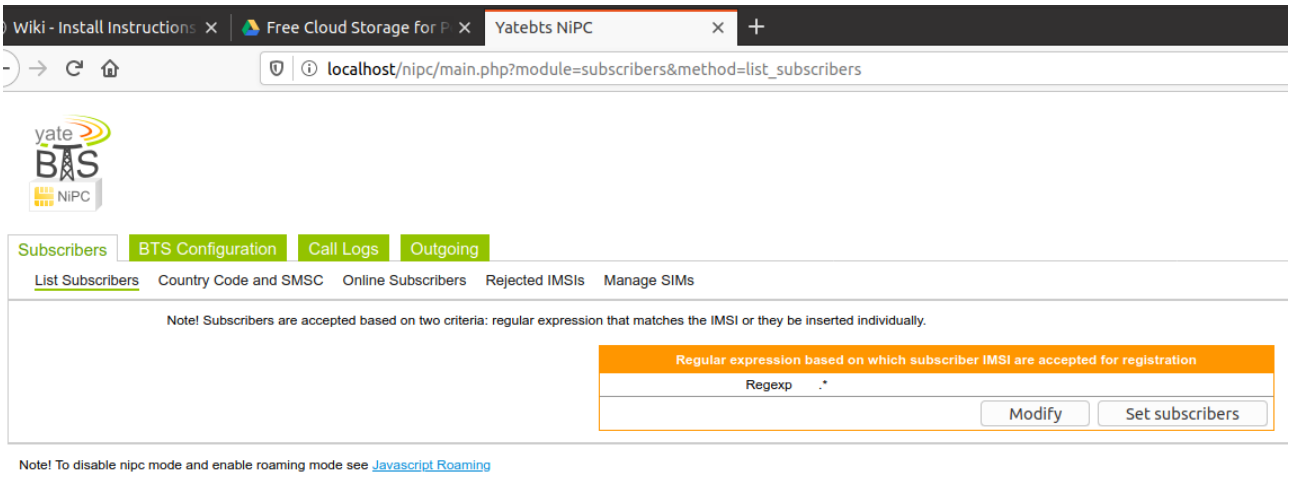
First Previous 1 Next Last

Once you have the appropriate values to plug into YateBTS, you'll want to enter them into the following screens before starting Yate. Below is the configuration from our side of the world.



RADIO MOBILE HACKING





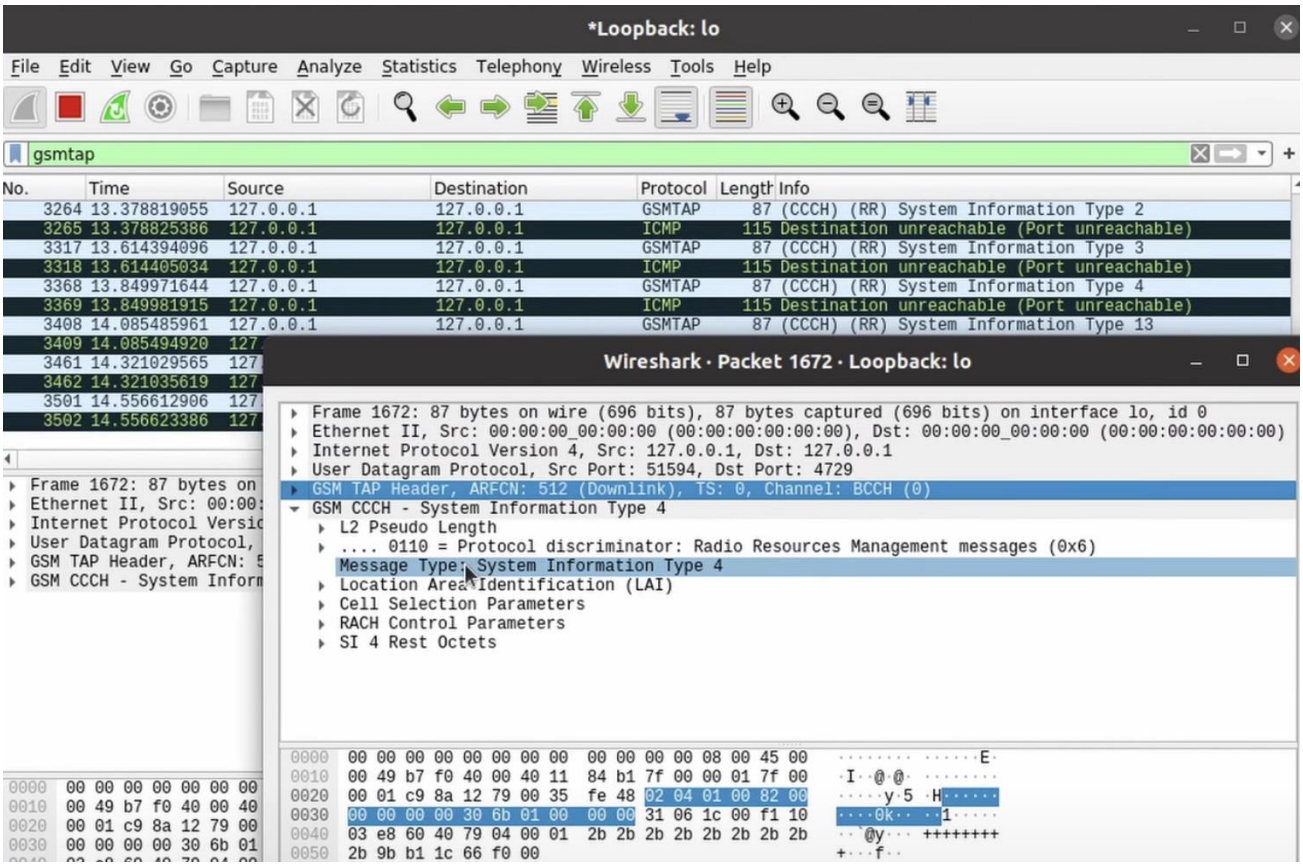
- **Step 12: Plug in the BladeRF to the USB cable and laptop and soft load the FPGA**

```
$ bladeRF-cli -l /usr/src/Nuand/bladeRF/hostedxA9.rbf (or whatever FPGA file matches your board)
```

- **Step 13: Start Yate**

```
$ sudo yate -v
```

- Once Yate has been started, you should be able to start Wireshark and point it at your local loopback interface in order to see the GSM traffic flowing across your BTS.



3.3.2 Extra info

NickvsNetworking has done an amazing job for the installation I also what to mention here to:

Configuring YateBTS for Software Defined GSM/GPRS

https://nickvsnetworking.com/connecting-any-3rd-party-hss-to-open5gs-mme/

Compiling YateBTS NIPC for Software Defined GSM / GPRS

https://nickvsnetworking.com/compiling-yatebts-nipc-for-software-defined-gsm-gprs/

3.4 Create 2G network with OsmocomBB

OsmocomBB is a Free Software / Open Source GSM Baseband software implementation. It intends to completely replace the need for a proprietary GSM baseband software, such as:

- drivers for the GSM analog and digital baseband
- drivers for (integrated and external) peripherals
- the GSM phone-side protocol stack, from layer 1 up to layer 3

In short: by using OsmocomBB on a compatible phone, you will be able to make and receive phone calls, send and receive SMS, etc. based on Free Software only.

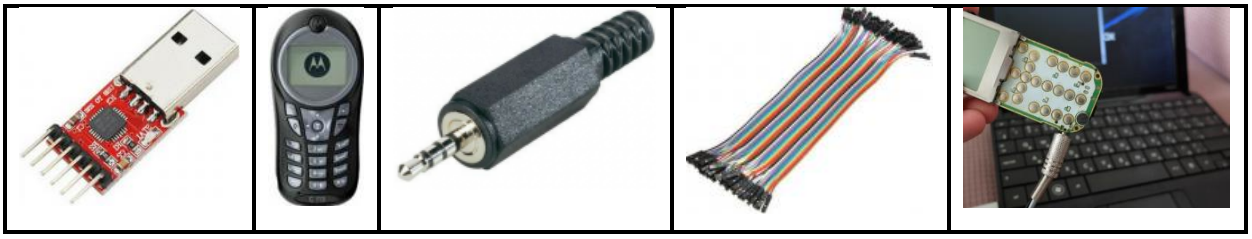
Osmocom-BB is not an user oriented project (yet?). It's mainly targeted at developers / hackers / researchers that want to learn more and play with GSM. As such, there are a few things we expect from you.

3.4.1 Hardware setup

- **Hardware**

- Computer with 32 bit Ubuntu 14.04 installed (not a virtual). Recommend using Ubuntu 14.04, which is the 32-bit version. Perhaps we can install everything on 64-bit Ubuntu 16.04, but then we will have to solve all the problems with the dependencies when installing and compatibility with the Osmocom project branches.
- 2 phones on the TI Calypso chipset (Motorola c113, c118, c123, …)
- 2 wires (2.5 mm jack + jumper)
- 2 USB-TTL converter that can work on CP2102, FT232 or PL2303 chips.
  The author recommend using CP2102 because using a specialized utility you can make this converter work at non-standard speeds, which is required for some branches of OsmocomBB.

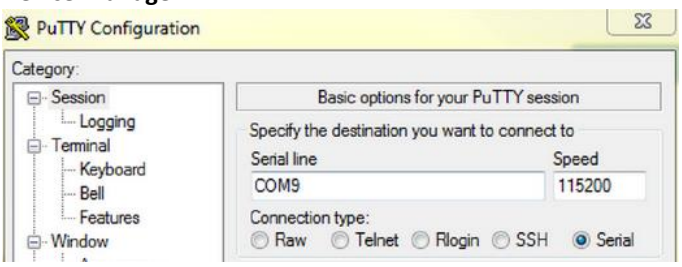
  ⇨ Connect the Tx, Rx, GND outputs of the converter to the jack contacts as follows:
  TxD connect to the tip of the jack
  RxD connect to the middle jack contact
  Connect GND to the bottom of the jack.



- **Unclear problem**

- After purchasing a jack, make sure that it can be inserted into the headset jack on the end. Otherwise, you may receive errors due to an unreliable connection to the phone or not at all.
- Jacks, which are most often sold in the shops of radio components, are not inserted into the socket until the end. They are hindered by the body (of their own phone).
- To make sure that the jack comes through, you can get the phone out of the case and try to insert the jack

- **You can check the reliability of the connection using PuTTY. You can find out the COM port number by looking in the Device Manager.**



- Connect the phone to the computer via the USB-TTL converter and the assembled wire, briefly press the power button and in the PuTTY window, the @ftmtoolerror message should appear among other symbols.

3.4.2 Software setup

- **Software**
- Transceivers based on OsmocomBB
- Base station based on OsmoBTS
- Base station controller based on OsmoBSC
- MSC, HLR, SMS center based on OsmoNTIB

- **Install the basic packages that we need to build Osmocom:**

```
apt-get install build-essential libtool libtalloc-dev shtool autoconf automake git-core pkg-config make gcc libpcsclite-dev
```

- **Installing the lib Osmocore library**

```
git clone git://git.osmocom.org/libosmocore.git or https://github.com/osmocom/libosmocore
cd libosmocore/
autoreconf -i
./configure
make
make install
ldconfig -i
```

- **Install the toolchain**

To build programs for the phone, we need a cross-toolchain compiler. The official website now presents a newer version, but with it will not work the old Osmocombb branches that were written under the old compiler.

Of course, you can slightly fix the code and fix the errors that occur when compiling with the new toolchain, but I leave it outside the scope of this article and I recommend using the following script to install the cross-compiler.

The process can take a long time, up to several hours. Be patient and make sure that the file system is free at least 3 GB.

```
git clone https://github.com/axilirator/gnu-arm-installer.git
cd gnu-arm-installer
apt-get install libgmp3-dev libmpfr-dev libx11-6 libx11-dev flex bison libncurses5 libncurses5-dbg libncurses5-dev libncursesw5 libncursesw5-dbg libncursesw5-dev zlibc zlib1g-dev libmpfr4 libmpc-dev texinfo
./download.sh
./build.sh
```

- **After the compilation is completed**, add the path to the executable files in PATH, in my case/root/cosmo com/gnu-arm-installer/install/bin

```
vi /etc/bash.bashrc
# add in the end
export PATH=$PATH:/root/osmocom/gnu-arm-installer/install/bin
```

- **Collecting Osmocombb**

  The master branch takes part in launching the GSM network, but it will be useful if you want to work with other applications, such as RSSI or cell_log (see further in the text). If you want to be able to send anything to the network, you need to uncomment the src/target/firmware/Makefile line:

```
CFLAGS += -DCONFIG_TX_ENABLE
```

- Assemble

```
git clone git://git.osmocom.org/osmocom-bb.git osmocombb
cd osmocombb/src
make
```

- **Installing the FFT package**

```
wget http://www.fftw.org/fftw-3.3.6-pl2.tar.gz
tar -xvzf fftw-3.3.6-pl2.tar.gz
cd fftw-3.3.6-pl2
./configure –enable-threads –enable-float
make
make install
ldconfig
```

- **Installing the lib Osmo-DSP library**

```
git clone git://git.osmocom.org/libosmo-dsp.git
cd libosmo-dsp/
autoreconf -i
./configure
make
make install
ldconfig
```

- **Assembling the OsmocomBB branch for OsmoBTS**

```
git clone git://git.osmocom.org/osmocom-bb.git trx
cd trx/
git checkout jolly/testing
cd src/

# You need to uncomment the target/firmware/Makefile line:
CFLAGS += -DCONFIG_TX_ENABLE

# Compile:
make HOST_layer23_CONFARGS=–enable-transceiver
```

- **Install libdbi for SQLite**

```
apt-get install sqlite3 libsqlite3-dev libsctp-dev

# Download: sourceforge.net/projects/libdbi/files/libdbi/libdbi-0.8.3
tar -xvzf libdbi-0.8.3.tar.gz
cd libdbi-0.8.3
autogen.sh
./configure –disable-docs
make
make install
ldconfig
cd ..
```

```
# Download: sourceforge.net/projects/libdbi-drivers/files/libdbi-drivers/libdbi-drivers-0.8.3
tar -xvzf libdbi-drivers-0.8.3.tar.gz
cd libdbi-drivers-0.8.3

# There is a type in the driver that causes errors during connection to the HLR. We correct it before compilation.
vi drivers/sqlite3/dbd_sqlite3.c
Change _dbi_internal_error_handler to _dbd_internal_error_handler.

# Collect:
./autogen.sh
./configure –disable-docs –with-sqlite3 –with-sqlite3-dir=/usr/bin –with-dbi-incdir=/usr/local/include
make
make install
ldconfig
```

- **Installing ORTP**

```
wget http://download.savannah.gnu.org/releases/linphone/ortp/sources/ortp-0.22.0.tar.gz
tar -xvf ortp-0.22.0.tar.gz
cd ortp-0.22.0/
./autogen.sh
./configure
make
make install
ldconfig
```

- **Installing the libosmo-abis library**

```
git clone git://git.osmocom.org/libosmo-abis.git
cd libosmo-abis
autoreconf -i
./configure
make
make install
ldconfig
```

- **Installing the libosmo-natif library**

```
git clone git://git.osmocom.org/libosmo-netif.git
cd libosmo-netif
autoreconf -i
./configure
make
make install
ldconfig
```

- **Installing OpenBSC**

```
apt-get install libssl0.9.8 libssl-dev
ldconfig
git clone git://git.osmocom.org/openbsc.git
cd openbsc/openbsc/
autoreconf -i
./configure
make
make install
```

- **Installing OsmoBTS**

```
git clone git://git.osmocom.org/osmo-bts.git
cd osmo-bts
autoreconf -i
./configure –enable-trx
make
make install
```

- **Configuration**

```
# We are working with Osmocom from under root, so my configuration files are in /root/.osmocom
mkdir /root/.osmocom;cd /root/.osmocom
touch ~/.osmocom/osmo-bts.cfg
touch ~/.osmocom/open-bsc.cfg
```

- **Then there are two options:**

- Download OsmoNTIB manuals and customize everything yourself
- Instead of empty files, use my own, modified to suit your needs.

- **Our configuration files osmo-btf.cfg and open-bsc.cfg are at the end of the article.**

- We deliberately removed the band for the GSM band and ARFCN from the files.

ARFCN is the radio channel on which your base station will operate. A suitable ARFCN can be found using the RSSI program, the OsmocomBB package, or using the cell_log tool.

Remember that the signal from your base station should not interfere with the signals of commercial GSM networks. Depending on which channel you use, select band.

In order to securely limit the signal from your base station, you can build a Faraday Cage.

Without adding ARFCN and band to our configuration files, OsmoNTIB will not start.

3.4.3 Launching

- **We connect both phones to the computer and check their availability.**

```
ls -l /dev/ttyUSB*
```

You should see ttyUSB0 and ttyUSB1. Next, each command must be executed in a separate terminal.

In the Osmocon syntax, you can have differences. For example, in your case, there may be compal_e86 or e87 and not c123xor, but something else.

- **Initialize the first transceiver**

```
cd /root/osmocom/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
```

- Press the power button of the phone that was connected first. After the download is complete, you will see TRX on the phone screen.

- **Initialize the second transceiver**

```
cd /root/osmocom/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
```

- Press the power button of the phone, which was connected by the second. After the download is complete, you will see TRX on the phone screen.

- **Set up transceivers to follow the timer of commercial BTS**

```
cd /root/osmocom/trx/src/host/layer23/src/transceiver/
./transceiver -a ARFCN -2 -r 99
```

- Instead of ARFCN, you must specify the channel number on which the commercial base station operates with a good signal. Again, can be found using RSSI or cell_log.

- **Launch MSC, HLR and SMS Center**

```
cd /root/.osmocom
osmo-nitb -c ~/.osmocom/open-bsc.cfg -l ~/.osmocom/hlr.sqlite3 -P -C –debug=DRLL:DCC:DMM:DRR:DRSL:DNM
```

- **We start the base station**

```
cd /root/.osmocom
osmo-bts-trx –debug DRSL:DOML:DLAPDM -r 99
```

- All components of GSM network should now be in working order and you are ready to become the first subscriber!

3.4.4 Testing

Now you can connect to the network from any cell phone by selecting it in manual mode. The network is displayed as 00101 or TestNet. The network may not be on the first try.

If something went wrong during the connection, turn on the air mode, turn it off and try again to connect to the network. After connecting, you can find your number using USSD code * # 100 #.

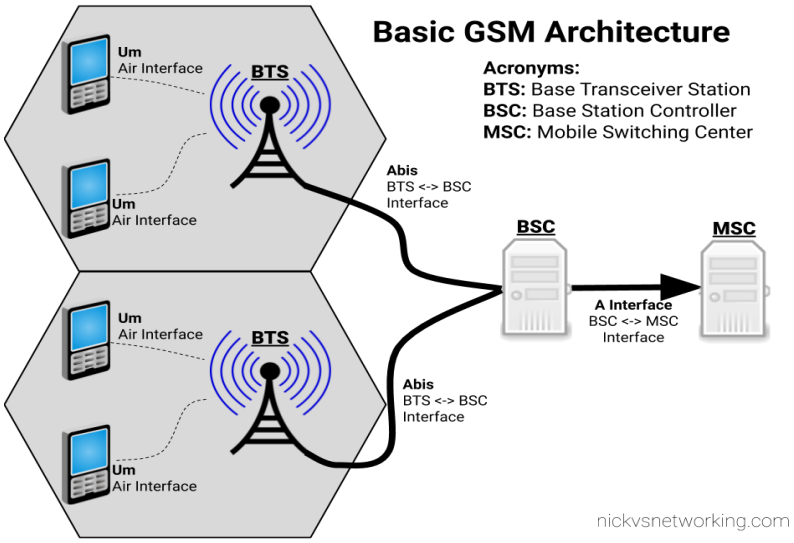
- You can connect to the OsmoNTIB console in this way:telnet localhost 4242
- Connect to the OsmoBTS console as follows:telnet localhost 4241
- **Configuration files:**

[su_spoiler title=”osmo-bts.cfg” open=”yes” style=”fancy” icon=”arrow”]! ! OsmoBTS (0.4.0.433-8913) configuration saved from vty !!! ! log stderr logging filter all 1 logging color 1 logging print category 0 logging timestamp 0 logging level all everything logging level rsl info logging level oml info logging level rll notice logging level rr notice logging level meas notice logging level pag info logging level l1c info logging level l1p info logging level dsp debug logging level pcu notice logging level ho notice logging level trx notice logging level loop notice logging level abis notice logging level rtp notice logging level sum notice logging level lglobal notice logging level llapd notice logging level linp notice logging level lmux notice logging level lmi notice logging level lmib notice logging level lsms notice logging level lctrl notice logging level lgtp notice logging level lstats notice logging level lgsup notice logging level loap notice logging level lss7 notice logging level lsccp notice logging level lsua notice logging level lm3ua notice log file OsmoBTS.log logging filter all 0 logging color 1 logging print category 0 logging timestamp 1 logging level all everything logging level rsl info logging level oml info logging level rll notice logging level rr notice logging level meas notice logging level pag info logging level l1c info logging level l1p info logging level dsp debug logging level pcu notice logging level ho notice logging level trx notice logging level loop notice logging level abis notice logging level rtp notice logging level sum notice logging level lglobal notice logging level llapd notice logging level linp notice logging level lmux notice logging level lmi notice logging level lmib notice logging level lsms notice logging level lctrl notice logging level lgtp notice logging level lstats notice logging level lgsup notice logging level loap notice logging level lss7 notice logging level lsccp notice logging level lsua notice logging level lm3ua notice ! line vty no login ! e1_input e1_line 0 driver ipa e1_line 0 port 0 no e1_line 0 keepalive phy 0 osmotrx ip 127.0.0.1 osmotrx fn-advance 30 osmotrx rts-advance 5 instance 0 bts 0 band [ЗАДАТЬ GSM900 ИЛИ DCS1800] ipa unit-id 1801 0 oml remote-ip 127.0.0.1 rtp jitter-buffer 0 paging queue-size 200 paging lifetime 0 uplink-power-target -75 min-qual-rach 50 min-qual-norm -5 ms-power-loop -65 timing-advance-loop setbsic trx 0 power-ramp max-initial 0 mdBm power-ramp step-size 2000 mdB power-ramp step-interval 1 ms-power-control dsp phy 0 instance 0[/su_spoiler]

[su_spoiler title=”open-bsc.cfg” open=”yes” style=”fancy” icon=”arrow”]! ! OpenBSC (0.15.0.763-5121) configuration saved from vty !!! ! log stderr logging filter all 1 logging color 1 logging print category 0 logging timestamp 0 logging level all everything logging level rll everything logging level cc everything logging level mm everything logging level rr everything logging level rsl everything logging level nm everything logging level mncc notice logging level pag notice

logging level meas notice logging level sccp notice logging level msc notice logging level mgcp notice logging level ho notice logging level db notice logging level ref notice logging level gprs debug logging level ns info logging level bssgp debug logging level llc debug logging level sndcp debug logging level nat notice logging level ctrl notice logging level smpp debug logging level filter debug logging level ranap debug logging level sua debug logging level lglobal notice logging level llapd notice logging level linp notice logging level lmux notice logging level lmi notice logging level lmib notice logging level lsms notice logging level lctrl notice logging level lgtp notice logging level lstats notice logging level lgsup notice logging level loap notice logging level lss7 notice logging level lsccp notice logging level lsua notice logging level lm3ua notice log file OsmoBSC.log logging filter all 0 logging color 1 logging print category 0 logging timestamp 1 logging level all info logging level rll notice logging level cc notice logging level mm notice logging level rr notice logging level rsl notice logging level nm info logging level mncc notice logging level pag notice logging level meas notice logging level sccp notice logging level msc notice logging level mgcp notice logging level ho notice logging level db notice logging level ref notice logging level gprs debug logging level ns info logging level bssgp debug logging level llc debug logging level sndcp debug logging level nat notice logging level ctrl notice logging level smpp debug logging level filter debug logging level ranap debug logging level sua debug logging level lglobal notice logging level llapd notice logging level linp notice logging level lmux notice logging level lmi notice logging level lmib notice logging level lsms notice logging level lctrl notice logging level lgtp notice logging level lstats notice logging level lgsup notice logging level loap notice logging level lss7 notice logging level lsccp notice logging level lsua notice logging level lm3ua notice ! stats interval 5 ! line vty no login ! e1_input e1_line 0 driver ipa e1_line 0 port 0 no e1_line 0 keepalive network network country code 1 mobile network code 1 short name TestNet long name TestNet auth policy accept-all authorized-regexp .* location updating reject cause 13 encryption a5 0 neci 1 paging any use tch 0 rrlp mode none mm info 1 handover 0 handover window rxlev averaging 10 handover window rxqual averaging 1 handover window rxlev neighbor averaging 10 handover power budget interval 6 handover power budget hysteresis 3 handover maximum distance 9999 timer t3101 10 timer t3103 0 timer t3105 40 timer t3107 0 timer t3109 0 timer t3111 0 timer t3113 60 timer t3115 0 timer t3117 0 timer t3119 0 timer t3122 10 timer t3141 0 dyn_ts_allow_tch_f 0 subscriber-keep-in-ram 0 bts 0 type sysmobts description calypso band DCS1800 cell_identity 0 location_area_code 1 base_station_id_code 63 ms max power 30 cell reselection hysteresis 4 rxlev access min 0 periodic location update 30 radio-link-timeout 32 channel allocator ascending rach tx integer 9 rach max transmission 7 channel-descrption attach 1 channel-descrption bs-pa-mfrms 5 channel-descrption bs-ag-blks-res 1 early-classmark-sending forbidden ip.access unit_id 1801 0 oml ip.access stream_id 255 line 0 neighbor-list mode automatic codec-support fr amr amr tch-h modes 0 amr tch-h start-mode 1 gprs mode none no force-combined-si trx 0 rf_locked 0 arfcn[/su_spoiler]

3.5 OsmoBTS



Note: we'll be using a Linux system and trying where possible to use packages from Repos instead of compiling from source.

- **This will get the Osmocom key added to your package manager and the Osmocom sources in apt ready for us to install.**

```
wget https://download.opensuse.org/repositories/network:/osmocom:/latest/Debian_10/Release.key
apt-key add Release.key && rm Release.key
echo "deb https://download.opensuse.org/repositories/network:/osmocom:/latest/xUbuntu 18.04/ ./" >
/etc/apt/sources.list.d/osmocom-latest.list
apt-get update
```

3.5.1 OsmoBTS Install

To get started we'll install a virtual BTS. This **virtual BTS won't simulate the Um (air) interface**, but it will simulate the Abis interface towards the BSC so we can configure this virtual BTS in our BSC.

- **Installation is pretty straightforward:**

```
apt-get install osmo-bts-virtual (if this command not work try: apt-get install osmo-bts)
```

- **By default Osmocom software runs as a daemon in *systemctl*, we'll disable and stop this behavior for now so we can better understand it running in the foreground:**

```
systemctl stop osmo-bts-virtual
systemctl disable osmo-bts-virtual
```

3.5.2 OsmoBTS Config – Text Files

If you have a look in /etc/osmocom/ you'll see *.cfg* files that contain our config in text files. But that's not the only way (or even the recommended way) that we'll put together the config for Osmocom software, but we'll get started by editing the config file manually.

- **We'll start by setting a Unit ID of the BTS and setting the IP of the BSC.**

```
cd /etc/osmocom/
vi osmo-bts-virtual.cfg
```

- We'll edit the *oml remote-ip* to point to the IP of the server that will run our BSC:
  If you're planning on running the BTS and BSC on the same machine you can leave it as localhost (127.0.0.1).
- Next up we'll set the Unit-ID of the BTS, this identifies the BTS inside the BSC,
  I'll set it to unit-id 4242 by changing *ipa unit-id 4242 0*

- **Finally we'll change the logging config to show everything by changing it to:**

```
log stderr
logging filter all 1
!
```

- **So that's it in terms of config for our virtual BTS through text files, so we'll save the file and try starting up *osmo-bts-virtual.***

```
osmo-bts-virtual -c osmo-bts-virtual.cfg
```

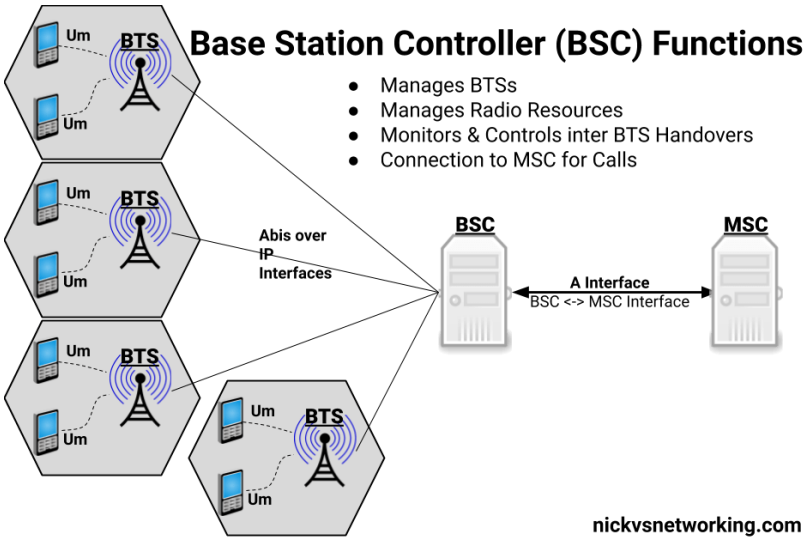
- **You should get a result similar to this:**

```
root@gsm-bts:/etc/osmocom# osmo-bts-virtual -c osmo-bts-virtual.cfg
 ((*))
   |
  / \ OsmoBTS
<0010> telnet interface.c:104 Available via telnet 127.0.0.1 4241
<0012> input/ipaccess.c:901 enabling ipaccess BTS mode, OML connecting to 127.0.0.1:3002
<000d> abis.c:142 Signalling link down
<0001> bts.c:292 Shutting down BTS 0, Reason Abis close
Shutdown timer expired

root@gsm-bts:/etc/osmocom#
```

- **So what are we seeing here?**
  Well *Osmo-BTS-Virtual* is trying to bring up it's Abis interface but it's not getting a connection to the the BSC (We haven't set one up yet). No connection to a BSC means the BTS won't go on the air as it doesn't have any processing for itself, so it eventually times out and shuts down.

- **In the next chapter we'll pick up adding our Virtual BTS to the Virtual BSC but first we need to creat the Virtual BSC which is in our case OsmoBSC**

3.6 OsmoBSC



So in our last post we finished setting up a **Base Transceiver Station (BTS)** but it's no use unless it can home itself to a **Base Station Controller (BSC)**.

- **So what does a BSC do?**
- The BSC acts as a central controller for one or more BTS.
- In practice this means the BSC configures most of the parameters on the BTS and brings each one up onto the air when they're ready.
- The BSC monitors measurements from users to work out when to hand off from one BTS to neighboring BTS,
- The BSC also handles the allocation of radio channels and radio resources across the BTSs it manages.
- In short, it does pretty much everything radio related for the BTS except transmitting and receiving data over the Air (Um) interface.
- As well as managing the BTS under it, the other other equally important role of the BSC is to provide connectivity to the rest of the GSM network, by connecting to a **Mobile Switching Center (MSC)** which handles calls to and from our mobile subscribers and authenticating them.

  By acting as a funnel of sorts, the MSC only needs a connection to each BSC instead of to each BTS (Which would be an impractically large number of connections)

3.6.1 OsmoBSC Installation

Osmocom have their own BSC – Aptly called Osmo-BSC.

- **Installation is pretty straightforward, assuage you've got the Osmocom repo in your sources list:**

```
apt-get install osmo-bsc
systemctl stop osmo-bsc
```

- In order to serve the BTSs it controls, Osmo-BSC relies on connectivity to a Mobile Switching Center (MSC), which in turn connects to a HLR (Home Location Register). The BSC and MSC communicate via SS7, and the routing is done by a Signal Transfer Points (STP).

- **In order to bring our BSC up in a useful way, we'll need to install and start these applications.**

```
apt-get install osmo-stp osmo-msc osmo-hlr
systemctl start osmo-stp
systemctl start osmo-msc
systemctl start osmo-hlr
```

3.6.2 OsmoBSC Config – Telnet Interactive Terminal

So now we've got the BSC installed we've pointed our BTS at the IP of the BSC, we'll need to get *osmo-bsc* running and add the config for our new BTS.

Instead of working with the text file we'll start the service and work on it through Telnet, like we would for many common network devices.

- **Osmo-BSC listens on port 4242, so we'll start Osmo-BSC and connect to it via Telnet:**

```
systemctl start osmo-bsc
telnet localhost 4242
```

- **We'll start by enabling logging so we can get an idea of what's going on:**

```
OsmoBSC> enable
OsmoBSC# logging enable
OsmoBSC# logging filter all 1
OsmoBSC# logging color 1
```

- **Next up in a new terminal / SSH session, we'll run the OsmoBTS again;**

```
osmo-bts-virtual -c osmo-bts-virtual.cfg
```

- **This time we'll get a different output from the BTS when we try to start it:**

```
root@gsm-bts:/etc/osmocom# osmo-bts-virtual -c osmo-bts-virtual.cfg
 ((*))
   |
  / \ OsmoBTS
<0010> telnet_interface.c:104 Available via telnet 127.0.0.1 4241
<0012> input/ipaccess.c:901 enabling ipaccess BTS mode, OML connecting to 127.0.0.1:3002
<0012> input/ipa.c:128 127.0.0.1:3002 connection done
<0012> input/ipaccess.c:724 received ID_GET for unit ID 4242/0/0
<0012> input/ipa.c:63 127.0.0.1:3002 lost connection with server
<000d> abis.c:142 Signalling link down
<000d> abis.c:156 OML link was closed early within 0 seconds. If this situation persists, please
check your BTS and BSC configuration files for errors. A common error is a mismatch between unit_id
configuration parameters of BTS and BSC.

root@gsm-bts:/etc/osmocom#
```

- **We'll also see errors in the terminal on the BSC too:**

```
<0016> input/ipa.c:287 0.0.0.0:3002 accept()ed new link from 10.0.1.252:42867
<0016> bts_ipaccess_nanobts.c:480 Unable to find BTS configuration for 4242/0/0,
 disconnecting
<0016> input/ipaccess.c:165 Unable to set signal link, closing socket.
```

- **So what's happening here?**

  Well our virtual BTS is trying to connect to our BSC, and this time it's able to, but our BSC doesn't have any config in place for that BTS, so the BSC has rejected the connection.

  So now we've got to configure the BSC to recognize our BTS (Provisioning a new OsmoBTS in the OsmoBSC)

3.6.3 Provisioning a new OsmoBTS in the OsmoBSC

So as to keep this tutorial generic enough for anyone to follow along, we're first going to configure a virtual BTS in our BSC to begin with.

- **We can get the information about the rejected BTS connection attempt from the BSC terminal:**

```
OsmoBSC# show rejected-bts
Date                    Site      ID BTS ID IP
2020-03-29 01:32:37     4242      0         10.0.1.252
```

- **So we know the Site-ID is 4242 (we set it earlier) and the BTS ID for that site is 0, so let's create a BTS in the BSC;**

```
OsmoBSC> enable
OsmoBSC# configure terminal
OsmoBSC(config)# network
OsmoBSC(config-net)# bts 1
OsmoBSC(config-net-bts)# type sysmobts
OsmoBSC(config-net-bts)# description "Virtual BTS"
OsmoBSC(config-net-bts)# ipa unit-id 4242 0
OsmoBSC(config-net-bts)# band DCS1800
OsmoBSC(config-net-bts)# codec-support fr hr efr amr
OsmoBSC(config-net-bts)# cell identity 4242
OsmoBSC(config-net-bts)# location_area_code 4242
OsmoBSC(config-net-bts)# base station id code 4242
OsmoBSC(config-net-bts)# base_station_id_code 42
OsmoBSC(config-net-bts)# ms max power 40
OsmoBSC(config-net-bts)# trx 0
OsmoBSC(config-net-bts-trx)# max power red 20
OsmoBSC(config-net-bts-trx)# arfcn 875
OsmoBSC(config-net-bts-trx)# timeslot 0
OsmoBSC(config-net-bts-trx-ts)# phys_chan_config CCCH+SDCCH4
OsmoBSC(config-net-bts-trx-ts)# exit
OsmoBSC(config-net-bts-trx)# timeslot 1
OsmoBSC(config-net-bts-trx-ts)# phys chan config TCH/F
OsmoBSC(config-net-bts-trx-ts)# exit
OsmoBSC(config-net-bts-trx)# timeslot 2
OsmoBSC(config-net-bts-trx-ts)# phys_chan_config TCH/F
OsmoBSC(config-net-bts-trx-ts)# exit
OsmoBSC(config-net-bts-trx)# timeslot 3
OsmoBSC(config-net-bts-trx-ts)# phys_chan_config TCH/F
OsmoBSC(config-net-bts-trx-ts)# exit
OsmoBSC(config-net-bts-trx)# timeslot 4
OsmoBSC(config-net-bts-trx-ts)# phys_chan_config TCH/F
OsmoBSC(config-net-bts-trx-ts)# exit
OsmoBSC(config-net-bts-trx)# timeslot 5
OsmoBSC(config-net-bts-trx-ts)# phys_chan_config TCH/F
OsmoBSC(config-net-bts-trx-ts)# exit
OsmoBSC(config-net-bts-trx)# timeslot 6
OsmoBSC(config-net-bts-trx-ts)# phys_chan_config TCH/F
OsmoBSC(config-net-bts-trx-ts)# exit
OsmoBSC(config-net-bts-trx)# timeslot 7
OsmoBSC(config-net-bts-trx-ts)# phys_chan_config TCH/F
OsmoBSC(config-net-bts-trx-ts)# exit
OsmoBSC(config-net-bts-trx)# exit
OsmoBSC(config-net-bts)# exit
OsmoBSC(config-net)# exit
OsmoBSC(config)# exit
OsmoBSC# copy running-config startup-config
```

- **So what did we actually do here?**

  Well as we're getting the majority of the smarts for the BTS from the BSC, we've got to tell the BSC all about how we want the BTS setup.

  The type, **IPA Unit ID**, band and Cell Identity make up some of the parameters we need to identify the BTS (IPA Unit ID) and give it it's basic identity parameters.

  - Next up in the ***trx 0*** section we set the contents of the 8 GSM timeslots:
    - Our first time slot we configure as *CCCH+SDCCH4* meaning the first timeslot will contain the *Common Control Channel* and 4 *Standalone dedicated control channels*, used for signaling,
    - While the reamining 7 timeslots will be used with traffic channels for full-rate speech (TCH/F).

  - It's important that what we tell the BSC the capabilities of the BTS are match the actual capabilities of the BTS. For example there's no point configuring GPRS or EDGE support on the BSC if the BTS doesn't support it.

  - If you've got logging enabled when the BTS connects to the BSC you'll see errors listing the features mismatch between the two.

3.6.4 Connecting the OsmoBTS to the OsmoBSC

So let's go ahead and connect our BTS to the BSC.

- **If you've closed the BSC terminal since we enabled logging you'll need to enable it again:**

```
OsmoBSC> logging enable
OsmoBSC> logging filter all 1
OsmoBSC> logging color 1
```

- **And next up we'll try and start the BTS again:**

```
root@gsm-bts:/etc/osmocom# osmo-bts-virtual -c osmo-bts-virtual.cfg
 ((*))
   |
  / \ OsmoBTS
<0010> telnet_interface.c:104 Available via telnet 127.0.0.1 4241
<0012> input/ipaccess.c:901 enabling ipaccess BTS mode, OML connecting to 127.0.0.1:3002
<0012> input/ipa.c:128 127.0.0.1:3002 connection done
<0012> input/ipaccess.c:724 received ID_GET for unit ID 4242/0/0
```

- **And on the BSC you'll see roughly the same thing:**

```
OsmoBSC#
<0016> input/ipa.c:287 0.0.0.0:3002 accept()ed new link from 127.0.0.1:40193
<0004> abis_nm.c:490 BTS1 reported variant: omso-bts-virtual
<0004> abis_nm.c:578 OC=BASEBAND-TRANSCEIVER(04) INST=(00,00,ff): BTS1: ARI reported sw[0/1]:
TRX PHY VERSION is Unknown
<0016> input/ipa.c:287 0.0.0.0:3003 accept()ed new link from 127.0.0.1:44053
<0003> osmo_bsc_main.c:291 bootstrapping RSL for BTS/TRX (1/0) on ARFCN 0 using MCC-MNC 001-01
LAC=4242 CID=4242 BSIC=42
```

- If you've made it this far, congratulations. **Our virtual BTS is now connected to our BSC** – If it wasn't virtual we'd be on the air!

- In the next chapter we'll setup an SDR hardware as a BTS, then provision it on the BSC, and then our cell will be on the air !!

3.7 Connect Rogue BTS (LimeSDR) to OsmoBSC

OsmoBSC accepts Abis over IP connections from a number of different sources:

- There's a list of supported BTS hardware that can talk out of the box to the OsmoBSC, such as the Ericsson RBS series, ip.access nanoBTS, Nokia and Siemens units and even a virtual BTS so you can simulate the connections.

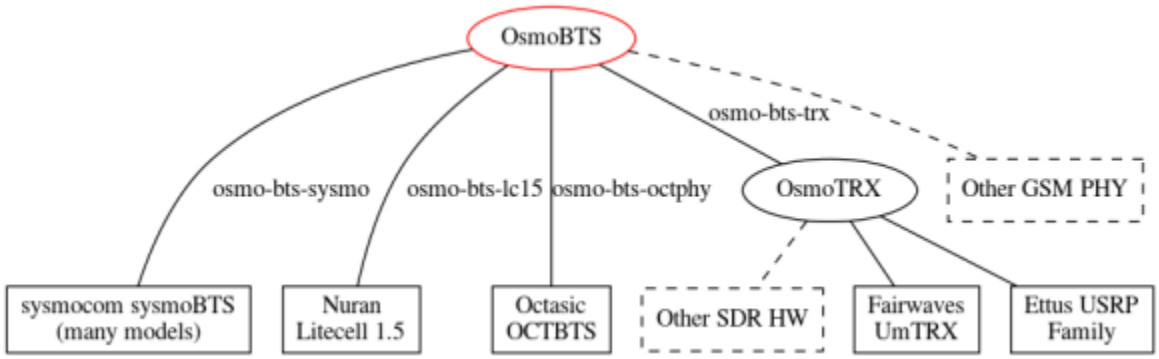
- If you're using any of these premade BTS hardware options, or osmo-bts-virtual, you probably just need to setup the basics on your BTS and point it to your BSC, end of story.

- The below chapter will touch on using common SDR hardware to act as our BTS (i.e. **LimeSDR**)

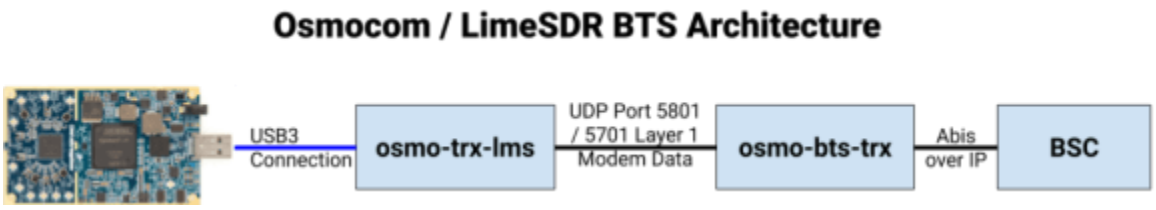
3.7.1 OsmoTRX

In order to bring in a large array of SDR hardware, Osmocom have introduced ***OsmoTRX***, which handles the Layer 1 physical layer of the BTS, and connects to *OsmoBTS* which serves as the BTS and talks Abis over IP to the MSC.

- **Certain hardware can talk directly to *OsmoBTS***, but we're going to rely on *OsmoTRX* to act as the middleman between our SDR hardware and the BTS.



- The above diagram from the Osmocom wiki shows how this fits together with generic SDR platforms, here's how it fits together for us:



- *osmo-trx-lms* will take care of the SDR side of the equation, pretty much serving as a modem and sending everything it gets on the Uu interface to *osmo-bts-trx* over UDP, and everything it receives from *osmo-bts-trx* over UDP it sends out the Uu interface.
- *osmo-bts-trx* will then setup an Abis over IP connection to our BSC.

3.7.2 The LimeSDR

To setup the prerequisites for the LimeSDR to be able to interface with **OsmoTRX**. Osmocom now provide a binary for interfacing with LimeSDR boards directly, instead of having to use the UHD abstraction. This is a much cleaner way of interfacing with the boards and the path I'll be taking.

3.7.3 Software installation

- **LimeSuite provides the drives and utilities for interfacing with the LimeSDR**

```
add-apt-repository -y ppa:myriadrf/drivers
apt-get update
apt-get install limesuite limesuite-udev
```

- **Next we'll connect up the LimeSDR to a USB port, confirm it's there and upgrade its firmware:**

```
LimeUtil --find
```

- **Assuming LimeSDR is hooked up and everything installed we should see an output similar to this:**

```
nick@gsm:~$ sudo LimeUtil --find
  * [LimeSDR-USB, media=USB 3.0, module=FX3, addr=1d50:6108, serial=0[redacted]]
```

- **In which case we can upgrade the LimeSDR firmware with:**

```
LimeUtil --update
```

- **Next we'll start installing Osmocom Sources**

```
wget https://download.opensuse.org/repositories/network:/osmocom:/latest/Debian_10/Release.key
apt-key add Release.key && rm Release.key
echo "deb https://download.opensuse.org/repositories/network:/osmocom:/latest/xUbuntu_18.04/ ./" >
/etc/apt/sources.list.d/osmocom-latest.list
apt-get update
```

- **Now that we've got the Osmocom Debian repos added we can install the packages we need, we're going to install Osmo-BTS-TRX for talking to the BSC over Abis, and install Osmo-TRX-LMS for talking to the SDR.**

```
apt-get install osmo-bts-trx osmo-trx-lms
```

- **After we ve installed the packages, *Osmo-BTS-TRX* will run as a daemon, we'll stop it for now and bring it up manually in the foreground.**

```
systemctl disable osmo-bts-trx
systemctl disable osmo-trx-bts
```

3.7.4 Software Configuration

So now we've got two pieces of the puzzle, it's time to connect the SDR to Osmo-TRX-LMS and connect Osmo-TRX-LMS to Osmo-BTS-TRX.

- **We'll begin by running Osmo-TRX-LMS to connect to the LimeSDR and encapsulate the Uu data into UDP packets we send to Osmo-BTS-TRX.**
- Config files for Osmocom are installed in /etc/osmocom/ so we'll run everything from that directory.

```
osmo-trx-lms -C osmo-trx-lms.cfg
```

- If all was successful we'll see something similar to what we've got below, showing Osmo-TRX-LMS has connected to the SDR and is ready to go.

```
root@gsm:/etc/osmocom# osmo-trx-lms -C osmo-trx-lms.cfg
Info: SSE3 support compiled in and supported by CPU
Info: SSE4.1 support compiled in and supported by CPU
Sat Mar 28 01:36:51 2020 DLGLOBAL <0007> telnet_interface.c:104 Available via telnet 127.0.0.1 4237
Sat Mar 28 01:36:51 2020 DLCTRL <000e> control_if.c:911 CTRL at 127.0.0.1 4236
Sat Mar 28 01:36:51 2020 DMAIN <0000> osmo-trx.cpp:485 [tid=140629064841984] Config Settings
   Log Level............... 0
   Device args.............
   TRX Base Port........... 5700
   TRX Address............. 127.0.0.1
   GSM BTS Address......... 127.0.0.1
   Channels................ 1
   Tx Samples-per-Symbol... 4
   Rx Samples-per-Symbol... 4
   EDGE support............ 0
   Extended RACH support... 0
   Reference............... 0
   Filler Burst Type....... Empty bursts
   Filler Burst TSC........ 0
   Filler Burst RACH Delay. 0
   Multi-Carrier........... 0
   Tuning offset........... 0
   RSSI to dBm offset...... 0
   Swap channels........... 0
   Tx Antennas............. 'BAND1'
   Rx Antennas............. 'LNAW'

Sat Mar 28 01:36:51 2020 DMAIN <0000> osmo-trx.cpp:439 [tid=140629064841984] Setting SCHED_RR priority 18
Sat Mar 28 01:36:51 2020 DDEV <0005> LMSDevice.cpp:54 [tid=140629064841984] creating LMS device...
Sat Mar 28 01:36:51 2020 DDEV <0005> LMSDevice.cpp:143 [tid=140629064841984] Opening LMS device..
Sat Mar 28 01:36:51 2020 DDEV <0005> LMSDevice.cpp:149 [tid=140629064841984] Devices found: 1
Sat Mar 28 01:36:51 2020 DDEV <0005> LMSDevice.cpp:159 [tid=140629064841984] Device [0]: LimeSDR-USB, media=USB 3.0,
 module=FX3, addr=1d50:6108, serial=0
Sat Mar 28 01:36:51 2020 DDEV <0005> LMSDevice.cpp:168 [tid=140629064841984] Using device[0]
Sat Mar 28 01:36:51 2020 DLMS <0006> LMSDevice.cpp:96 [tid=140629064841984] Reference clock 30.72 MHz
Sat Mar 28 01:36:51 2020 DDEV <0005> LMSDevice.cpp:194 [tid=140629064841984] Init LMS device
Sat Mar 28 01:36:52 2020 DDEV <0005> LMSDevice.cpp:207 [tid=140629064841984] Setting Internal clock reference
Sat Mar 28 01:36:52 2020 DLMS <0006> LMSDevice.cpp:96 [tid=140629064841984] Disabling external reference clock
Sat Mar 28 01:36:52 2020 DDEV <0005> LMSDevice.cpp:101 [tid=140629064841984] Sample Rate: Min=100000 Max=6.144e+07 S
tep=0
Sat Mar 28 01:36:52 2020 DDEV <0005> LMSDevice.cpp:230 [tid=140629064841984] Setting sample rate to 1.08333e+06 4
Sat Mar 28 01:36:52 2020 DDEV <0005> LMSDevice.cpp:236 [tid=140629064841984] Sample Rate: Host=1.08333e+06 RF=3.4666
7e+07
Sat Mar 28 01:36:52 2020 DMAIN <0000> LMSDevice.cpp:214 [tid=140629064841984] Antennas configured successfully
Sat Mar 28 01:36:52 2020 DMAIN <0000> osmo-trx.cpp:533 [tid=140629064841984] -- Transceiver active with 1 channel(s)
Sat Mar 28 01:36:52 2020 DMAIN <0000> Threads.cpp:119 [tid=140628992050944] Thread 140628992050944 (task 19410) set
```

- **But if we go scanning the airwaves now, we won't see any data coming out of the SDR's transmitter.**
  That's because Osmo-TRX-LMS needs to connect to Osmo-BTS-TRX,
- **We'll leave Osmo-TRX-LMS running, so let's open up another session and start Osmo-BTS-TRX.**

```
osmo-bts-trx -c osmo-bts-trx.cfg
```

- You'll see for starters that it's Opened our transceiver

```
root@gsm:/etc/osmocom# osmo-bts-trx -c osmo-bts-trx.cfg
((*))
  |
 / \ OsmoBTS
<0017> control_if.c:911 CTRL at 127.0.0.1 4238
<0010> telnet_interface.c:104 Available via telnet 127.0.0.1 4241
<0012> input/ipaccess.c:901 enabling ipaccess BTS mode, OML connecting to 192.168.122.1:3002
<000b> trx_if.c:1174 phy0.0: Open transceiver
```

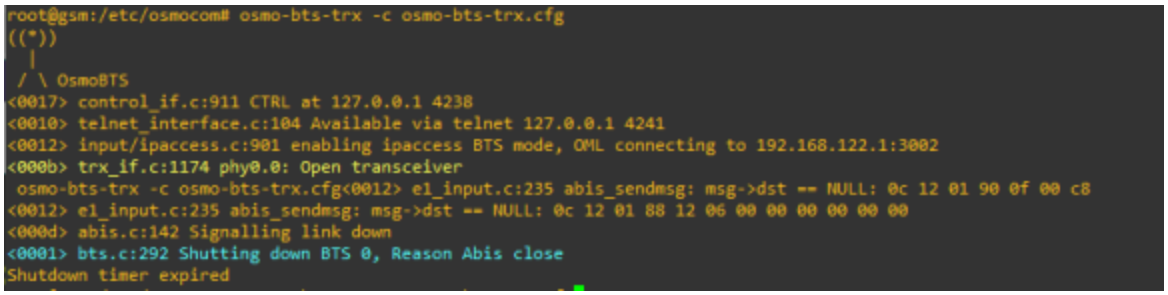
- You'll see this reflected in the Osmo-TRX-LMS stdout, but it'll show the poweroff command has been sent to it, so what gives?

```
Sat Mar 28 02:05:36 2020 DTRXCTRL <0002> Transceiver.cpp:778 [tid=140385354704640][chan=0] command is 'POWEROFF'
Sat Mar 28 02:05:36 2020 DTRXCTRL <0002> Transceiver.cpp:923 [tid=140385354704640][chan=0] response is 'RSP POWEROFF 0'
```

Well, the answer becomes clear if you leave Osmo-BTS-TRX running for a minute or two,

- **Eventually the process stops, reporting:**

```
<000d> abis.c:142 Signalling link down
<0001> bts.c:292 Shutting down BTS 0, Reason Abis close
```



So what's going on? In the same way we saw our Virtual BTS shut itself down, without a connection to the BSC (Via the Abis interface) the BTS will shut itself down, as it's not able to run on it's own.

- **In our next post we'll introduce our BSC and provision a BTS on it.**

3.7.5 Integrating our LimeSDR BTS with OsmoBSC

- Before we fire up the BTS side of things make sure you've stopped the virtual BTS and disabled it.

```
systemctl stop osmo-bts-virtual
systemctl disable osmo-bts-virtual
```

3.7.5.1 Configure Osmo-BTS-TRX

- Next up we'll edit the config of osmo-bts-trx

```
vi /etc/osmocom/osmo-bts-trx.cfg
```

- We'll edit the ***oml remote-ip*** to the IP of the server running your BSC, if you're running on the same machine you can leave it as localhost (127.0.0.1).

- Next up we'll set the **Unit-ID** of the BTS, this identifies the BTS inside the BSC, I'll set it to **unit-id 1234** by changing ***ipa unit-id 1234 0***

- Finally we'll change the logging config to show everything by changing it to:

```
log stderr
 logging filter all 1
!
```

- Next up we'll configure the BTS on the BSC

3.7.5.2 BSC Provisioning

This is essentially a copy of the provisioning process we followed in chapter 6.4.3, the only difference is we'll use BTS 2 (as BTS 1 is setup for our Virtual BTS) in the config, and set the few different identifier such as the ipa unit id for the SDR based BTS.

```
telnet localhost 4242
OsmoBSC> enable
OsmoBSC# configure terminal
OsmoBSC(config)# network
OsmoBSC(config-net)# bts 2
OsmoBSC(config-net-bts)# type sysmobts
OsmoBSC(config-net-bts)# description "LimeSDR Based BTS"
OsmoBSC(config-net-bts)# ipa unit-id 1234 0
OsmoBSC(config-net-bts)# band DCS1800
OsmoBSC(config-net-bts)# codec-support fr hr efr amr
OsmoBSC(config-net-bts)# cell identity 1234
OsmoBSC(config-net-bts)# location_area_code 1234
OsmoBSC(config-net-bts)# base_station_id_code 1234
OsmoBSC(config-net-bts)# base_station_id_code 12
OsmoBSC(config-net-bts)# ms max power 40
OsmoBSC(config-net-bts)# trx 0
OsmoBSC(config-net-bts-trx)# max power red 20
OsmoBSC(config-net-bts-trx)# arfcn 876
```

```
OsmoBSC(config-net-bts-trx)# timeslot 0
OsmoBSC(config-net-bts-trx-ts)# phys chan config CCCH+SDCCH4
OsmoBSC(config-net-bts-trx-ts)# exit
OsmoBSC(config-net-bts-trx)# timeslot 1
OsmoBSC(config-net-bts-trx-ts)# phys_chan_config TCH/F
OsmoBSC(config-net-bts-trx-ts)# exit
OsmoBSC(config-net-bts-trx)# timeslot 2
OsmoBSC(config-net-bts-trx-ts)# phys chan config TCH/F
OsmoBSC(config-net-bts-trx-ts)# exit
OsmoBSC(config-net-bts-trx)# timeslot 3
OsmoBSC(config-net-bts-trx-ts)# phys_chan_config TCH/F
OsmoBSC(config-net-bts-trx-ts)# exit
OsmoBSC(config-net-bts-trx)# timeslot 4
OsmoBSC(config-net-bts-trx-ts)# phys chan config TCH/F
OsmoBSC(config-net-bts-trx-ts)# exit
OsmoBSC(config-net-bts-trx)# timeslot 5
OsmoBSC(config-net-bts-trx-ts)# phys_chan_config TCH/F
OsmoBSC(config-net-bts-trx-ts)# exit
OsmoBSC(config-net-bts-trx)# timeslot 6
OsmoBSC(config-net-bts-trx-ts)# phys_chan_config TCH/F
OsmoBSC(config-net-bts-trx-ts)# exit
OsmoBSC(config-net-bts-trx)# timeslot 7
OsmoBSC(config-net-bts-trx-ts)# phys_chan_config TCH/F
OsmoBSC(config-net-bts-trx-ts)# exit
OsmoBSC(config-net-bts-trx)# exit
OsmoBSC(config-net-bts)# exit
OsmoBSC(config-net)# exit
OsmoBSC(config)# exit
OsmoBSC# copy running-config startup-config
```

3.7.5.3 Starting the SDR based BTS

Before we start the SDR based BTS it's probably best to have 3 terminals open,

One logged into Osmo-BSC with logging enabled (check chapter 6.4.2 for info on how to do that).

- We'll start another terminal for running the TRX modem / Layer 1 interface:

```
osmo-trx-lms -C /etc/osmocom/osmo-trx-lms.cfg
```

- And in another new terminal we'll start the BTS side;

```
osmo-bts-trx -c /etc/osmocom/osmo-bts-trx.cfg
```

- All going well our terminal with Osmo-BSC should report the connection:

```
OsmoBSC#
<0016> input/ipa.c:287 0.0.0.0:3003 accept()ed new link from 10.0.1.252:39595
<0003> osmo_bsc_main.c:291 bootstrapping RSL for BTS/TRX (2/0) on ARFCN 875 using MCC-MNC 001-01
LAC=1234 CID=1234 BSIC=12
```

- And the osmo-trx-lms and osmo-bts-trx windows should have data flying by at a rate of knots.

3.7.5.4 Verifying Cell Operation

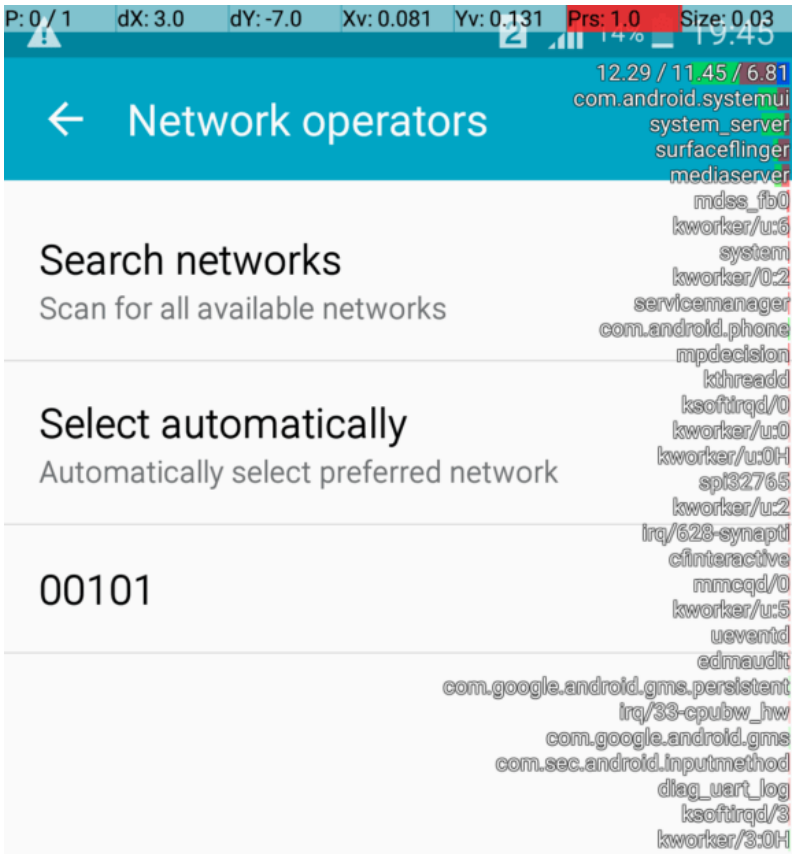
If all is going to plan, our SDR is connected to our machine via *osmo-trx-lms* which is acting as a modem for *osmo-bts-trx* which is now connected to the BSC. Lot to go through, but it gets easier from here.

Let's run a scan of the networks on our phone. I found putting mine on GSM only before scanning for networks meant it popped up a heck of a lot faster.

And lo, there it is.

Our cell is online and broadcasting it's info. You won't be able to connect to it at this stage as we've still got a few more steps to go.

In the next post we'll introduce the Home Location Register and then the MSC.


P: 0 / 1
dX: 3.0
dY: -7.0
Xv: 0.081
Yv: 0.131
Prs: 1.0
Size: 0.03
14%
19:45
12.29 / 11.45 / 6.81
com.android.systemui
system_server
surfaceflinger
mediaserver
mdss_fb0
kworker/u:6
system
kworker/0:2
servicemanager
com.android.phone
mpdecision
kthreadd
ksoftirqd/0
kworker/u:0
kworker/u:0H
spi32765
kworker/u:2
irq/628-synapti
cfinteractive
mmcqd/0
kworker/u:5
ueventd
edmaudit
com.google.android.gms.persistent
irq/33-cpubw_hw
com.google.android.gms
com.sec.android.inputmethod
diag_uart_log
ksoftirqd/3
kworker/3:0H
Network operators
Search networks
Scan for all available networks
Select automatically
Automatically select preferred network
00101

3.8 OsmoHLR

- We actually installed OmsoHLR in the post on Base Station Controllers **chapter 6.4**, so we'll just need to start the daemon / service:

```
systemctl start osmo-hlr
```

- We are going to enable the EIR functionality of the HSS by changing the configuration of the HLR, this is optional but it's useful to use the EIR functionality.

- Like with our other network elements we'll use Telnet to interactively configure this one,

```
root@gsm-bts:/home/nick# telnet localhost 4258
Welcome to the OsmoHLR VTY interface
OsmoHLR> enable
OsmoHLR# configure terminal
OsmoHLR(config)# hlr
OsmoHLR(config-hlr)# store-imei
OsmoHLR(config-hlr)# exit
OsmoHLR(config)# exit
OsmoHLR# copy running-config startup-config
```

3.8.1 Adding Subscribers to OsmoHLR

- If you want to authenticate subscribers properly (confirm their identity) and enable encryption on the air interface, you'll need to know the K key of the SIM, [for that you'll need a programmable SIM card like the Sysmocom programmable SIM cards,](#) (By buying from Sysmocom you're supporting the Osmocom project too).

- **We'll connect to OsmoHLR via Telnet, the port it listens on is 4258:**

```
root@gsm-bts:/home/nick# telnet localhost 4258
Welcome to the OsmoHLR VTY interface
OsmoHLR> enable
OsmoHLR# subscriber imsi 001010000000004 create
OsmoHLR# subscriber imsi 001010000000004 update msisdn 61412341234
OsmoHLR# subscriber imsi 001010000000004 update aud2g comp128v3 ki
465B5CE8B199B49FAA5F0A2EE238A6BC
```

  - ✓ So we've created a subscriber with IMSI 001010000000004 in the HSS and assigned an MSISDN (phone number).
  - ✓ Optionally, if we're using SIM cards we can program by setting the Ki / K key for authentication using the *update aud2g* function, if not we can skip that step.
  - ✓ And with that we've added our first subscriber, lather rinse repeat with any additional subscribers / SIMs you want to provision.

- By default subscribers created using this method have access to both Circuit Switched (Voice and SMS) and Packet Switched (Data) networks. (We haven't configured Packet Switched services yet)

- So, If you'd like to restrict access to one, both or none of the above options, you can do that by using the subscriber update command to set the services available to those subscribers.

```
OsmoHLR# subscriber id 3 update network-access-mode cs+ps
OsmoHLR# subscriber id 3 update network-access-mode cs
OsmoHLR# subscriber id 3 update network-access-mode ps
OsmoHLR# subscriber id 3 update network-access-mode none
```

3.8.2 Creating Subscribers on Demand (Optional)

For most scenarios you would pre-provision each SIM in the HLR, if the SIM's IMSI isn't in the HLR then it's access is rejected. However there are some scenarios where you may want to allow anyone to access the network, in this scenario Osmo-HLR features a "Create Subscribers on Demand" function.

This may be useful if you're setting up a network where you don't control the SIMs for example.

- Let's say we want to automatically create users with access to voice & data services and assign a 10 digit MSISDN for that subscriber, we can do that with:

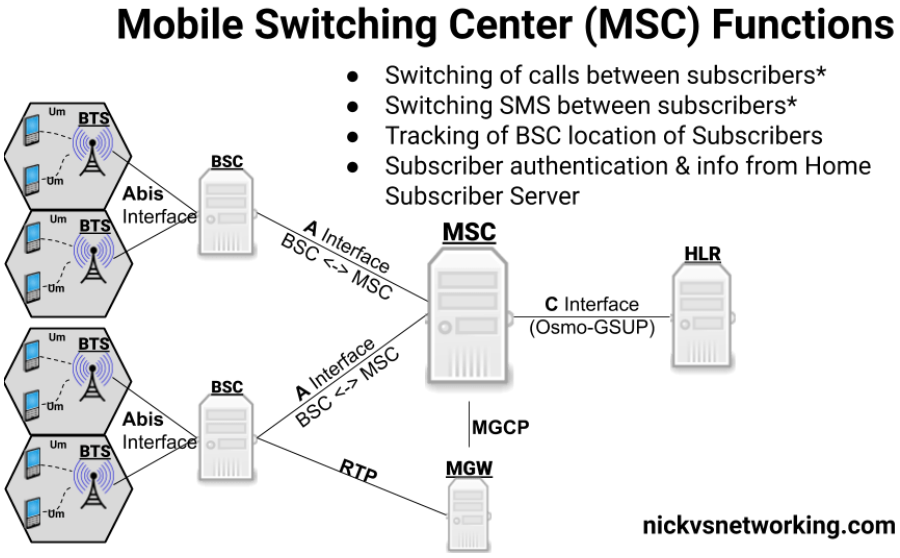
```
OsmoHLR> enable
OsmoHLR# configure terminal
OsmoHLR(config)# hlr
OsmoHLR(config-hlr)# subscriber-create-on-demand 10 cs+ps
```

- Alternatley you may wish to simply add the subscriber to the HLR but not provide any services:

```
OsmoHLR> enable
OsmoHLR# configure terminal
OsmoHLR(config)# hlr
OsmoHLR(config-hlr)# subscriber-create-on-demand no-msisdn none
```

3.9 OsmoMSC

The MSC handles switching of voice calls and SMS/text messages between local & remote subscribers and networks.



3.9.1 Switching Function

Because GSM was designed to be voice centric (Keep in mind the first GSM network went live in 1991) the MSC's primary function is switching phone calls between subscribers.* For this the MSC has to keep track of which subscribers it's currently serving, their capabilities and how to reach them -which BSC they're being served by and therefore which BTS they're being served by.

- The OsmoMSC also features a minimalistic **SMSC** (Short Message Service Server) for routing SMS traffic between subscribers on the network.
- This basic SMSC acts in a store-and-forward fashion. Production networks would typically use an external SMSC for handling SMS, OsmoMSC has the SMSC functionality built in by default, but the interfaces are there if you wanted to use an external SMSC.
- Any calls/texts to subscribers/destinations outside the MSC (for example a call to a mobile subscribers on a different carrier or on the PSTN) are typically routed to another MSC known as the **Gateway MSC**.
- The GMSC handles the interconnection with other networks. We'll touch upon this later with the SIP connector, but for now we'll focus just on on-net calls between subscribers.
- It's worth noting that the MSC does not sit in the media stream, it just sets up and tears down the calls, we'll cover more on the nitty-gritty of calling in GSM soon.

3.9.2 Setup & Connections

The BSC we setup earlier in **chapter 6.4** communicates with the MSC via *SS7 Point Codes*.

We'll go into how point codes route requests in a later post, but so long as you're running Osmo-BSC, Osmo-MGW, Omso-MSC and Osmo-HLR on the same machine you won't need to link them to each other like we had to do with adding our BTS to the BSC.

- Instead we'll just need to start everything required:

```
systemctl restart osmo-stp
systemctl restart osmo-hlr
systemctl restart osmo-mgw
systemctl restart osmo-msc
```

- The GSUP connection between the MSC and the HLR will be established at startup, but BSCs will only establish a connection to the MSC when they need something from the MSC.
- Once we've got everything started we can Telnet into the MSC to confirm it's running and check it's status:

```
root@gsm # telnet localhost 4254
```

3.10 Calls & SMS

- **Let's get some traffic on our network.**
- For starters we'll need to start each of our network elements and bring up whichever BTS hardware we're using.
- In order for our calls to have audio, we'll need to set a parameter on the Media Gateway. We'll cover the Media Gateway in more detail down the line, but there's one value in the MGW we'll need to set in order to have calls working, and that's the **rtp bind-ip** value.
- You can either set it in the config file or via VTY/Telnet on port 4243.
- We've talked about using systemctl to start all the services, but there's a script in the /etc/osmocom directory called *osmocom-all.sh* which starts all the network elements for us.
- Once you've got all the services started I'd suggest hopping onto the OsmoBSC and enabling all the logging you can, then connecting / starting your BTS.
- You should see the Abis over IP connection & OML link come up as the BTS connects to the BSC.
- And then, hold your breath, power up a phone and search for networks.
- All going well you'll see OsmoMSC in the network search, select it and you should see log data flying by as the phone ("terminal") connects to the network.
- Assuming you configured the IMSI of the SIM on the HLR you should be connected to the network and showing bars on the phone.
- You can check your phone number (MSISDN) by dialling the USSD code ***#100#**
- But it's not a network with just one phone connected, connect a second phone, check it's phone number the same way and call from one to the other.
- SMS should also just work.

4 How to get in LTE Network

Like in GSM network Osmocom offered for us the possibility to build an entire virtual network and we were able also to make it half-real network by connecting it to the LimeSDR and feed it with necessary configuration that we can get from the IMSI catcher.

From this point we can say that we was ready to do all the attack described in chapter 2.1, you can imagine if an insider attack was offered some good network information (IR21 should be good ☺) to make our virtual network (active IMSI catcher) a part from a real network operator.

The same for the LTE, in a way to simulate the threat attack described in chapter 2.4 we also need some virtualized network but this time we will use OpenLTE, srsRAN (old name srsLTE) and Open air interface by connecting it to supported USRP devices like (B210 or B200).

4.1 Open aire interface

OAI is open source software that implements both the core network (EPC) and access-network (EUTRAN) of 3GPP cellular networks. Currently, it provides a standard compliant implementation of Release 10 LTE and establishes generic interfaces with various hardware manufactures (including Ettus Research).

4.1.1 OAI IMSI catcher with B200mini

Our theoretical work based on this research:
https://www.researchgate.net/publication/318925138_Easy_4GLTE_IMSI_Catchers_for_Non-Programmers/link/59f6f4dd0f7e9b553ebd46ee/download

4.1.1.1 Hardware

- **Two computer**s: one Intel NUC D54250WYK (i5-4250U CPU@1.30GHz) and one Lenovo ThinkPad T460s (i7-6600U CPU@ 2,30GHz).
- Both run 64-bit Kubuntu 14.04 kernel version 3.19.0-61-low latency and have USB3 ports, which are prerequisites for running the OAI software.
- The Intel NUC computer was attached with standard peripherals (display screen, mouse, and keyboard).
- **Two USRP B200mini**: used to set up the eNodeBs from Ettus Research that can be programmed to operate over a wide radio-frequency range (70MHz - 6GHz), communicating in full duplex
- **One Samsung Galaxy S4 device:** used to find the LTE channels and TACs used in the targeted area.
- **Two LG Nexus 5X phones**: running Android v6 with different USIMs from the two biggest Norwegian operators, and the two biggest Romanian operators used for testing the IMSI Catcher.

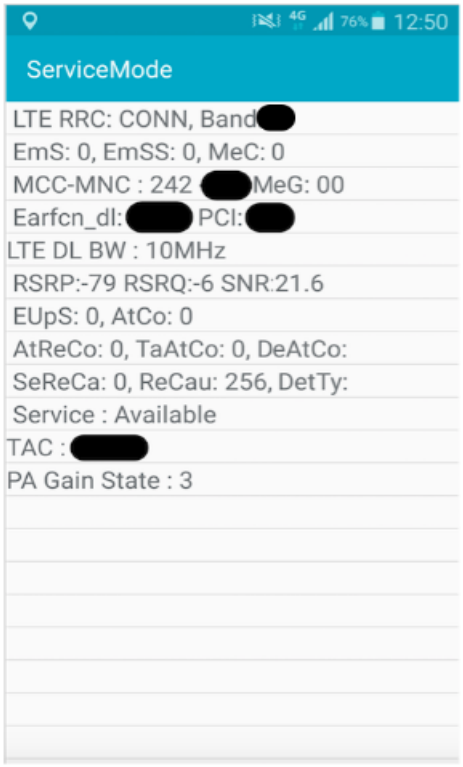
4.1.1.2 Software

- **Service and Testing Modes**: Seen as a facility of the operating system and the privilege rights of the user, service or testing modes of mobile devices offer important information about the LTE network.

  We describe, for comparison, the information displayed by the two types of phones we used during the experiments:

- ***Samsung phones*** offer LTE connection details by default. To access Service Mode, call *#0011#. The most important pieces of information are the EARFCN DL (downlink EARFCN) and the TAC. Other interesting information include the MCC, MNC and Cell ID.

- ***Android phones (including Samsung phones)*** access Testing Mode by calling *#*#4636#*#*. This is a feature available on all Android phones, which does not necessarily display EARFCN DL by default (it is dependent of the Android version), but displays information about several LTE cells that coexist in the area on which the phone might downgrade to in case the actual cell becomes unavailable.



(a) Service Mode (Samsung, root)

(b) Testing Mode (Android, non-root)

The UE is registered to the first displayed cell (mRegistered=YES), while the others are showed to be accessible. However, last versions of Android or applications such as LTE G-Net Track Lite or NetCell Tracker (in root mode) can provide EARFCN DL and other information in a user-friendly format.

- **OAI (Open Air Interface):** Basically, the LTE network is emulated on a computer, and the USRP is used as the radio platform for the eNodeB implementation.
  It is recommended to run EPC and EUTRAN on different machines, but OAI accepts both on a single computer too.
  In this research, they used two machines, one for each of the two rogue eNodeBs that must run concurrently.

  To install OAI follow the instruction in this link: https://open-cells.com/index.php/2019/09/22/all-in-one-openairinterface/

4.1.1.3 Build and operate LTE IMSI Catcher

Two rogue eNodeBs should run using the OAI software, and setting up with the proper configuration files:

- **One rogue eNodeB** (called eNodeB Jammer from now on) causes the UE to detach from the serving cell that it is camped on, and to reselect to our rogue cell set up by the second eNodeB (called eNodeB Collector), which masquerades as an authorized eNodeB but with higher signal power.

- The eNodeB Jammer is turned on, using the radio channel of the cell that the UE camps on. This jams the radio interface and decreases the signal of the commercial eNodeB under the specified threshold, causing the UE to trigger a new search for available eNodeBs. The UE tries to camp to the cell that runs on the next priority frequency and provides the best signal, namely the rogue eNodeB.

  Phase 1. Gather the configuration parameters (EARFCN and TAC):

  1 Connect a mobile phone to the target network and read the EARFCN DL and TAC
  2 Configure and run the eNodeB Jammer, using the MCC and MNC of the target network and the EARFCN DL from the previous step
  3 Read the new value of EARFCN DL, after the UE performs reselection.

- **One eNodeB Collector** broadcasts the MCC and the MNC of the target network operator to impersonate the real network.

- The eNodeB Collector signals a TAC value different from the commercial one, which brings the UE to initiate a TAU REQUEST message.

  For simplicity, they configured it to the next available TAC (TAC of the commercial network + 1). They found that the TAC must not be changed for multiple runs of the experiment (assuming the commercial TAC is unchanged), therefore they kept this value constant.

- Besides the MCC-MNC of the target network, the eNodeB Collector must run on the LTE EARFCN (the absolute physical radio channel) which corresponds to the highest priority next to the jammed channel.

  This assures that the UE prefers the eNodeB Collector prior to any other commercial eNodeB in the area.

  Phase 2. Configure and run the LTE IMSI Catcher:

  1 Configure and run the eNodeB Collector, using the MCC and MNC of the target network, a different TAC than the one in Phase 1 and the EARFCN DL set to the value in Phase 1.3
  2 Configure and run the eNodeB Jammer, using the MCC and MNC of the target network and the EARFCN DL set to the value in Phase 1.1.

The channel displayed in Phase 1.1 is associated with the highest priority (unless the signal power is below the reselection threshold).

The UE connects to it even if the signal power is not so strong. This can be easily seen by comparing the information displayed by the mobile device before and after the eNodeB Jammer is turned on.

The channel in Phase 1.3 has either the same priority, but lower signal power, or lower priority, regardless the signal power.

Once the eNodeB Jammer is active, this triggers an ATTACH REQUEST message from the UE to the eNodeB Collector. Then the UE will reveal its IMSI as a response to an IDENTITY REQUEST query from our Collector cell.

4.2 OpenLTE

OpenLTE is an open source implementation of the 3GPP LTE specifications. To make easy and simple like it should, this is the link into the project: https://github.com/mgp25/OpenLTE and all what we need to do is to follow the steps.

While am reading the instruction something really got my attention which is this part down below:

Note that the first 3 digits of IMSI is the MCC (or Mobile Country Code) **and the two digits after the MCC is the MNC** (or Mobile Network Code). In the above example the MCC is 901 and the MNC is 55. It is not necessary but helps the Mobile Station a lot to set the MCC/MNC of your LTE network as your programmed SIM cards dictates. You can change the IMSI value during the SIM card programming stage also to match the specification of a test network: `MCC=001` and `MNC=01`.

Please, note that you will need to setup an APN on your device in order to successfully have data connectivity. We will add indications on how to achieve this as far as our work progresses.

You should NEVER use the MCC/MNC configuration of a commercial provider!

So I think you got the message! This is what bad guy do but not us today (we don't like jail ☺)

OpenLTE Tx Configuration

Tx configuration:

```
write band 20
write bandwidth 5
write dl_earfcn 6300
write mcc 214
write mnc 12
write n_ant 1
write rx_gain 30
write tx_gain 86
```

So to be honest with you I don't have the budget to create a real attack scenario as you can see hardware not low cost, and implementation like this it will pass the 1000$ (PC, Phone, B210).

4.2.1 OpenLTE scanner with BladeRF

What it could be better than reading some Chinese or Russian work, do you agree with me? So, for that, to make short this is the link : https://blog.csdn.net/sinat_16643223/article/details/106702462

- **BladeRF Drivers:**

```
mkdir bladeRF
wget -c https://github.com/Nuand/bladeRF/archive/master.zip
unzip master.zip
cd bladeRF-master
cd host
mkdir build
cd build
cmake -DCMAKE_BUILD_TYPE=Release -DCMAKE_INSTALL_PREFIX=/usr/local -DINSTALL_UDEV_RULES=ON ../
make -j4
make install > install.log
ldconfig
```

- **BladeRF firmware:**

```
wget -c http://www.nuand.com/fx3/bladeRF_fw_v1.8.0.img
bladeRF-cli -f bladeRF_fw_v1.8.0.img -v verbose
```

- **GNU Radio:**

```
mkdir gnuradio
cd gnurdio
wget http://www.sbrac.org/files/build-gnuradio
chmod a+x build-gnuradio
./build-gnuradio -v
sudo apt-get install libpolarssl-dev
```

The above steps require a lot of dependency packages. Children who want to be lazy can use the Ubuntu LiveCD released by GnuRadio, which has already built a series of dependency environments required by SDR such as gnuradio, HackRF, BladeRF, USRP, gqrx, rtl-sdr, etc. Using this method can avoid most of the pits encountered in the installation system environment.

- **OpenLTE compilation**

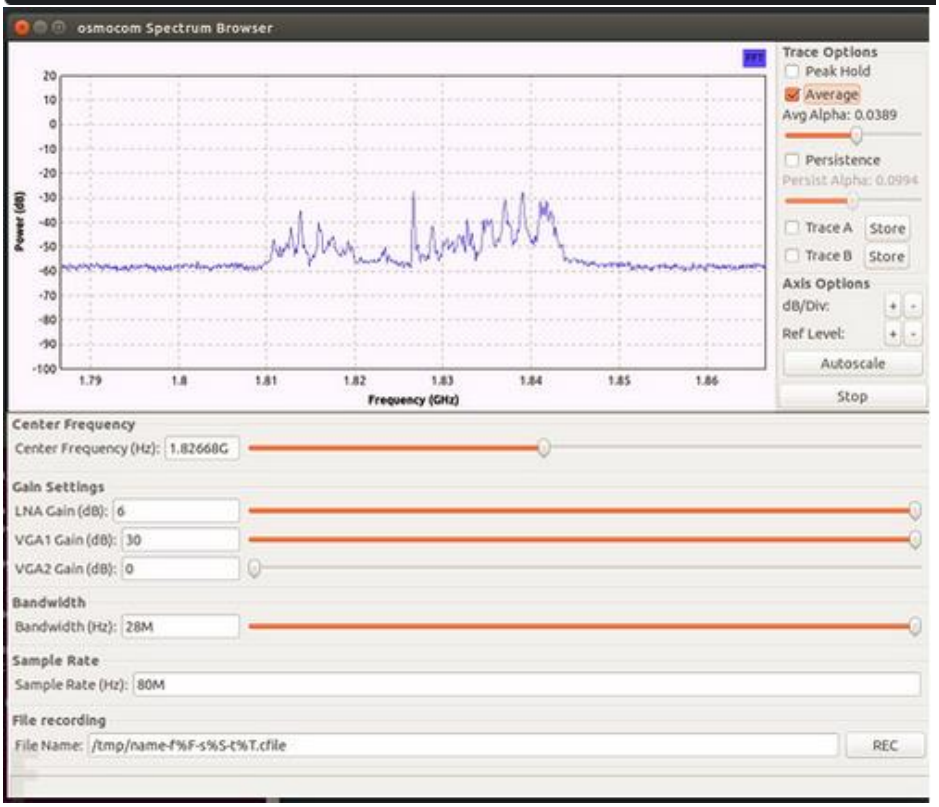
```
wget http://ufpr.dl.sourceforge.net/project/openlte/openlte_v00-19-04.tgz  //（目前最新版）
tar zxvf openlte_v00-19-04.tgz
cd openlte_v00-19-04/
mkdir build
cd build
sudo cmake ../
sudo make
sudo make install
```

- **Search nearby base station**

  Insert the SDR device, here they used the BladeRF (Tested HackRF can also be used, but because HackRF uses USB 2.0 to transmit data, its efficiency will be much lower than BladeRF, students with conditions can use USRP):

```
ubuntu@ubuntu:~/LTE/openlte_v00-19-04$ ls
AUTHORS          cmn_hdr              LTE_fdd_dl_file_gen   NEWS
build            COPYING              LTE_fdd_dl_file_scan  octave
ChangeLog        enodeb_nat_script.sh LTE_fdd_dl_scan       README
cmake            liblte               LTE_fdd_enodeb
CMakeLists.txt   libtools             LTE_file_recorder
ubuntu@ubuntu:~/LTE/openlte_v00-19-04$ bladeRF-cli -p

  Backend:        libusb
  Serial:         e2c24e2f49e0e4d0c76b7f53d9d97a95
  USB Bus:        1
  USB Address:    3

ubuntu@ubuntu:~/LTE/openlte_v00-19-04$
```

```
osmocom_fft --samp-rate 80000000
```



- **After OpenLTE compilation is completed, an executable file will be generated in the build directory:**

```
ubuntu@ubuntu:~/LTE/openlte_v00-19-04/build$ ls
CMakeCache.txt          get_swig_deps.py      LTE_fdd_dl_scan
CMakeFiles              liblte                LTE_fdd_enodeb
cmake_install.cmake     libtools              LTE_file_recorder
cmake_uninstall.cmake   LTE_fdd_dl_file_gen   Makefile
CTestTestfile.cmake     LTE_fdd_dl_file_scan  python_compile_helper.py
```

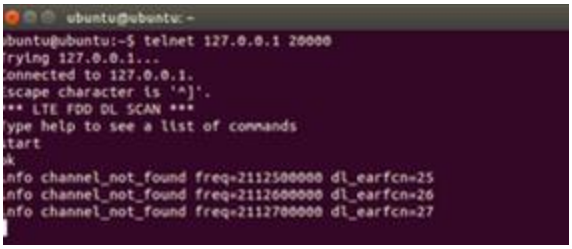
```
1 cd LTE_fdd_dl_scan
2 ./LTE_fdd_dl_scan
```

- **Create a new terminal and enter the OpenLTE working terminal interactive interface through Telnet:**

```
telnet 127.0.0.1 20000
```

登录后复制（为作者贡献原力分）

- **Execute start on the telnet side to start scanning**



LTE_fdd_dl_scan will scan the FCN value in the **dl_earfcn_list** list: from 25 to 575

ARFCN: Absolute Radio Frequency Channel Number (ARFCN) is a numbering scheme used to identify special radio frequency channels in the GSM wireless system.

ARFCN in 4G LTE is called EARFCN.

- **Search Telecom FDD LTE network: (telnet side)**

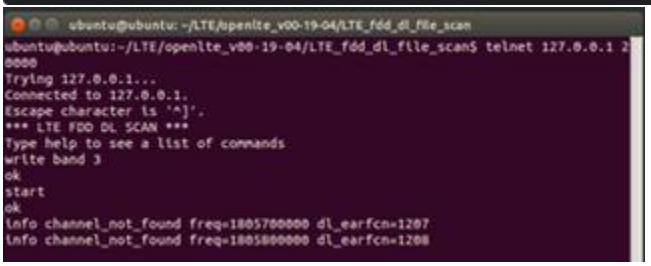
```
1 write band 1
2 help
3 start
```

登录后复制（为作

```
write band 1
ok
help
***System Configuration Parameters***
        Read parameters using read <param> format
        Set parameters using write <param> <value> format
        Commands:
                start    - Starts scanning the dl_earfcn_list
                stop     - Stops the scan
                shutdown - Stops the scan and exits
                help     - Prints this screen
        Parameters:
                band = 1
                dl_earfcn_list = 25,26,27,28,29,30,31,32,33,34,35,36,37,38,3
                repeat = on
start
ok
info channel_not_found freq=2112500000 dl_earfcn=25
info channel_not_found freq=2112600000 dl_earfcn=26
info channel_not_found freq=2112700000 dl_earfcn=27
info channel_not_found freq=2112800000 dl_earfcn=28
info channel_not_found freq=2112900000 dl_earfcn=29
info channel_not_found freq=2113000000 dl_earfcn=30
info channel_not_found freq=2113100000 dl_earfcn=31
info channel_not_found freq=2113200000 dl_earfcn=32
```

- **Search Unicom FDD LTE network: (telnet side)**

```
stop
write band 3
start
```



4.3 srsRAN (from libLTE to srsLTE)

Early 2014 the first public release of what is now the srsLTE project. At that time, it was known as libLTE

In the 7 years since then, srsLTE has grown to almost a million lines of code with full-stack UE, eNodeB and EPC applications providing a complete end-to-end 4G network.

In late 2018, 3GPP delivered the final Release 15 specifications for 5G NR. Building upon the technical foundation of 4G LTE, 5G NR introduces more flexibility, higher bandwidths and support for new millimeter wave frequency bands.

Since early 2020, the SRS team has been developing support for this new standard and with the 21.04 release, they have added support for their first complete 5G application, the NSA-mode UE. This will be followed by the gNodeB application.

The focus of the project beyond 4G LTE and into 5G NR, the “srsLTE” naming no longer fits like it used to. So, coinciding with the 21.04 release, the srsLTE project has become the **srsRAN** project.

The srsRAN suite includes:
srsUE - a full-stack SDR 4G/5G-NSA UE application (5G-SA coming soon)
srsENB - a full-stack SDR 4G/5G-NSA eNodeB application (5G-SA coming soon)
srsEPC - a light-weight 4G core network implementation with MME, HSS and S/P-GW

Now I wish you a good deep reading because there no sense to repeat the official documentation here: For application features, build instructions and user guides see the srsRAN documentation.

Also, you can check again the work of NickvsNetworking with srsLTE:

https://nickvsnetworking.com/srs-lte-software-defined-lte-stack-with-bladerf-x40/

https://nickvsnetworking.com/srslte-install-for-bladerf-limesdr-on-debian-ubuntu/

4.3.1 srsRAN/OpenLTE IMSI catcher with B210

During my research I found two available publish:

- https://www.blackhat.com/docs/eu-15/materials/eu-15-Borgaonkar-LTE-And-IMSI-Catcher-Myths-wp.pdf
- https://arxiv.org/pdf/1510.07563.pdf
  paper was prepared within the scope of employment.
  NDSS '16, 21-24 February 2016, San Diego, CA, USA
  Copyright 2016 Internet Society, ISBN 1-891562-41-X
  http://dx.doi.org/10.14722/ndss.2016.23236

According to this two research, to build an LTE network, they used a USRP B210 device, which acts as a base station. On the software side, they modified OpenLTE and srsLT packages in order to be able to communicate with commercial LTE devices.

Following figure depicts the setup.

According to their research they take advantage by Exploit weaknesses in the cellular network security design: Device attach, authentication, & paging procedure

- **Implementation for the Passive and semi-passive attack setup:** In order to sniff LTE broadcast channels, they utilized parts of srsLTE.
  - In particular, they used the pdsch-ue application to scan a specified frequency and detect surrounding eNodeBs. It can listen and decode SIB messages broadcast by eNodeB.
  - Further, they modified pdsch-ue to decode paging messages which are identified over-the-air with a Paging-Radio Network Temporary Identifier (P-RNTI). Upon its detection, GUTI(s) and/or IMSI(s) can be extracted out of paging messages.
- **In semi-passive attack mode**, they used Facebook and WhatsApp applications over the Internet, in addition to initiating communication with targets via silent text messages or phone calls.

- **Implementation for the active attack (Rogue eNodeB):** make sure to visit chapter 2.4.4 to understand how LTE eNodeB selection work.

  The rogue eNodeB broadcasts MCC and MNC numbers identical to the network operator of targeted subscribers to impersonate the real network operator. Generally, when UE detects a new TA it initiates a "TAU Request" to the eNodeB. In order to trigger such request messages, the rogue eNodeB operates on a TAC that is different from the real eNodeB

  Their active attack is launched using the USRP B210 and a host laptop which together are running OpenLTE.

  Further, they programmed LTE_Fdd_enodeb to include LTE RRC and NAS protocol messages to demonstrate active attacks.

  In addition, we modified the telephony protocol dissector available in Wireshark to decode all messages exchanged between the rogue eNodeB and UE. These modifications are submitted to the Wireshark project and are being merged into the mainstream application.

Two things not clear to me, and my question was: How they did for:

- modifying the telephony protocol dissector available in Wireshark
- Programming the LTE_Fdd_enodeb to include LTE RRC and NAS protocol messages

And sure that was on purpose that they didn't described in their research, because this the key of the wall Rogue eNodeB

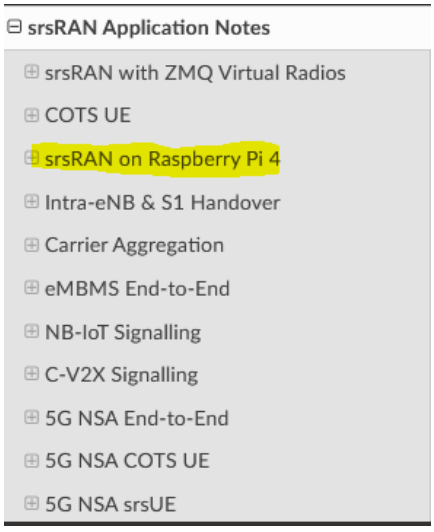
But today in our work we don't want to answer these two question, also it doesn't seem to me something hard, I already done some work in decoding radio message:

https://www.scribd.com/document/360213237/Decoding-radio-messages-of-the-layer3-protocols-In-GSM-UMTS-and-LTE-Networks

My research will help to understand even what is NAS and relation with protocol, how RRC protocol are programmed etc…

4.3.2 FemtoCell (LimeSDR-Mini) connect to RPi

Again this is one many application build on the srsRAN project:



For hardware and software setup and eNodeB configuration all of these you can found detailed in this link:

https://docs.srsran.com/en/latest/app_notes/source/pi4/source/index.html

5 Virtual 5G network

First, today 5G is based on the successor of the LTE there is no standalone 5G system, all what we have from the 5G is the new radio part NR.

- So the question is do LTE attack work on 5G NSA? Yes but I will take different faces,

- Do 5G SA better then 5G NSA? Sure:

Security Features	5G NSA	5G SA
Encrypted SUPI	❌	✅
Mandatory Fresh 5G-GUTI reallocation	❌	✅
Paging by only 5G-S-TMSI	❌	✅
ABBA parameter	❌	✅
Integrity protection	❌	✅
UE-assisted Network based IMSI catcher detection	❌	✅
Secure UE capabilities transfer	✅	✅

- **Do we have some 5G virtualized network? Yes and these are the links to it**:
  https://github.com/open5gs/open5gs
  https://nickvsnetworking.com/my-first-5g-core-open5gs-and-ueransim/
  https://nickvsnetworking.com/open5gs-nrf-setup/
  https://nickvsnetworking.com/connecting-any-3rd-party-hss-to-open5gs-mme/

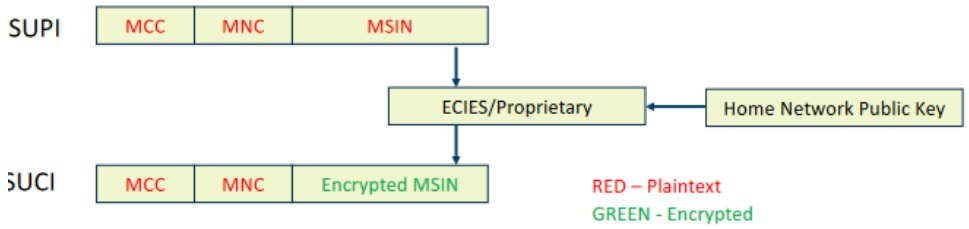
- **Do IMSI catcher attack on 5G SA work? Can we still exploit weaknesses in the cellular network security design: Device attach, authentication, & paging procedure?**

  - First in 5G we speech SUPI + SUCI instead of IMSI for the public key of home network, and SUCI vulnerable to decoding. If SUPI is not based on IMSI, SUCI may not be random (lengh differs)*

    SUPI – Subscription Permanent Identifier

    SUCI – Subscription Concealed Identifier (SU-SHI)

    Public key of the home network operator



- Paging in 5G we speech 5G-S-TMSI or I-RNTI with a mandatory refresh applied after paging but sometime Lack of randomness and refreshens (when user is not moving) and possible to link 5G-GUTI to a subscriber

Time	4G LTE 2015	4G LTE 2021	5G NSA 2021
10:00	0xadf02cd4	0xdd348782	0xdd082f96
11:00	0xadf12cd4	0xdd72392f	0xdd097b32
12:00	0xadf32cd4	0xdd0423de	0xdd823ef3
13:00	0xadf62cd4	0xdd639202	0xdd87663d
14:00	0xadf82cd4	0xdd63192f	0xdd84c782

Date	5G NSA
20 June	0xC1A2B000
25 June	0xC1A33000
2 July	0xC1A3F008
3 July	0xC1B23007
21 June	0xC1B4E001

Does not change even after 10+ days
Remains same after device restart or flight mode on/off

24 June	0xF5863006
25 June	0xF5863006
2 July	0xF5863006
3 July	0xF5863006
6 July	0xF5863006

- Downgrade to 3G/2G or lower generations with unprotected messages (Registration Reject: LTE not allowed) Automatic timer-based recovery? Not implemented in many phones
- Tracking with 5G-AKA Vulnerabilities
- 5G NR tower carry data-traffic
  - Optional integrity protection for data-traffic
  - Not enabled in 4 NSA networks: Vulnerable to alter-attacks

  https://i.blackhat.com/USA21/Wednesday-Handouts/us-21-5G-IMSI-Catchers-Mirage.pdf

To be honest with you 5G deserver a standalone research☺, I will try to make something about that.

But for the moment I hope that you enjoyed reading this theoretical research.

6 Detection of IMSI Catcher

There are different applications available, which help to find the IMSI Catcher in your location. Applications contain a database of all the cell towers of mobile carriers in different countries and regularly update this list. Every time it detects a cell tower, it checks the list to see if it exists.

If it exists, then it is a legitimate one, and there is no danger. However, if the tower is not on the list, there is something suspicious going on – and there is a high probability that this is an IMSI Catcher. In this case, the best you can do is to turn off your phone and turn it on again, once you reach a safe location.

- **Below are some of the IMSI Catcher detector applications:**

- Osmocom – used to detect and fingerprint certain network characteristics
- Android IMSI-Catcher Detector
- SnoopSnitch
- Cell Spy Catcher
- GSM Spy Finder

https://www.findbestopensource.com/tagged/bladerf
https://www.ericsson.com/en/blog/2018/6/detecting-false-base-stations-in-mobile-networks
https://i.blackhat.com/USA-20/Wednesday/us-20-Quintin-Detecting-Fake-4G-Base-Stations-In-Real-Time.pdf

6.1 4G IMIS catcher Detection

- **Software setup:** Crocodile Hunter Software Stack:

- https://github.com/EFForg/crocodilehunter
- Backend based on SRSLTE (srsRAN)
- Open source LTE software stack
- Written in C++
- Communicates with frontend over a local socket

- **Hardware setup:**

- Laptop / Raspberry Pi
- USB GPS Dongle
- SDR compatible with SRSLTE: BladeRF, Ettus B200
- LTE Antennas

- **How that happen:**

- By Decoding MIB and SIB1 for all the cells that we can see and record them.
  - ✓ SRSLTE scans a list of EARFCNS
  - ✓ If we find a mib we decode mib and sib and send over socket
- Map the probable location of cells
  - ✓ Using trilateration and distance estimates we can figure out where all the towers are
  - ✓ Compare this to a ground truth such as wigle or opencellid
- Look for anomalies in the readings
  - ✓ Cells moving
  - ✓ Cells that change signal strength
  - ✓ Cells that aren't where they should be
  - ✓ Cells changing parameters
  - ✓ Cells missing parameters
  - ✓ New cells
- Locate suspicious cells and confirm results