JES, Vol. 52, No. 1, Jan. 2024 DOI: 10.21608/JESAUN.2023.226718.1249 Part B: Electrical Engineering

Journal of
Engineering Sciences
Faculty of Engineering
Assiut University

JES

مجلة
العلوم الهندسية
كلية الهندسة
جامعة أسيوط

journal homepage: http://jesaun.journals.ekb.eg

Penetration Testing of GSM Network using Man-In-The-Middle Attack

Received 7 August 2021; Revised 1 October 2021; Accepted 1 October 2021

Nosa Bello[1]
Ogechukwu A. Kanu[2]

Keywords
IMSI catcher, OpenBTS, Kraken, B210, GR-GSM, RTL-SDR, Wireshark, A5/1, rainbow tables

Abstract
Even though wireless communication technologies have advanced beyond the Global Systems for Mobile (GSM) Communications standard to mitigate its vulnerabilities, it is still a fallback technology when the coverage is limited, and modern protocols aren't available. There is a need for a comprehensive practical demonstration of the pools of vulnerabilities of the GSM architecture in the past decades using man-in-the-middle open-source tools and software-defined radios (SDRs) amidst the latest developments. It can be shown that an attacker can successfully carry out base station spoofing, IMSI catching, GSM packet sniffing, decoding, decryption and Denial of Service (DoS) attacks. Thus, this paper aims to comprehensively present practical demonstrations of the many vulnerabilities possible with available tools and SDRs. We exploited IMSI catching with a rogue BTS deployed using OpenBTS and USRP B210, GSM sniffing and decoding using GR-GSM and RTL-SDR, and A5/1 decryption using clever thinking and rainbow tables. It was observed that the one-way authentication of the GSM protocol allows most mobile devices to easily authenticate to the rogue BTS with spoofed MCC/MNC and that the strongest signal mostly wins. Also, it was observed that the possibilities of attacks on the target user like a DoS, or unencrypted communication, can be successfully carried out because the rogue BTS is in total control.
Though the vulnerabilities of GSM have been made known to the general public some network providers have not taken simple measures to mitigate them, thus this work can serve as a guideline for research purposes and an awareness to the general public.

1. Introduction

GSM telephony, recognized as the world's most popular communication technology, connects over four billion devices [1], [2]. Originally, GSM was designed with a strong emphasis on security and user authentication. It achieves this through mechanisms like a pre-shared key, challenge-response, and over-the-air encryption. Despite these measures, GSM remains susceptible to various types of attacks targeting different network components [3].

[1] Department of Electrical/Electronic Engineering, University of Benin, Benin City, Nigeria. nosabello@uniben.edu
[2] Department of Electrical/Electronic Engineering, University of Benin, Benin City, Nigeria. kanu.ogechukwu@eng.uniben.edu

https://doi.org/10.21608/JESAUN.2023.226718.1249

When a mobile entity (ME) attempts to join a GSM network, it undergoes an authentication process to validate the user's identity. This process involves the creation of a 128-bit pseudo-random number (RAND) by the Authentication Center (AuC), which is then sent to the mobile device. Within the Subscriber Identity Module (SIM), the mobile phone computes a 32-bit signed response (SRES) by encrypting RAND using the individual subscriber authentication key (Ki) [4]. Importantly, this encryption process takes place entirely in the SIM, ensuring that sensitive subscriber information, such as Ki, remains confidential. Subsequently, the AuC compares the received SRES from the subscriber with its own value. If they match, authentication is successful, granting the subscriber access to the network; otherwise, the connection is terminated.

GSM security primarily focuses on confidentiality and authentication, but it falls short in providing non-repudiation due to its one-way authentication [3]. In terms of data confidentiality, the SIM plays a crucial role by generating a 64-bit ciphering key (Kc). This key is responsible for encrypting and decrypting the data exchanged between the ME and the base station. The key derivation process involves the use of the same RAND employed in the authentication process, combined with the individual subscriber authentication key (Ki). Periodic rotations of the cyphering key further enhance security, and all decryption computations occur within the SIM.

The remainder of the paper is structured as follows: Section 2 presents the works of literature on A5 cyphering algorithms and related works. Then in Section 3, the materials and methodology employed in the work are discussed with the correspondingly obtained results shown in Section 4. In Section 5 we review the results, main findings and implications. Finally, Section 6 offers concluding remarks on the research.

2. Literature Review

Secured exchanges linking the mobile stations (MS) and the network involve the utilization of one of the A5 family of cyphering algorithms. Encrypted communication begins with a cyphering mode request from the GSM network. The mobile station performs real-time encryption and decryption of data using the designated cyphering algorithm in tandem with the cyphering key (Kc). These A5 algorithms are implemented in the hardware of the ME to ensure the security of data exchanges [5].

Within the framework of the GSM protocol, information is dispatched in the form of frame sequences, with each frame encompassing 228 bits. Every unencrypted data frame undergoes an XOR operation with a pseudorandom sequence generated by one of the A5 stream cypher algorithms. These algorithms, namely A5/1, A5/2 and A5/3, fulfil the role of safeguarding voice confidentiality during wireless transmission. The A5/1 cryptographic algorithm is used within Europe, Africa and the United States. Its operational parameters encompass a 64-bit key and a public initial vector spanning 22 bits. The A5/2 algorithm was introduced to extend the GSM standard to a broader spectrum of nations. This incorporation was undertaken while adhering to cryptography-related export restrictions prevalent within the United States. A5/2 can be understood as an intentionally attenuated rendition of the A5/1, exhibiting a striking similarity to its source algorithm. However, due to security concerns, the A5/2 is no longer integrated into mobile devices. Subsequently, the A5/3 algorithm emerged, drawing inspiration from the MISTY cypher [6]. Through a series of modifications, a swifter and hardware-compatible iteration dubbed KASUMI [5] was derived.

Unlike in recent times, the procurement expense associated with the requisite hardware for intercepting GSM traffic was notably high. This circumstance resulted in exclusive accessibility primarily by research and government entities due to financial constraints. Consequently, the scrutiny of GSM networks was substantially confined to organizations with government backing. This is no longer the case. Presently, open-source software applications geared towards the examination of GSM traffic and radio equipment, including SDRs (Software-Defined Radios) have emerged. These tools have been harnessed to exploit the vulnerabilities within the GSM protocol [7]. The primary focus of these exploits' centres on the A5 ciphering algorithms, although the precise composition of these algorithms remains officially confidential. Nevertheless, the research community has managed to reconstruct them through a fusion of reverse engineering and cryptoanalysis. Notably, the fundamental blueprint of A5/1 was disseminated in 1994, and the first cryptanalysis of A5/1 was conducted by Golit [8]

The first practicable attack that could be executed through open-source software and commercially available hardware was unveiled by Nohl in 2009 [9]. This revelation underscored the susceptibility of A5/1 to general pre-computation attacks. Specifically, for cyphers featuring modest small key lengths (such as A5/1 with its 64-bit key), it becomes feasible to formulate a codebook. This codebook functions as a form of a lookup table, facilitating the mapping between all conceivable ciphertexts and corresponding plaintexts. This setup can then be exploited for executing a known plaintext attack. With A5/1 as the focal point, a sufficient number of plaintext/ciphertext pairs permit the retrieval of the encryption key. Within the realm of GSM, numerous predetermined control messages are leveraged as known plaintexts [1].

Considering the full range of potential permutations, Nohl approximated that generating a codebook for A5/1 would necessitate a colossal 128 Petabyte of data and more than a century's worth of computational effort on a conventional personal computer. During their discourse, Nohl and Paget revisited strategies for expedited codebook computation and its compact storage. Essentially, Nohl proposed an optimized A5/1 engine tailored for parallelization with CUDA technology, a parallel computing platform [10] harnessing profoundly enhanced graphic processing units (GPUs). By employing this approach, Nohl projected that a complete codebook for A5/1 could be computed in three months using 80 GPUs. Subsequent presentations introduced refinements that enabled even shorter computation times—within a single month using 4 ATI GPUs [1], [4]. Furthermore, Nohl advocated for a hybrid approach to data storage, integrating distinguished point and rainbow tables [11]– [13]. This hybrid methodology significantly reduced the size of the codebook to a mere 2 terabytes.

Nohl estimations indicate that the success rate of this attack reaches 99% when data can be collected from a phone registered to the network, thereby maximizing the pool of known control frames. Conversely, if such data collection is not feasible, the success rate drops to 50% due to the limited availability of frames with known plaintext. In a subsequent presentation, Nohl and Munaut conducted a live demonstration showcasing the identification and decryption of phones and their calls [14]. Moreover, the contemplation of near real-time decryption through a distributed cracking network was put forth. The culmination of these experiments yielded rainbow tables—precomputed tables for reversing cryptographic hash functions—crafted for decrypting GSM conservations. These tables were publicly released in 2010, accompanied by an open-source tool able to retrieve intercepted communication keys [1]. Notwithstanding knowledge of frequency hopping sequences on the GSM air interface, the central limitation of these attacks lies in the intricacy of processing raw data from the radio channel

In a subsequent discourse, Nohl and Munaut undertook a comprehension demonstration, elucidating how the identification and decryption of phones and their calls can be actualized [14]. The complete process of decryption can be achieved by employing the subsequent open-source software tools

i. GNURadio [15] for capturing data from the radio channel.

ii. Airprobe [16] for parsing GSM control data.
iii. Kraken [17] for cracking the A5/1 session key based on the parsed data.
iv. Airprobe again, for decoding voice data.

The mentioned attacks capitalize on certain vulnerabilities inherent in the telecommunication infrastructure, facilitating the creation of a counterfeit mobile tower under the control of the attacker. The central security loophole exploited by the spurious tower, often referred to as an IMSI Catcher, stems from the fact that the GSM specification only mandates the authentication of the handset to the network while neglecting to enforce network authentication to the handset. Positioned between the target mobile phone(s) and the authentic towers supplied by the service provider, the IMSI Catcher exercises authority over communication parameters, including encryption algorithms, and can also engage in eavesdropping on traffic. This form of assault aligns with the Man-In-The-Middle (MITM) attack category. Some MITM attacks against GSM were introduced in [18], envisioning the victim connected to a counterfeit base station capable of intercepting and forwarding data between the victim’s phone and the network.

For authentication, the attacker connects to the network, prompting the network to dispatch an authentication request. This request is then forwarded to the victim, who computes SRES and transmits it back to the attacker. Leveraging this computed SRES, the attacker can authenticate itself to the network, effectively mimicking both the network of the victim and the targeted phone. Irrespective of the encryption algorithm chosen by the network, the attacker can direct the victim to employ a weak cypher like A5/2 (or even no encryption). This enables the attacker to utilize cryptanalysis of A5/2 to uncover the encryption key. Notably, the generation of the encryption key hinges solely on the network-specified RAND parameter. Consequently, the encryption key utilized between the victim and the attacker mirrors the key used between the attacker and the network, facilitating the decryption of all the traffic, even if the network requests a secure encryption algorithm like A5/3.

Concurrently, a substantial number of IMSI catcher devices have been commercialized. Paget and Nohl showed how a subscriber’s IMSI can be captured through an active attack [19]. This attack employs a counterfeit base station that could be assembled using open-source components such as OpenBTS, a 52 MHz clock, an Asterisk, and an inexpensive Universal Software Radio Peripheral (USRP). Data collection and decoding can be accomplished through open-source software like Wireshark. In 2010, Paget presented a practical GSM attack using open-source components [20]. This exploit capitalizes on the tendency of mobile phones to connect to the strongest base station signal. Since the base station wields full control over communication protocols, the phone can be instructed to employ no-traffic encryption (A5/0), permitting the attacker to intercept all plaintext traffic. The demonstration utilized hacked IM-ME equipment [21] and a USRP, connected to a laptop running OpenBTS and Asterisk. Additionally, when a 3G (UMTS) signal is present, Paget revealed the ability to force the victim’s connection to downgrade to 2G by jamming the 3G frequencies. This reinstates the conditions for the aforementioned attack. A limitation of this approach is that only outbound calls can be intercepted, as the phone becomes disconnected from the authentic network. Paget’s solution involves an MITM attack where the attacker also impersonates the victim’s telephone to the carrier. By negotiating the weakest feasible cypher (A5/2 or A5/1) for traffic encryption, the attacker can be subsequently cracked and proceed with decryption attempts.

Since the first demonstration of the attack on the GSM protocol by Karsten Nohl in 2009, there have been several other attacks to finely automate the task with more open-source tools as well as improvements to the underlying tool used. Thus, in this paper, we intend to extensively examine and demonstrate the variety of attacks that can be carried out on a targeted GSM user by employing open-source penetration tools in modern-day society.

3. Methods

3.1. Material

The different hardware and software needed include:

1. Personal Computer
2. USRP B210 [22]
3. RTL-SDR
4. Antennas: RFX900 for GSM 850/900 (800-1000MHz Transceiver, 200 mW output), RFX1800 for GSM 1800/1900 (1.5-2.1 GHz Transceiver, 100 mW output), VERT900 (824-960 MHz, 1710-1990 MHz Quad-band Cellular/PCS and ISM Band Vertical Antenna, 3dBi Gain, 9 Inches, Ideal for RFX900 and RFX1800).
5. GSM-compatible cellular phone.

The software needed includes:

1. Ubuntu 16.04 LTS, 18.04 LTS and Kali-Rolling.
2. OpenBTS 5.0 [23].
3. GNU Radio [24].
4. GR-GMS.
5. Airprobe.
6. Wireshark.
7. Kalibrate-RTL.
8. Kraken and A5/1 Rainbow Tables [25].

Practically, we achieved the base station spoofing, IMSI catching, GSM packet sniffing and decoding, A/1 decryption and DoS attack with the listed materials as follows.

3.2. Base station spoofing

A rogue BTS, whose aim is to imitate an actual GSM network tricking the intended victim’s mobile phone is paramount to the extent of the severity of the vulnerabilities of the GSM architecture. The implementation of such a system is very easy and sets the stage for the variety of attacks that are possible. It was done with OpenBTS, a free open-source software, and a USRP B210 SDR device using two VERT900 antennas for the uplink and downlink frequencies. The first step taken in deploying the rogue base station was to find suitable Absolute Radio Frequency Channel Numbers (ARFCNs) for it to operate on. This can be done with the GSM base station scanner, kalibrate-RTL or gr-GSM [26], [27].
A rogue base station can identify one of the neighboring BTSs that is far from the local cell and sit at that frequency (this is called base spoofing). This should make the mobile entity (ME) (victim) handover more quickly to the rogue BTS. Thus, establishing the rogue BTS with the list of neighboring ARFCNs involves the use of one of the neighboring cells (AFRCNs) and transmitting the rogue BTS at a power higher than the serving BTSs in the area. The next step taken in deploying the rogue BTS is spoofing a legitimate network provider. An MS identifies networks through the following: the Mobile Country Code (MCC) which is a 3-digit code specific to the country; the Mobile Network Code (MNC) which is a 2- to 3-digit code specific to the network; the Location Area Code (LAC) and the Network Name. Most SIM cards will connect to a base station with these parameters actively configured. These parameters can be configured in OpenBTS and as such, we

used the MCC in Nigeria, MNC for the most used cellular network in Nigeria and the LAC of the BTSs in the region.

3.3. Base station spoofing

With the spoofed BTS, it is possible to catch the IMSI (international mobile subscriber identity) of the victims (neighboring mobile users of the spoofed network). Acquiring the IMSI from the victim with the rogue station is a breeze as long as the victim can be tricked into authenticating with the rogue network. Using the command *tmsis* in the command line interface of OpenBTS, the list of authenticated users is shown in the console.

3.4. GSM Packet Sniffing and Decoding

GSM sniffing is even cheaper to conduct than a rogue BTS because it is possible to sniff GSM packets with a couple 100 dollars of hardware. With gr-GSM, the ARFCN that the victim connected to is given with the command grgsm_scanner. With the ARFCN, we were able to calculate the downlink and uplink frequency of the base station. To capture all the packets sent to the local loopback interface, we ran the command grgsm_livemon.py, a utility that gives an interactive window [28] with Wireshark to monitor data from the BCCH. Traffic is piped to the system's loopback interface inside UDP packets on port 4729 [29].

The program grgsm_capture.py captures the GSM signal to a file that is processed by grgsm_decode. Often the GSM traffic will hop between different channels during a transmission. These hops occur quite quickly, every few milliseconds and so in practice, this normally means that to capture more than the base station's broadcast control channel (BCCH) there is a need to simultaneously capture multiple channels at once. With a high instantaneous bandwidth SDR like the USRP this was achieved but, it is important to note that with cheap SDR like RTLSDR whose bandwidth is no more than 3 MHz, a wideband capture built through a multi-SDR was used [15], [30]. To determine the requirement for the setup to capture the GSM packets, we decode the BCCH traffic to obtain the "Hopping Channel" information whose values are either "Yes" or "No." The former implies the communication is done in one channel while the latter implies the base station communicates with the ME via multiple channels. The channels used in channel hopping are contained in the list of ARFCN channels. If hopping is turned off, then getting the encrypted packets is as simple as re-running grgsm_decode with the mode set to SDCCH8 and the timeslot from the "Immediate Assignment" packet. Then the encrypted packets are shown in Wireshark.

We did struggle with de-hopping the captured traffic when hopping is turned on but eventually, we found sources that use a GRC (GNU Radio Companion) script [30], [31]. The script can be used for as many channels as used by the base station for hopping by setting the channelized files using **file source** and **GSM Adapters** blocks using the "List of ARFCN", "Training Sequence", "Timeslot", "Hopping Channel MAIO" and "HSN" in the "Immediate Assignment" packet and other parameters to inform grGSM how the bytes in the several channels are pieced together.

3.5. A5/1 Decryption

The most common encryption used is the A5/1, and several attacks have been successful on this ciphering design. The most notable attack is the decryption of A5/1 with Kraken, a C++ software developed by Frank Stevenson [32]. It has the potential to decrypt the A5/1 ciphering algorithm in seconds. It is used with a large lookup table of 2 Terabytes size called the Berlin A5/1 rainbow table. We used a hard disk of 3TB to store the tables in a partition accessible by the software.

As discussed in the introduction section, there is a session key (ciphering key, K_c) used to encrypt calls and SMS traffic that are generated by combining a random number (RAND) sent by the network to the ME with the private key (subscriber authentication key, K_i). The session key, as the name implies, should be frequently changed to encrypt the traffic but the key can be repeatedly used across frames. Typically, a channel assignment procedure between the base station and the ME is as follows:

i. The base station sends a “Paging Request” to the ME (only with the downlink frequency)
ii. The ME sends a “Channel Request” to the base station.
iii. The base station sends an “Immediate Assignment” to the ME
iv. Communication continues a dedicated channel.

The encrypted data in GSM is a 57-byte packet chopped down to four bursts of 114 bits each. When cracking the session key, we start with a burst and go through the following steps:

i. Predict what one of the bursts would be.
ii. Determine the plain text of the guessed burst since the encryption is an XOR operation.
iii. Find the encryption keystreams with the guessed encrypted message and the corresponding plain (or clear) text that is fed to Kraken to find the session key.

Guessing an encrypted burst might seem like a difficult task but Karsten Nohl has shown that a lot of the bursts are predictable and repeat in certain intervals such as an empty ack message after “Assignment Complete,” an empty ack message after “Alerting,” “Connect Acknowledgement” and so on [33]. A commonly known message to use is the “System Information Type 5” message which is repeated every 102 frames. Thus, we can find the corresponding encrypted burst for the known clear text and compute the keystreams. In this task, we used Air probe to decode the packets, GSMFramecoder to encode the guessed GSM burst (“System Information Type 5”) and Kraken to crack the ciphering algorithm.

The session key can also be obtained from the ME for most phones [30], [34], [35]. This can be done in one of two ways; either by interfacing with the modem or by talking to the SIM directly with a card reader. The method of direct access to the modem is highly dependent on the type of phone and it is done by sending raw AT commands with the given modem serial number. The serial number is sometimes exposed over USB or through root access to the /dev/ directory. Using a card reader is simpler and tools like Cardpeek or SIMspy II [34], [35] can get the job done.

3.6. Denial of Service

The denial-of-service attacks on a single user are very possible in the GSM protocol. A victim that authenticates with the rogue BTS can be issued commands to shut down by the attacker. One such command is the phone stolen flag.

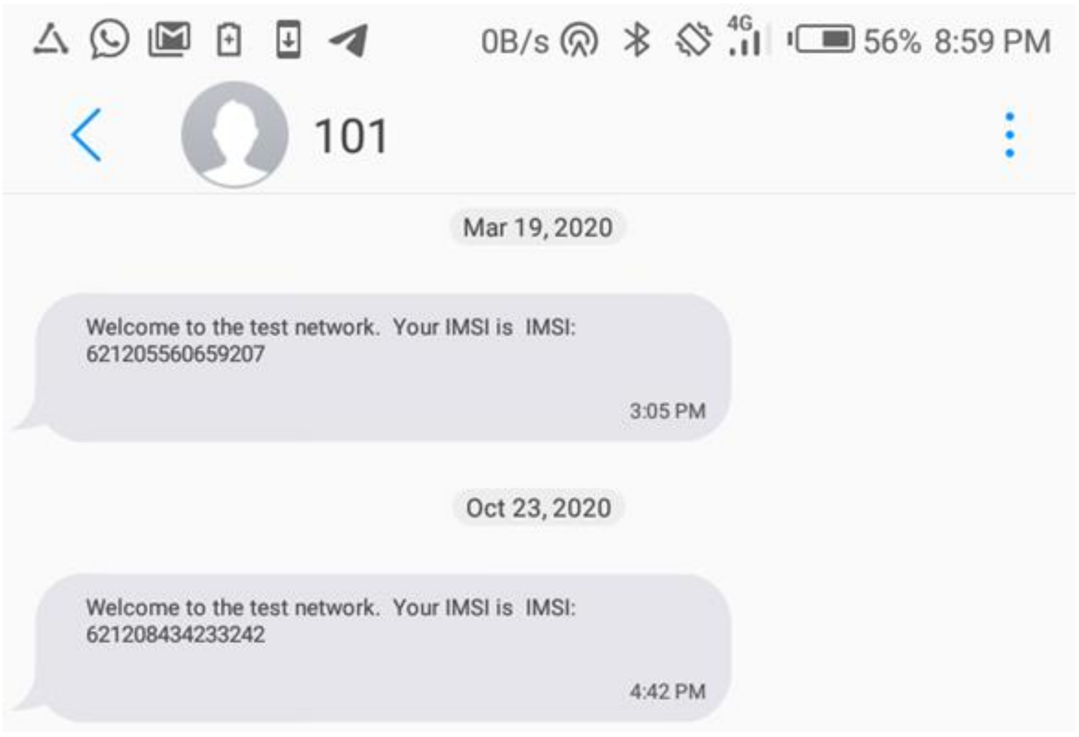
In summary, a completely planned attack on an ME through the successful attacks demonstrated in this work can be presented in a series of steps. First, obtaining the IMSI/TMSI of the victim using the spoofed base station, jamming the higher communication standards (3G, 4G and 5G), capturing packets in all base stations in the neighbourhood of the victim, sniffing, decoding and decrypting the captured SMS and voice packets of the victim using the obtained IMSI/TMSI if the intended attack is eavesdropping [30]. However, if the intended attack is DoS, then the victim can be shut down by the rogue BTS.

4. Results

The steps that were taken in the implementation of a rogue network to carry out an active attack of catching the IMSI of the victim and the passive packet sniffing with open-source penetration tools like the gr-GSM, Air probe Wireshark, A5/1 rainbow table for cryptoanalysis and Kraken have been discussed in the previous section. Hence, the results obtained in this paper will be presented following the steps discussed.

4.1. Base station spoofing and IMSI catching.

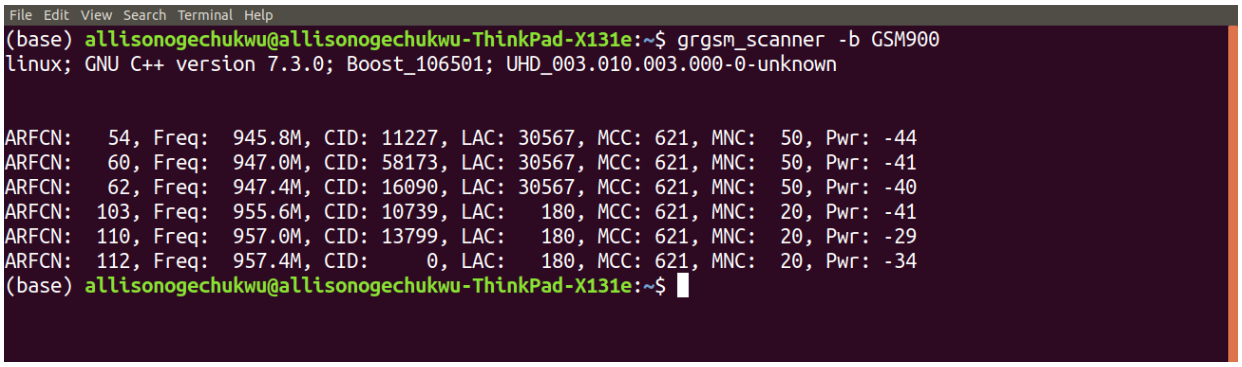
To show the successful implementation of a rogue BTS operating in a legitimate network provider; we sent a welcome message from the rouge BTS to the victim showing the caught IMSI as shown in Fig. 1. Fig. 2 is the list of neighbouring ARFCNs in the region of the attack in the GSM900 band. This GSM scan was conducted with the GR-GSM program at a gain of 40dB with an RTL-SDR dongle. It shows that ARFCN 110 is the channel with the highest power in the list.



Welcome to the test network. Your IMSI is IMSI:
621205560659207

Welcome to the test network. Your IMSI is IMSI:
621208434233242

Fig 1 SMS message from rogue BTS to victim showing the caught IMSI.



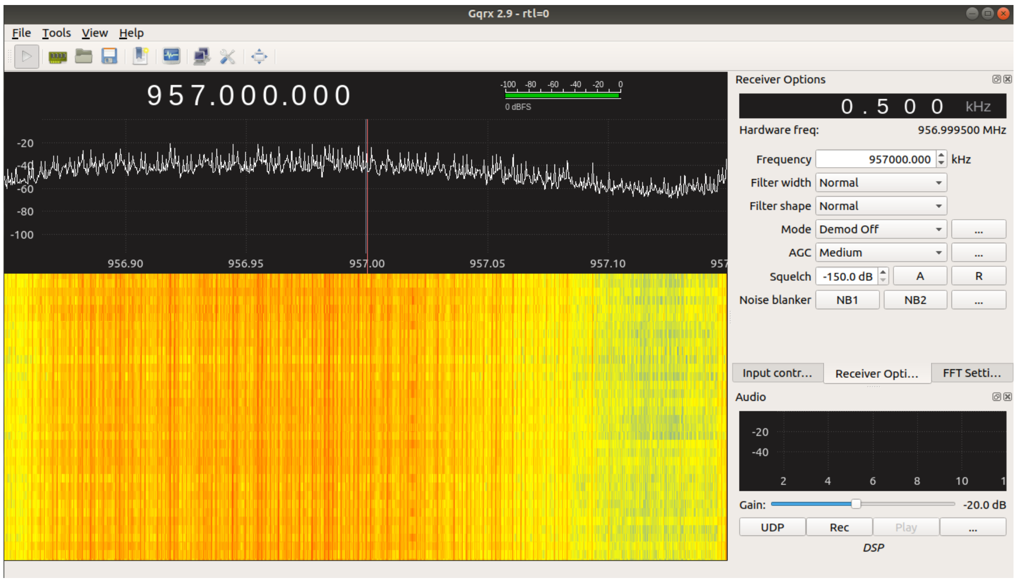
```
(base) allisonogechukwu@allisonogechukwu-ThinkPad-X131e:~$ grgsm_scanner -b GSM900
linux; GNU C++ version 7.3.0; Boost_106501; UHD_003.010.003.000-0-unknown


ARFCN:   54, Freq:  945.8M, CID: 11227, LAC: 30567, MCC: 621, MNC:  50, Pwr: -44
ARFCN:   60, Freq:  947.0M, CID: 58173, LAC: 30567, MCC: 621, MNC:  50, Pwr: -41
ARFCN:   62, Freq:  947.4M, CID: 16090, LAC: 30567, MCC: 621, MNC:  50, Pwr: -40
ARFCN:  103, Freq:  955.6M, CID: 10739, LAC:   180, MCC: 621, MNC:  20, Pwr: -41
ARFCN:  110, Freq:  957.0M, CID: 13799, LAC:   180, MCC: 621, MNC:  20, Pwr: -29
ARFCN:  112, Freq:  957.4M, CID:     0, LAC:   180, MCC: 621, MNC:  20, Pwr: -34
(base) allisonogechukwu@allisonogechukwu-ThinkPad-X131e:~$ 
```

Fig 2 List of base stations in the area using grgsm_scanner.

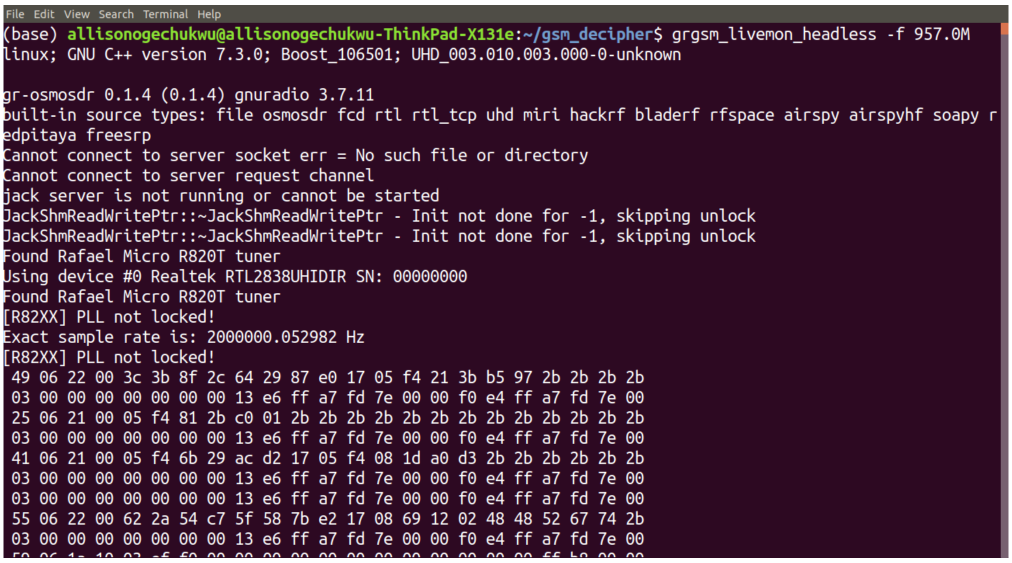
4.2. GSM Sniffing and decoding

The victim's ARFCN used for this challenge was 110 whose downlink and uplink frequencies are 957 MHz and 912 MHz, respectively. The accompanying commands that were run as discussed in the methodology are shown in Fig. 4 to 11. Fig. 4 shows the command to capture GSM packets from the victim and save them to a file. The BCCH signal is shown in Fig. 3.



Hardware freq: 956.999500 MHz
Frequency 957000.000 kHz
Filter width Normal
Filter shape Normal
Mode Demod Off
AGC Medium
Squelch -150.0 dB
Noise blanker NB1 NB2

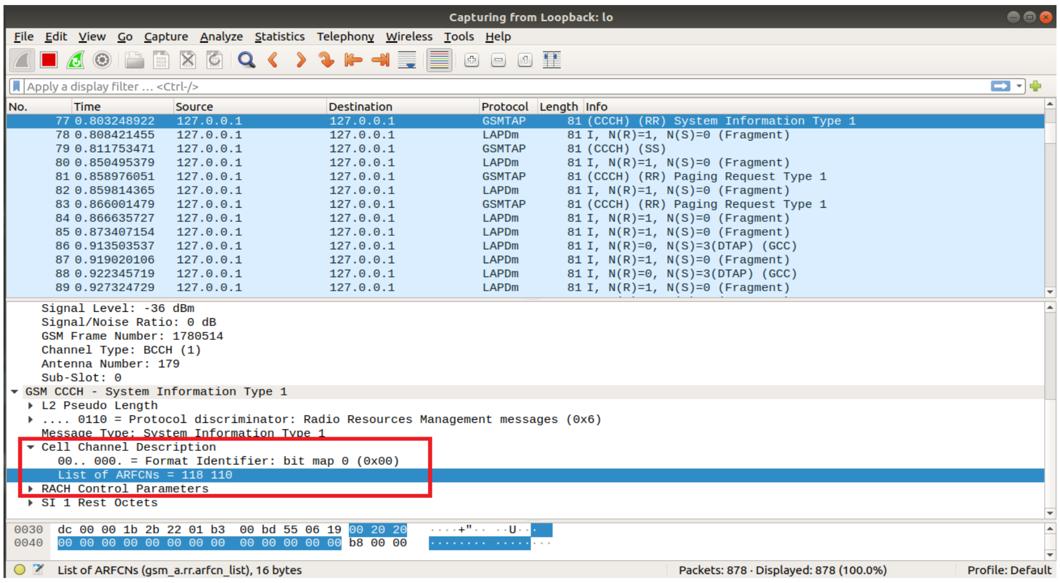
Fig 3 BCCH signal captured on ARFCN 110



```
(base) allisonogechukwu@allisonogechukwu-ThinkPad-X131e:~/gsm_decipher$ grgsm_livemon_headless -f 957.0M
linux; GNU C++ version 7.3.0; Boost_106501; UHD_003.010.003.000-0-unknown

gr-osmosdr 0.1.4 (0.1.4) gnuradio 3.7.11
built-in source types: file osmosdr fcd rtl rtl_tcp uhd miri hackrf bladerf rfspace airspy airspyhf soapy r
edpitaya freesrp
Cannot connect to server socket err = No such file or directory
Cannot connect to server request channel
jack server is not running or cannot be started
JackShmReadWritePtr::~JackShmReadWritePtr - Init not done for -1, skipping unlock
JackShmReadWritePtr::~JackShmReadWritePtr - Init not done for -1, skipping unlock
Found Rafael Micro R820T tuner
Using device #0 Realtek RTL2838UHIDIR SN: 00000000
Found Rafael Micro R820T tuner
[R82XX] PLL not locked!
Exact sample rate is: 2000000.052982 Hz
[R82XX] PLL not locked!
 49 06 22 00 3c 3b 8f 2c 64 29 87 e0 17 05 f4 21 3b b5 97 2b 2b 2b
 03 00 00 00 00 00 00 00 13 e6 ff a7 fd 7e 00 00 f0 e4 ff a7 fd 7e 00
 25 06 21 00 05 f4 81 2b c0 01 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b
 03 00 00 00 00 00 00 00 13 e6 ff a7 fd 7e 00 00 f0 e4 ff a7 fd 7e 00
 41 06 21 00 05 f4 6b 29 ac d2 17 05 f4 08 1d a0 d3 2b 2b 2b 2b 2b 2b
 03 00 00 00 00 00 00 00 13 e6 ff a7 fd 7e 00 00 f0 e4 ff a7 fd 7e 00
 03 00 00 00 00 00 00 00 13 e6 ff a7 fd 7e 00 00 f0 e4 ff a7 fd 7e 00
 55 06 22 00 62 2a 54 c7 5f 58 7b e2 17 08 69 12 02 48 48 52 67 74 2b
 03 00 00 00 00 00 00 00 13 e6 ff a7 fd 7e 00 00 f0 e4 ff a7 fd 7e 00
```

Fig 4 GSM sniffing at ARFCN 110 to decode beacon packets.


Capturing from Loopback: lo
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>

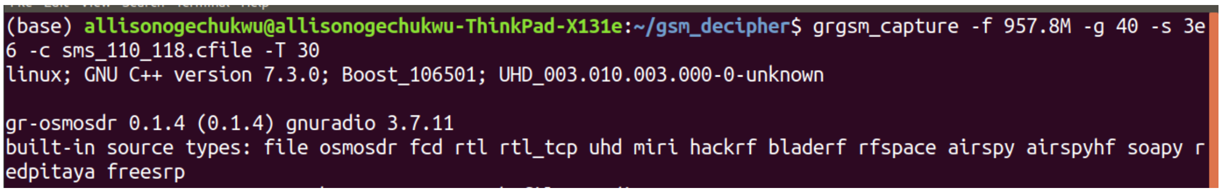
```
No.   Time         Source      Destination  Protocol  Length  Info
   77 0.803248922  127.0.0.1   127.0.0.1    GSMTAP        81  (CCCH) (RR) System Information Type 1
   78 0.808421455  127.0.0.1   127.0.0.1    LAPDm         81  I, N(R)=1, N(S)=0 (Fragment)
   79 0.811753471  127.0.0.1   127.0.0.1    GSMTAP        81  (CCCH) (SS)
   80 0.850495379  127.0.0.1   127.0.0.1    LAPDm         81  I, N(R)=1, N(S)=0 (Fragment)
   81 0.858976051  127.0.0.1   127.0.0.1    GSMTAP        81  (CCCH) (RR) Paging Request Type 1
   82 0.859814365  127.0.0.1   127.0.0.1    LAPDm         81  I, N(R)=1, N(S)=0 (Fragment)
   83 0.866001479  127.0.0.1   127.0.0.1    GSMTAP        81  (CCCH) (RR) Paging Request Type 1
   84 0.866635727  127.0.0.1   127.0.0.1    LAPDm         81  I, N(R)=1, N(S)=0 (Fragment)
   85 0.873407154  127.0.0.1   127.0.0.1    LAPDm         81  I, N(R)=1, N(S)=0 (Fragment)
   86 0.913503537  127.0.0.1   127.0.0.1    LAPDm         81  I, N(R)=0, N(S)=3(DTAP) (GCC)
   87 0.919020106  127.0.0.1   127.0.0.1    LAPDm         81  I, N(R)=1, N(S)=0 (Fragment)
   88 0.922345719  127.0.0.1   127.0.0.1    LAPDm         81  I, N(R)=0, N(S)=3(DTAP) (GCC)
   89 0.927324729  127.0.0.1   127.0.0.1    LAPDm         81  I, N(R)=1, N(S)=0 (Fragment)

    Signal Level: -36 dBm
    Signal/Noise Ratio: 0 dB
    GSM Frame Number: 1780514
    Channel Type: BCCH (1)
    Antenna Number: 179
    Sub-Slot: 0
▾ GSM CCCH - System Information Type 1
  ▸ L2 Pseudo Length
  ▸ .... 0110 = Protocol discriminator: Radio Resources Management messages (0x6)
    Message Type: System Information Type 1
  ▾ Cell Channel Description
      00.. 000. = Format Identifier: bit map 0 (0x00)
      List of ARFCNs = 118 110
  ▸ RACH Control Parameters
  ▸ SI 1 Rest Octets

0030  dc 00 00 1b 2b 22 01 b3  00 bd 55 06 19 00 20 20   ····+"·· ··U····
0040  00 00 00 00 00 00 00 00  00 00 00 00 00 b8 00 00   ········ ········

List of ARFCNs (gsm_a.rr.arfcn_list), 16 bytes     Packets: 878 · Displayed: 878 (100.0%)     Profile: Default
```

Fig 5 GSM sniffing at ARFCN 110 using Wireshark to analyse the packets showing the list of ARFCNs used when channel hopping mode is used



```
(base) allisonogechukwu@allisonogechukwu-ThinkPad-X131e:~/gsm_decipher$ grgsm_capture -f 957.8M -g 40 -s 3e
6 -c sms_110_118.cfile -T 30
linux; GNU C++ version 7.3.0; Boost_106501; UHD_003.010.003.000-0-unknown

gr-osmosdr 0.1.4 (0.1.4) gnuradio 3.7.11
built-in source types: file osmosdr fcd rtl rtl_tcp uhd miri hackrf bladerf rfspace airspy airspyhf soapy r
edpitaya freesrp
```

Fig 6 grgsm_capture command to capture GSM packets wideband

Fig. 7 shows the command used to decode the BCCH traffic from our captured file by setting the sample rate to 1MS/s with the timeslot set to zero. The timeslot information at the start of decoding is usually assumed so at the first attempt of decoding one can check the information provided by the "Immediate Assignment" packets in Wireshark for the appropriate time slot value which will serve as the parameter for the next attempt as shown in Fig 8. As stated earlier, if the Hopping Channel mode is used then the list of ARFCNs will determine the channels used for channel hopping and how the setup for capturing the packets will be conducted. In our case, we had two ARFCNs 110 and 118 as shown in Fig. 5. We configured the bandwidth (or sample rate) and centre frequency in between the two channels of interest whose downlink frequencies of 110 and 118 are 957 MHz and 958.6 MHz, respectively.

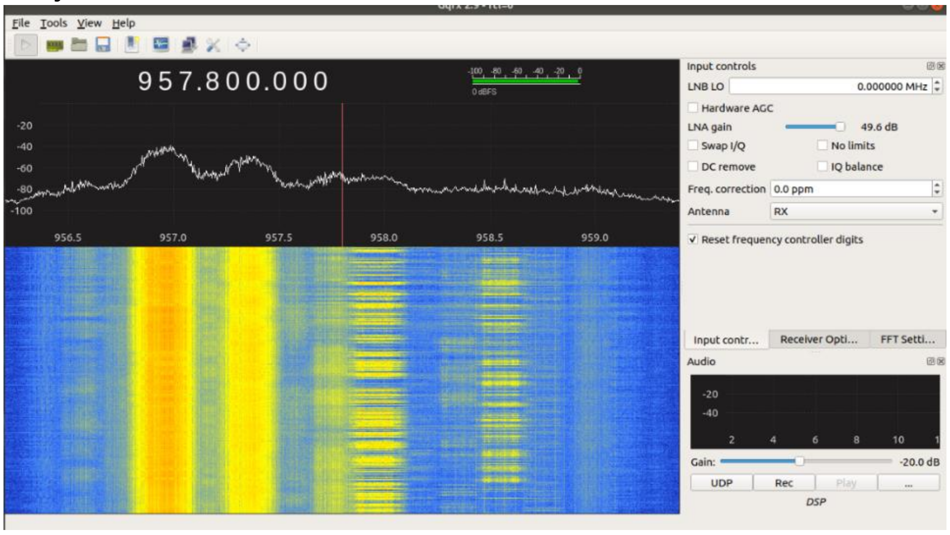
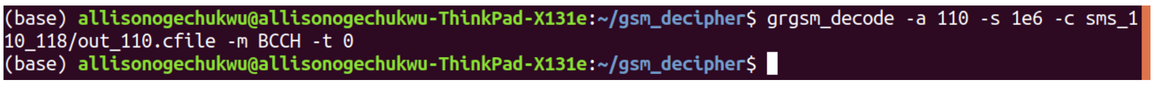
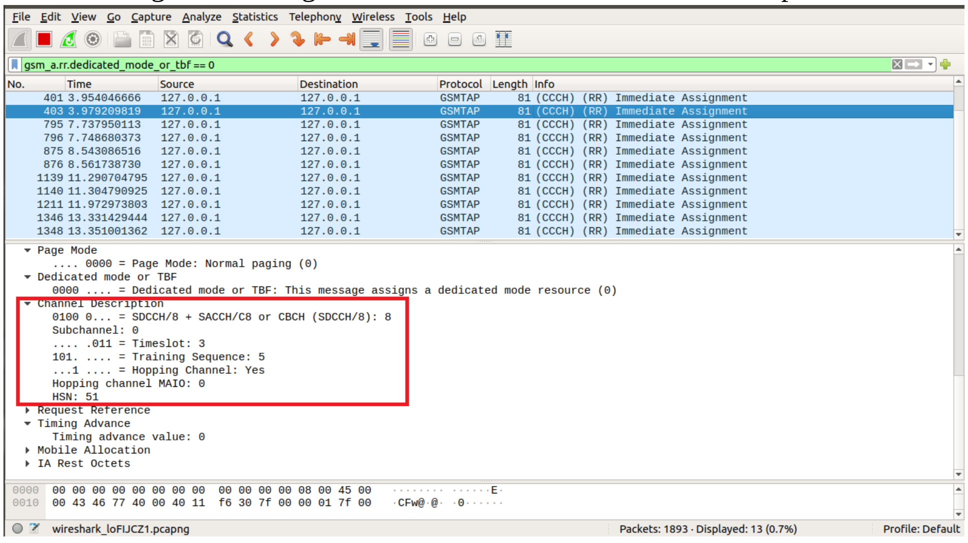


Fig 7 Wideband sniffing in GSM band showing control channel at ARFCN 110 and the hopping channels



```
(base) allisonogechukwu@allisonogechukwu-ThinkPad-X131e:~/gsm_decipher$ grgsm_decode -a 110 -s 1e6 -c sms_1
10_118/out_110.cfile -m BCCH -t 0
(base) allisonogechukwu@allisonogechukwu-ThinkPad-X131e:~/gsm_decipher$
```

Fig 8 BCCH signal decode on channelized wideband capture



```
gsm_a.rr.dedicated_mode_or_tbf == 0

No.  Time          Source      Destination  Protocol  Length Info
 401 3.954046666   127.0.0.1   127.0.0.1    GSMTAP        81 (CCCH) (RR) Immediate Assignment
 403 3.979209819   127.0.0.1   127.0.0.1    GSMTAP        81 (CCCH) (RR) Immediate Assignment
 795 7.737950113   127.0.0.1   127.0.0.1    GSMTAP        81 (CCCH) (RR) Immediate Assignment
 796 7.748680373   127.0.0.1   127.0.0.1    GSMTAP        81 (CCCH) (RR) Immediate Assignment
 875 8.543086516   127.0.0.1   127.0.0.1    GSMTAP        81 (CCCH) (RR) Immediate Assignment
 876 8.561738730   127.0.0.1   127.0.0.1    GSMTAP        81 (CCCH) (RR) Immediate Assignment
1139 11.290704795  127.0.0.1   127.0.0.1    GSMTAP        81 (CCCH) (RR) Immediate Assignment
1140 11.304790925  127.0.0.1   127.0.0.1    GSMTAP        81 (CCCH) (RR) Immediate Assignment
1211 11.972973803  127.0.0.1   127.0.0.1    GSMTAP        81 (CCCH) (RR) Immediate Assignment
1346 13.331429444  127.0.0.1   127.0.0.1    GSMTAP        81 (CCCH) (RR) Immediate Assignment
1348 13.351001362  127.0.0.1   127.0.0.1    GSMTAP        81 (CCCH) (RR) Immediate Assignment

▾ Page Mode
    .... 0000 = Page Mode: Normal paging (0)
▾ Dedicated mode or TBF
    0000 .... = Dedicated mode or TBF: This message assigns a dedicated mode resource (0)
▾ Channel Description
    0100 0... = SDCCH/8 + SACCH/C8 or CBCH (SDCCH/8): 8
    Subchannel: 0
    .... .011 = Timeslot: 3
    101. .... = Training Sequence: 5
    ...1 .... = Hopping Channel: Yes
    Hopping channel MAIO: 0
    HSN: 51
▸ Request Reference
▾ Timing Advance
    Timing advance value: 0
▸ Mobile Allocation
▸ IA Rest Octets

0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ········ ······E·
0010  00 43 46 77 40 00 40 11  f6 30 7f 00 00 01 7f 00   ·CFw@·@· ·0······

wireshark_loFIJCZ1.pcapng          Packets: 1893 · Displayed: 13 (0.7%)          Profile: Default
```

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Fig 9 Decoded packets of the BCCH signal showing the timeslot information and hopping channel

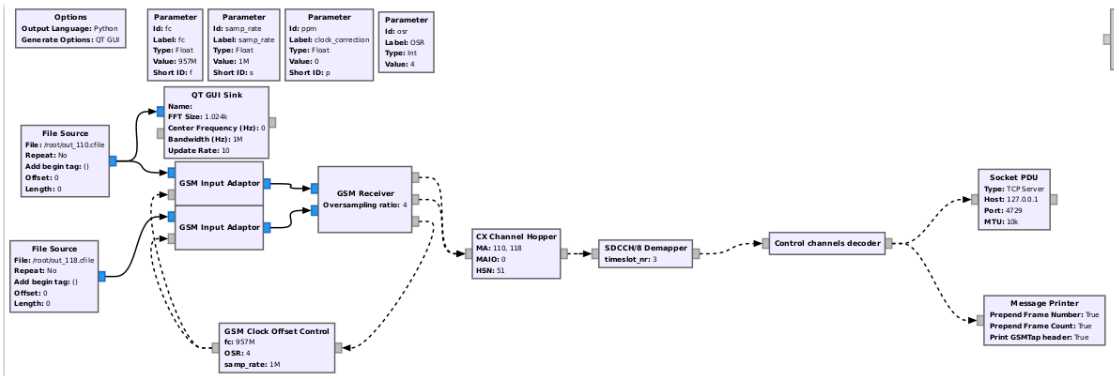
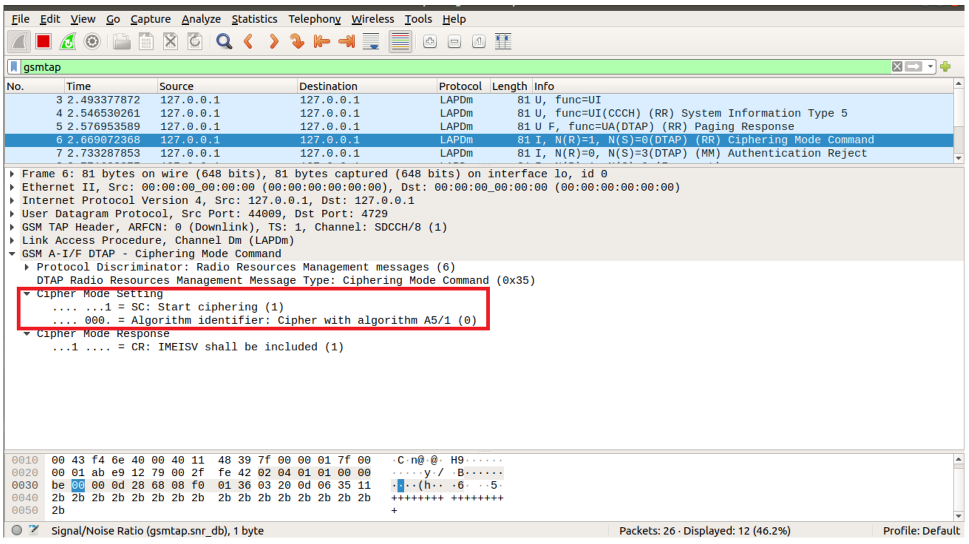


Fig 10 SDCCH signal decode when Hopping Channel mode is used by ME


File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
gsmtap

```
No.     Time         Source      Destination  Protocol Length Info
      3 2.493377872  127.0.0.1   127.0.0.1    LAPDm        81 U, func=UI
      4 2.546530261  127.0.0.1   127.0.0.1    LAPDm        81 U, func=UI(CCCH) (RR) System Information Type 5
      5 2.576953589  127.0.0.1   127.0.0.1    LAPDm        81 U F, func=UA(DTAP) (RR) Paging Response
      6 2.669072368  127.0.0.1   127.0.0.1    LAPDm        81 I, N(R)=1, N(S)=0(DTAP) (RR) Ciphering Mode Command
      7 2.733287853  127.0.0.1   127.0.0.1    LAPDm        81 I, N(R)=0, N(S)=3(DTAP) (MM) Authentication Reject

Frame 6: 81 bytes on wire (648 bits), 81 bytes captured (648 bits) on interface lo, id 0
Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
User Datagram Protocol, Src Port: 44009, Dst Port: 4729
GSM TAP Header, ARFCN: 0 (Downlink), TS: 1, Channel: SDCCH/8 (1)
Link Access Procedure, Channel Dm (LAPDm)
GSM A-I/F DTAP - Ciphering Mode Command
  Protocol Discriminator: Radio Resources Management messages (6)
  DTAP Radio Resources Management Message Type: Ciphering Mode Command (0x35)
  Cipher Mode Setting
    .... ...1 = SC: Start ciphering (1)
    .... 000. = Algorithm identifier: Cipher with algorithm A5/1 (0)
  Cipher Mode Response
    ...1 .... = CR: IMEISV shall be included (1)

0010  00 43 f4 6e 40 00 40 11  48 39 7f 00 00 01 7f 00   ·C·n@·@· H9······
0020  00 01 ab e9 12 79 00 2f  fe 42 02 04 01 01 00 00   ·····y·/ ·B······
0030  be 00 00 0d 28 68 08 f0  01 36 03 20 0d 06 35 11   ····(h·· ·6· ··5·
0040  2b 2b 2b 2b 2b 2b 2b 2b  2b 2b 2b 2b 2b 2b 2b 2b   ++++++++ ++++++++
0050  2b                                                 +

Signal/Noise Ratio (gsmtap.snr_db), 1 byte          Packets: 26 · Displayed: 12 (46.2%)          Profile: Default
```

Fig 11 Decoded packets from the captured data showing the ciphering mode used which is A5/1

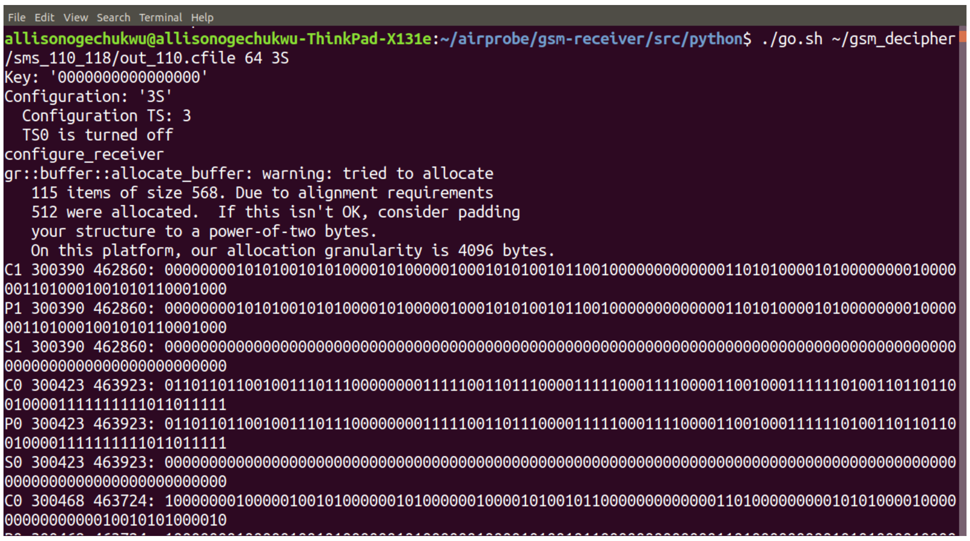
4.3. A5/1 Decryption

Finally, to get the encrypted packets of the captured data we need to decrypt the generated decoded data. To accomplish this, we need the used ciphering mode. In our case, it was A5/1 as shown in Fig. 10. We can obtain the burst with the decoded GSM packets as shown in Fig. 12. It is generated with the tool $go.sh$ in the Airprobe software. The generated bursts contain the encrypted and decrypted bursts, and keystreams. Table 1 shows the available configurations supported by Airprobe.

Table 1 Available configuration for Airprobe decode tool

Option	Description
0C	Timeslot0 "Combined configuration ", with SDCCH/4 (FCC + SCH + BCCH + CCCH + SDCCH/4)
0B	TS0 "FCCH + SCH + BCCH + CCCH"
xS	TSx SDCCH/8 (x is the timeslot given in the "Immediate Assignment")
1T	TS2 (Full Rate) Traffic
1TE	TS1 Enhanced Full Rate Traffic

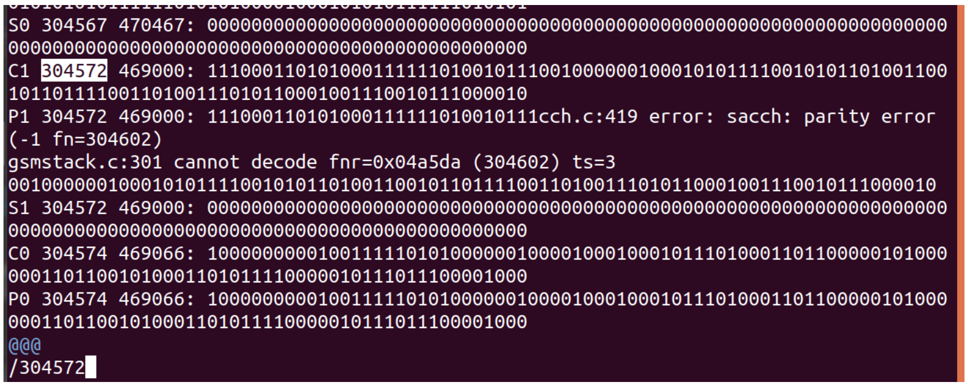
Cx are the encrypted burst (114 bits), Px are the decrypted burst (also 114 bits) and Sx are the keystreams (encrypted bits XOR decrypted bits). Besides Cx, Px and Sx, there is the GSM frame number and the modified frame number as required by the A5/1 algorithm.



```
File Edit View Search Terminal Help
allisonogechukwu@allisonogechukwu-ThinkPad-X131e:~/airprobe/gsm-receiver/src/python$ ./go.sh ~/gsm_decipher
/sms_110_118/out_110.cfile 64 3S
Key: '0000000000000000'
Configuration: '3S'
  Configuration TS: 3
  TS0 is turned off
configure_receiver
gr::buffer::allocate_buffer: warning: tried to allocate
   115 items of size 568. Due to alignment requirements
   512 were allocated.  If this isn't OK, consider padding
   your structure to a power-of-two bytes.
   On this platform, our allocation granularity is 4096 bytes.
C1 300390 462860: 0000000101010010101000010100000100010101001011001000000000000001101010000101000000000010000
0011010001001010110001000
P1 300390 462860: 0000000101010010101000010100000100010101001011001000000000000001101010000101000000000010000
0011010001001010110001000
S1 300390 462860: 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000
C0 300423 463923: 0110110110010011101110000000011111001101110000111110001111000011001000111111010011011011O
0100001111111111011011111
P0 300423 463923: 0110110110010011101110000000011111001101110000111110001111000011001000111111010011011011O
0100001111111111011011111
S0 300423 463923: 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000
C0 300468 463724: 1000000010000010010100000001010000001000010100101100000000000001101000000000101010000010000
0000000000100101010000010
```

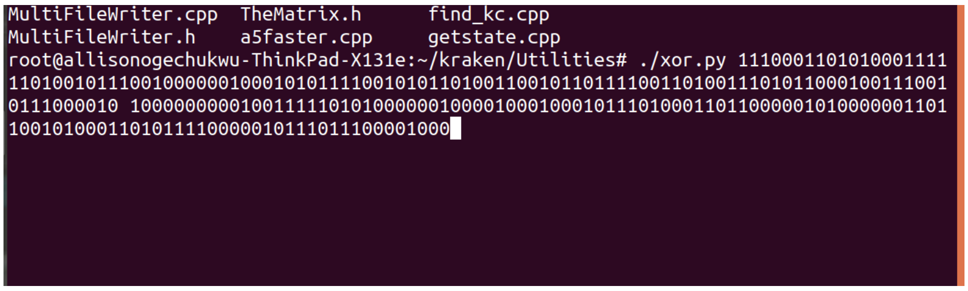
Fig 12 Airprobe decode tool to output burst

The “System Information Type 5” packet had a GSM frame number of 304572 and the following two guessed frame numbers for the encrypted bits were 304674 and 304776. With $gsmframecoder$ and the python utility program in Kraken, we computed the keystreams needed by Kraken as shown in Fig. 14. The Cx and Px bits are obtained for the GSM frame numbers to generate the keystreams; an example of this process is shown in Fig. 13 for the frame number 304572.



```
S0 304567 470467: 0000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000
C1 304572 469000: 1110001101010001111110100101110010000001000101011110010101101001100
101101111001101001110101100010011100101110000010
P1 304572 469000: 111000110101000111111010010111cch.c:419 error: sacch: parity error
(-1 fn=304602)
gsmstack.c:301 cannot decode fnr=0x04a5da (304602) ts=3
001000000100010101111001010110100110010110111100110100111010110001001110010111000010
S1 304572 469000: 0000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000
C0 304574 469066: 1000000000100111110101000000100001000100010111010001101100000101000
000110110010100011010111100000101110111000010000
P0 304574 469066: 1000000000100111110101000000100001000100010111010001101100000101000
000110110010100011010111100000101110111000010000
@@@
/304572
```

Fig 13 Encrypted bursts of the "System Information Type 5"



```
MultiFileWriter.cpp  TheMatrix.h      find_kc.cpp
MultiFileWriter.h    a5faster.cpp     getstate.cpp
root@allisonogechukwu-ThinkPad-X131e:~/kraken/Utilities# ./xor.py 111000110101000111
1101001011100100000010001010111100101011010011001011011110011010011101011000100111001
0111000010 10000000001001111101010000001000010001000101110100011011000001010000001101
100101000110101111000001011101110000100O
```

Fig 14 XOR operation of encrypted burst and encoded burst

This process of trying to guess the known clear text and the corresponding encrypted burst can be a hassle and might be unsuccessful a couple of times. A typical response from Kraken is a two-step process. First, the secret state is found in the burst which is the output of the crack command in Kraken (for example Found a56290409b507d75 @ 37). It is then fed to find_kc utility program which is used to find matches of the session key given the secret state, frame count numbers (derived from the frame numbers) and burst.

5. Discussion

We have shown the possible attacks from base station spoofing to session key attacks with precomputation tables (A5/1 rainbow tables). With base station spoofing, we noticed a high turn-up of users authenticating to the network even when GSM is becoming deprecated. Thus, the fallback of network providers to the 2G network is still viable. The experiment was carried out in one of the universities in Nigeria and most students were Android users using the spoofed network providers. We observed that the stronger the signal from our rogue BTS the more users we had authenticating with our network.

With GSM sniffing and decoding, an attacker can easily access information like the Ciphering Mode, Hopping Channel, Timeslot, etc. of the ME which potentially leads to further decryption of the GSM packets. The decryption of the session key has been shown to be successful within seconds even on accessible computers and laptops due to the lightweight version of the Kraken program [25] called Deka [36]. Also, the ease of obtaining the session key by direct access or the use of a modem has also been demonstrated to work with rooted Android phones, Samsung phones and iPhones [34]. Also, DoS attacks are possible with almost every phone that supports 2G GSM as once authenticated to the rogue network, the victim can be issued commands to permanently shut down. Though in this work and most works, the rogue BTS is mostly used for IMSI catching, this is because the rogue BTS can only operate with inbound calls[3] on the network and may face difficulties connecting with the spoofed legitimate network provider, and in addition, such an attack would be very unethical. However, there is a theoretical approach for decrypting both inbound and outbound calls using a rogue BTS.

For outbound calls, the rogue BTS can be programmed to cause even more harm than just decrypting a session of the traffic to an entire communication done by the victim. Usually, the challenge-response strategy of the GSM would authenticate a user through a RAND which the ME uses along with the K_i to generate an SRES that authenticates the user. So, the rogue BTS with the IMSI of the victim can send the IMSI to the legitimate network and when it receives the challenge from the network, it would pass it to the victim who completes the challenge and returns the response. With the response. The rogue BTS can crack the victim's keystream and recover the session key using the methods discussed; then, it sends the response to the carrier.

5. Conclusion

In this paper, we demonstrated that the attacks on the GSM protocol are still very viable despite its general knowledge. Furthermore, the tools for decrypting GSM packets are not only readily available but are constantly being improved. This means that SMS and voice traffic can be easily targeted. Moreover, when considering other higher technologies, the use of rogue BTS can be effortlessly implemented using open-source tools and USRP devices to function as IMSI catchers. This poses a significant threat to users of these technologies as it eases the task of intercepting and decrypting their communication packets.

Our study aims to bring together the numerous attack methods that are either practically demonstrated or theoretically feasible within the context of the GSM standard with modern-day tools accessible to the general public. [22], our research reveals how individuals with the appropriate tools and knowledge can exploit this one-way authentication vulnerability inherent in the GSM stack. As a result, they can eavesdrop on calls and intercept SMS data, highlighting the pressing need for improved security in GSM technology since it is still a fallback technology.

[3] This is because the rogue BTS is a separate network, so the network provider thinks the ME is off or has no signal.

Abbreviations

GSM	Global Systems for Mobile Communications
SDR	Software-Defined Radio
BTS	Base Transceiver Station
IMSI	International Mobile Subscriber Identity
TMSI	Temporal Mobile Subscriber Identity
ME	Mobile Entity
MNC	Mobile Network Code
MCC	Mobile Country Code
LAC	Local Area Code
SRES	Signed Response
RAND	Random number
ARFCN	Absolute Radio Frequency Channel Number
FCC	Federal Communication Commission
BCCH	Broadcast Control Channel
SDCCH	Standalone Dedicated Control Channel
SCH	Synchronization Channel
CCCH	Common Control Channel
MITM	Man-In-The-Middle
Ki	Authentication key
Kc	Ciphering key
AuC	Authentication Centre
USRP	Universal Software Radio Peripheral

References

[1] K. Nohl, “Attacking phone privacy Attacking phone privacy,” 2010.

[2] G. Liu and D. Jiang, “5G: Vision and Requirements for Mobile Communication System towards Year 2020,” *Chinese Journal of Engineering*, vol. 2016, 2016, doi: 10.1155/2016/5974586.

[3] M. Toorani and A. A. B. Shirazi, “Solutions to the GSM security weaknesses,” in *Proceedings - The 2nd International Conference on Next Generation Mobile Applications, Services, and Technologies, NGMAST 2008*, 2008, pp. 576–581. doi: 10.1109/NGMAST.2008.88.

[4] G. Cattaneo, G. De Maio, P. Faruolo, and U. Ferraro Petrillo, “A Review of Security Attacks on the GSM Standard,” *International Conference on Information and Communication Technology*, 2013.

[5] G. Cattaneo, G. De Maio, and U. Ferraro Petrillo, “Security Issues and Attacks on the GSM Standard: A Review,” *Journal of Universal Computer Science*, vol. 19, pp. 2437–2452, 2013.

[6] M. Matsui, “New Block Encryption Algorithm MISTY,” *Lecture Notes in Computer Science Springer Berlin*, vol. 1267, 1997, doi: https://doi.org/10.1007/BFb0052334.

[7] T. Ulversoy, “Software Defined Radio: Challenges and Opportunities,” *IEEE Communications Surveys & Tutorials*, vol. 12, no. 4, pp. 531–550, 2010, doi: 10.1109/SURV.2010.032910.00019.

[8] J. Dj Golit, “Cryptanalysis of Alleged A5 Stream Cipher,” *Advances in Cryptology*, pp. 239–255, 1997.

[9] K. Nohl and S. Krißler, “Phone with end-to-end encryption-soon needed?,” 2009.

[10] F. Oh, “What Is CUDA NVIDIA Official Blog,” Jan. 2022. https://blogs.nvidia.com/blog/2012/09/10/what-is-cuda-2/

[11] G. W. Lee and J. Hong, “A Comparison of Perfect Table Cryptanalytic Trade-off Algorithms,” 2014.

[12] J. Hong and S. Moon, “Erratum: A comparison of cryptanalytic trade-off algorithms,” *Journal of Cryptology*, vol. 27, no. 1. p. 181, Jan. 2014. doi: 10.1007/s00145-012-9140-7.

[13] J. Hong and S. Moon, “A comparison of cryptanalytic trade-off algorithms,” *Journal of Cryptology*, vol. 26, no. 4, pp. 559–637, 2013, doi: 10.1007/s00145-012-9128-3.

[14] K. Nohl and S. Munaut, “GSM Sniffing,” 2010.

[15] B. Hackerspace, “Camp++ 0x7e0 // GSM signal sniffing for everyone with gr-gsm and Multi-RTL by Piotr Krysik,” Jan. 2017. https://www.youtube.com/watch?v=xnXMKRIqkZ4
[16] F. Casanovas, “Airprobe – setup,” Jan. 2014. https://ferrancasanovas.wordpress.com/2014/01/27/airprobe-setup/
[17] Crazy Danish Hacker, “GSM Cracking: Kraken Install & Test - Software Defined Radio Series #15,” Aug. 2016. https://www.youtube.com/watch?v=UKmpw4gcMSE
[18] E. Barkan, E. Biham, and N. Keller, “Instant ciphertext-only cryptanalysis of GSM encrypted communication,” *Journal of Cryptology*, vol. 21, no. 3, pp. 392–429, Jul. 2008, doi: 10.1007/s00145-007-9001-y.
[19] Chaos Communication Camp, “26C3: GSM: SRSLY?,” 2009. https://fahrplan.events.ccc.de/congress/2009/Fahrplan/track/Hacking/3654.en.html
[20] DEFCON Conference, “DEF CON 18 - Chris Paget - Practical Cell phone Spying.” Nov. 2013. [Online]. Available: https://www.youtube.com/watch?v=fQSu9cBaojc
[21] F. Y. Hansen, *The Hacker’s Hardware Toolkit: best gadgets for Read Team hackers*, 2nd ed. 2019.
[22] Ettus Research, “USRP B210 USB Software Defined Radio (SDR) - Ettus Research,” 2023. https://www.ettus.com/all-products/ub210-kit/
[23] Range Networks, “GitHub - Range Networks/openbts: GSM+GPRS Radio Access Network Node,” 2014. https://github.com/RangeNetworks/openbts (accessed Sep. 23, 2023).
[24] GNU Radio, “GNU Radio Wiki,” 2022. https://wiki.gnuradio.org (accessed Aug. 02, 2023).
[25] K. Nohl, “Wiki - A5/1 Decryption - SRLabs Open-Source Projects,” 2011. https://opensource.srlabs.de/projects/a51-decrypt
[26] A. Knight, *Hacking Connected Cars: Tactics, Techniques and Procedures*, 1st ed. Wiley, 2020. doi: 10.1016/S1353-4858(20)30090-8.
[27] Kali Team, “kalibrate-rtl Kali Linux Tools,” 2023. https://www.kali.org/tools/kalibrate-rtl/
[28] U. Lamping and G. Bloice, “Wireshark Developers Guide,” 2023. https://www.wireshark.org/docs/wsdghtmlchunked/
[29] H. Sand, “A look at GSM,” Feb. 2021. https://harrisonsand.com/posts/gsm-security/
[30] H. Sand, “Decrypting GSM SMS traffic,” Feb. 2021. https://harrisonsand.com/posts/decrypting-gsm/
[31] P.-P. Braun, “Deal with hopping,” 2019. https://pub.nethence.com/radio/hopping
[32] R. McMillan, “New ‘Kraken’ GSM-cracking software is released,” 2010. https://www.computerworld.com/article/2753942/new--kraken--gsm-cracking-software-is-released.html (accessed Sep. 24, 2023).
[33] Christiaan008, “27c3: Wideband GSM Sniffing (en),” Jan. 2011. https://www.youtube.com/watch?v=fHfXSr-FhU
[34] M. Kupka, “Odposlouchávání GSM komunikace - Hacking Lab,” 2021. https://hackinglab.cz/cs/blog/odposlouchavani-gsm-komunikace/
[35] H. L., “Radiohacking - Part 1 ‘GSM,’” Jul. 2022. https://vk.com/@haccking1-radiohaking-chast-1-gsm
[36] M. Parker, “GitHub - prkrmx/decipher: GSM decipher,” 2019. https://github.com/prkrmx/decipher