Baseband Attacks:
Remote Exploitation of Memory Corruptions in Cellular Protocol Stacks

Author: Ralf-Philipp Weinmann
University of Luxembourg
WOOT, USENIX, 2012.

Presenter: Hyuntae Kim

Part 1. Introduction

- GSM overview
  - MS-BTS
- Cellular Baseband Stack
- Contribution

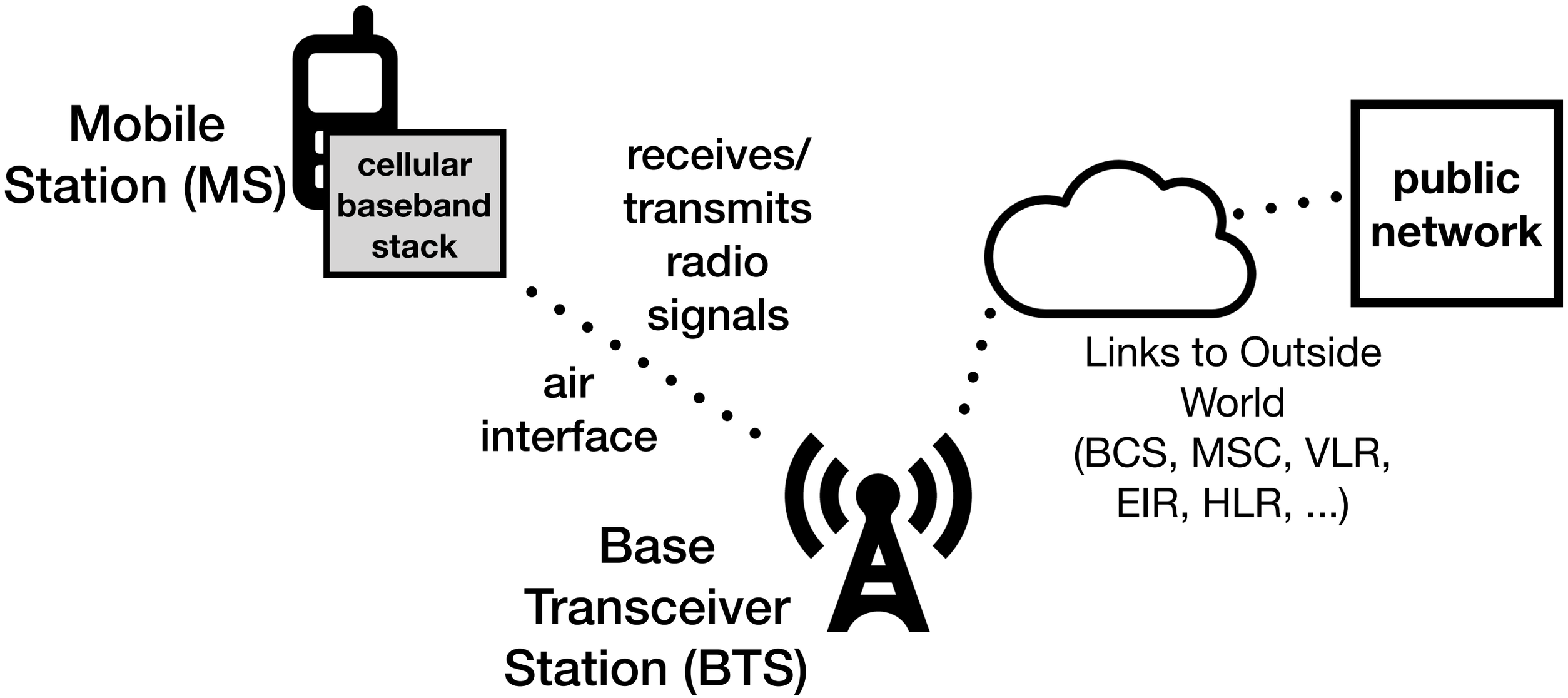
Introduction

GSM Overview

- **G**lobal **S**ystem for **M**obile communications **(GSM)**
  - It is also known as **2G**
- Long Term Evolution (LTE) and UMTS (3G) provide backwards compatible with **GSM**

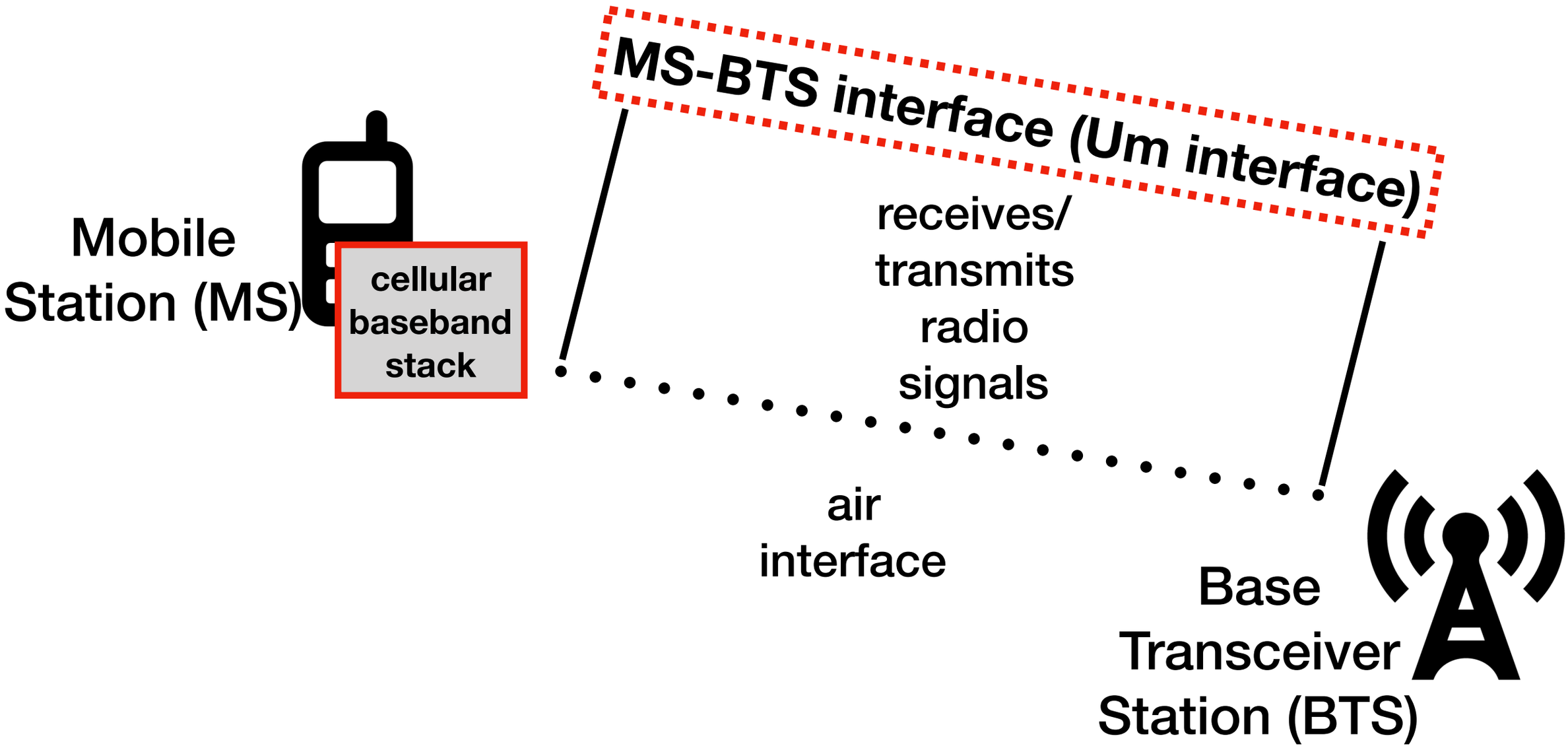
Introduction

GSM Overview



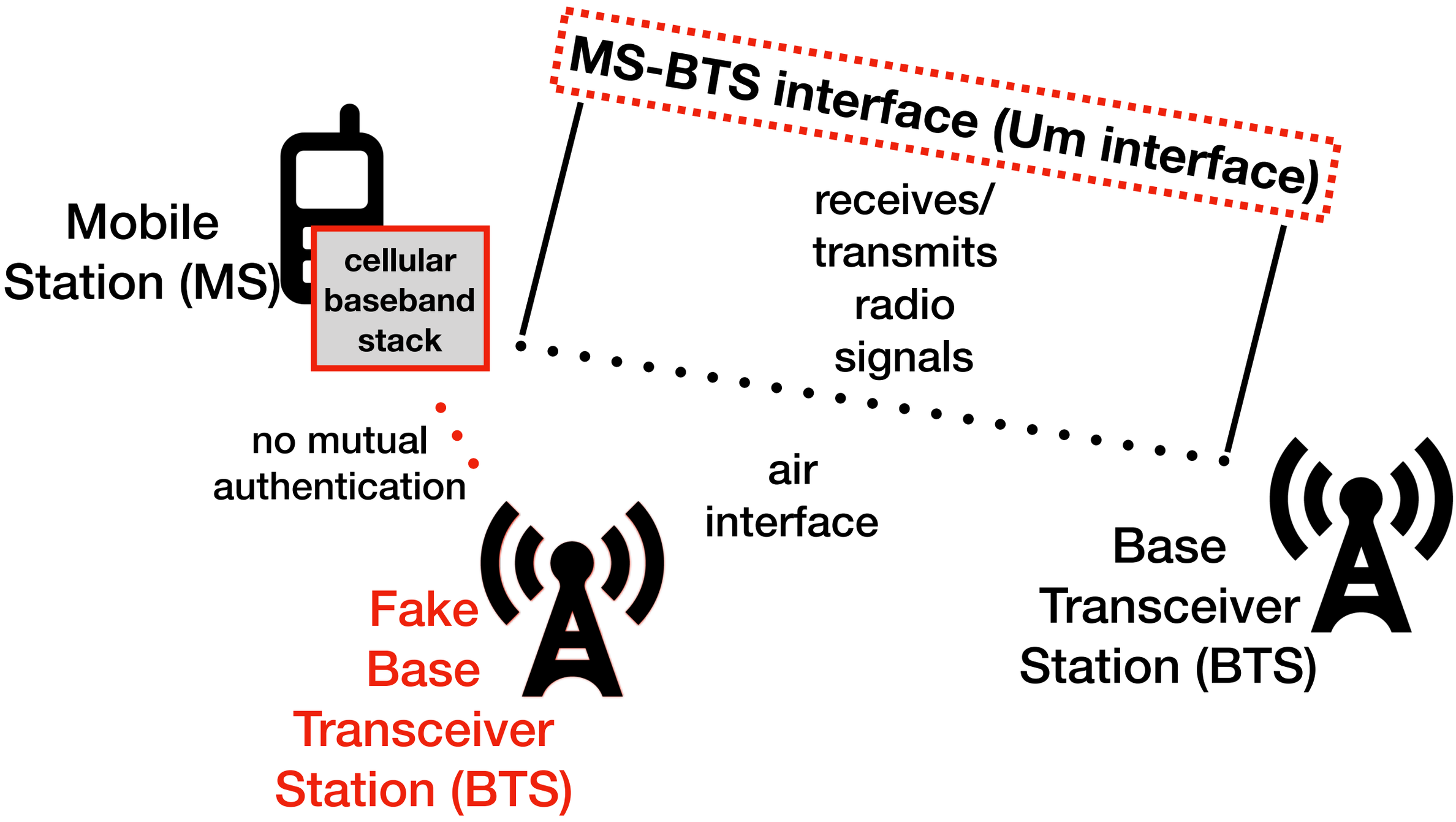
Introduction

GSM Overview - MS-BTS



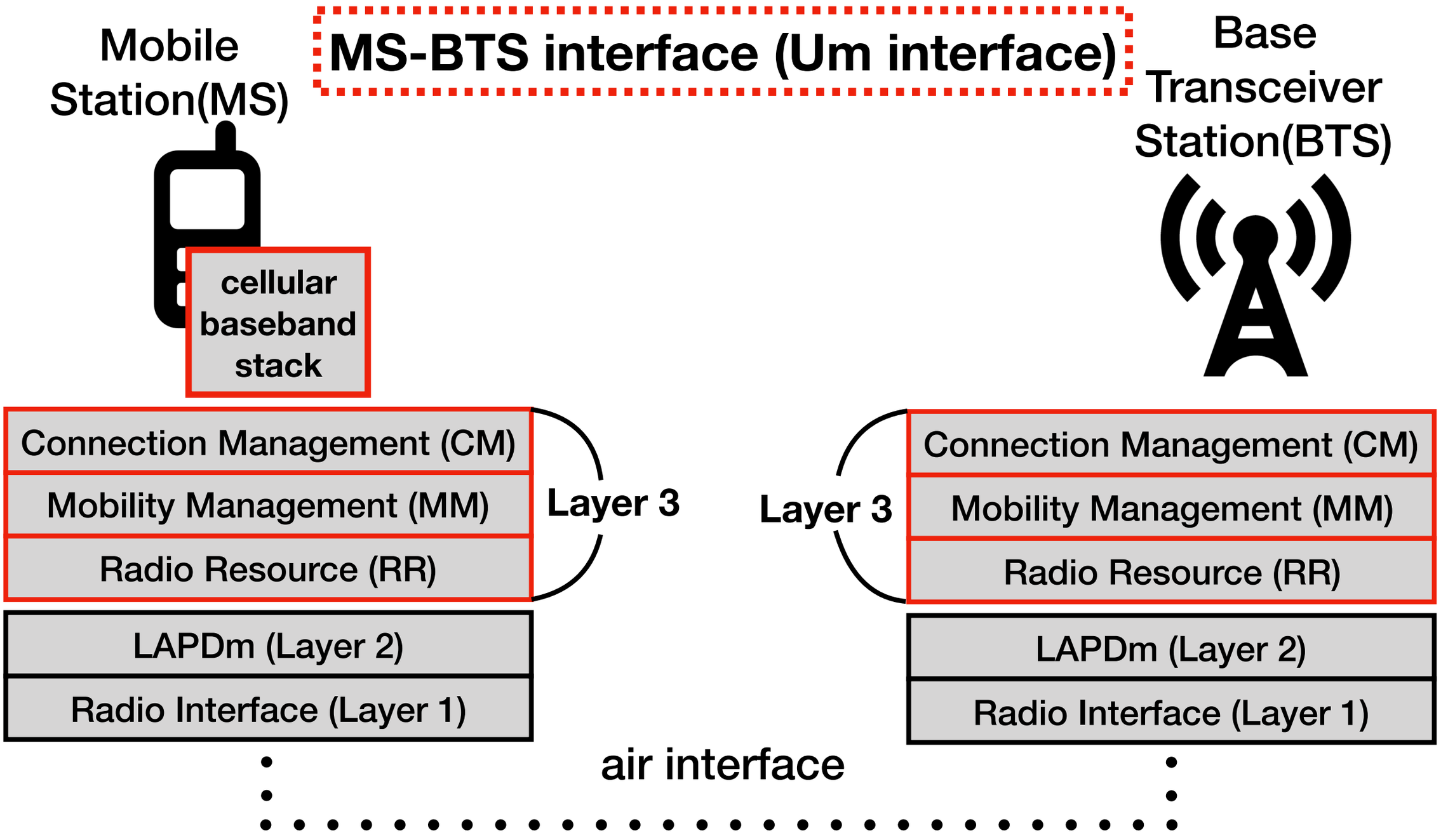
Introduction

GSM Overview - MS-BTS



Introduction

GSM Overview - MS-BTS



Introduction

Cellular Baseband Stack

- It's a part which is embedded in cellular phone
  - It's responsible for radio operations
- Smart phones have at least two CPU
  - Cellular processor (CP) for baseband software
  - Application processor (AP) for user interface and applications

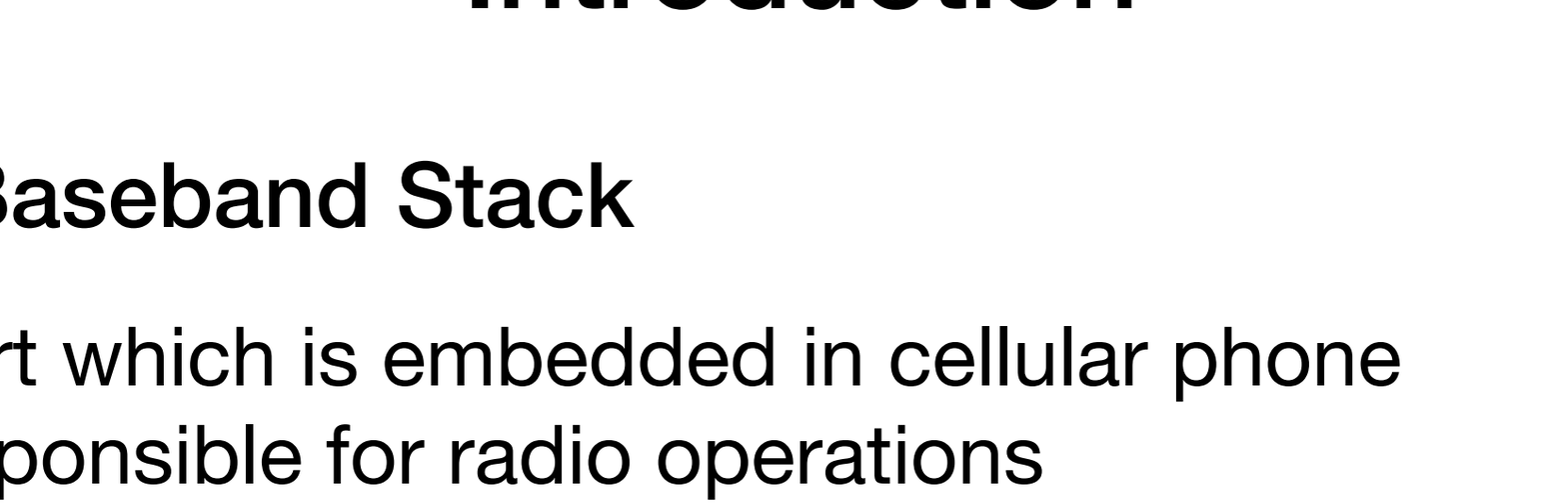


Figure. Qualcomm cellular processor & Intel Infineon baseband processor

Introduction

Cellular Baseband Stack

- It runs on RTOS separately from application processor
  - For radio performance/reliability
  - For government's law

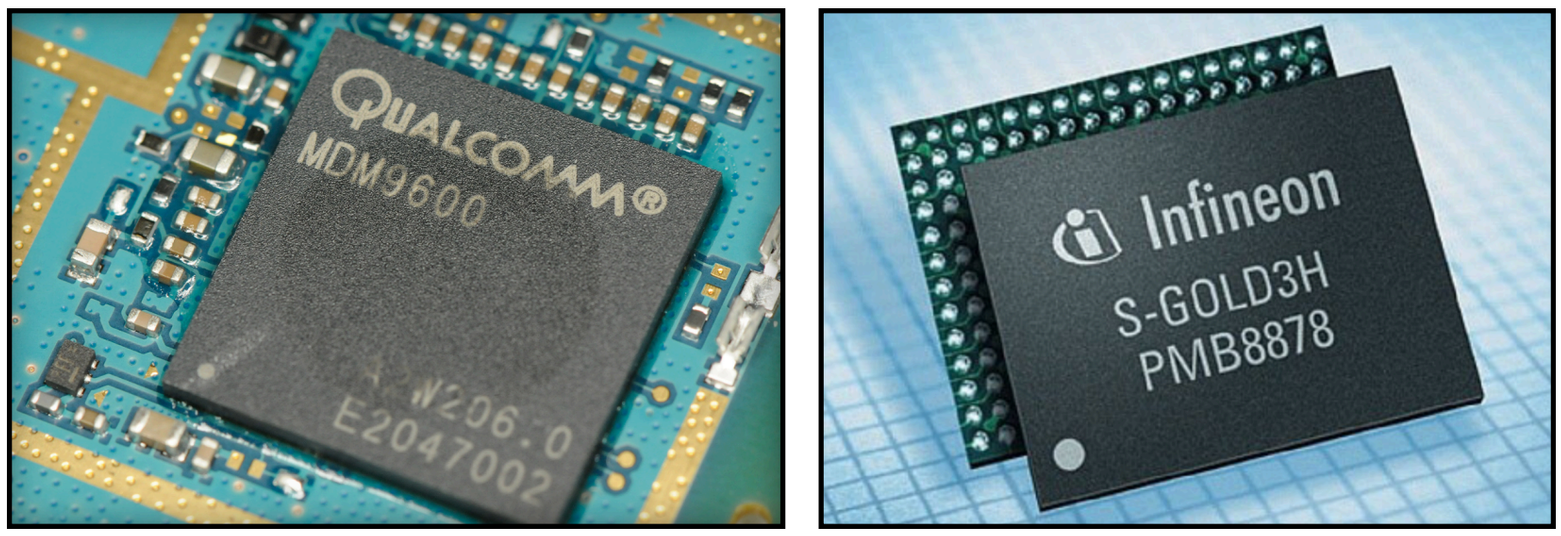


Figure. Qualcomm baseband processor & Intel Infineon baseband processor

Introduction

Contribution

- Author analyzed GSM baseband stacks
  - Mainly iPhone 4 and HTC Dream G1
  - Remotely exploitable memory corruptions are found
    - Due to programming error
- iPhone 4 (Intel infineon baseband)
  - heap-based buffer overflow
- HTC Dream G1 (Qualcomm baseband)
  - stack-based buffer overflow
- Bugs are patched

Part 2. Baseband Security

- Baseband Security Overview
- Layer 3 Message Format

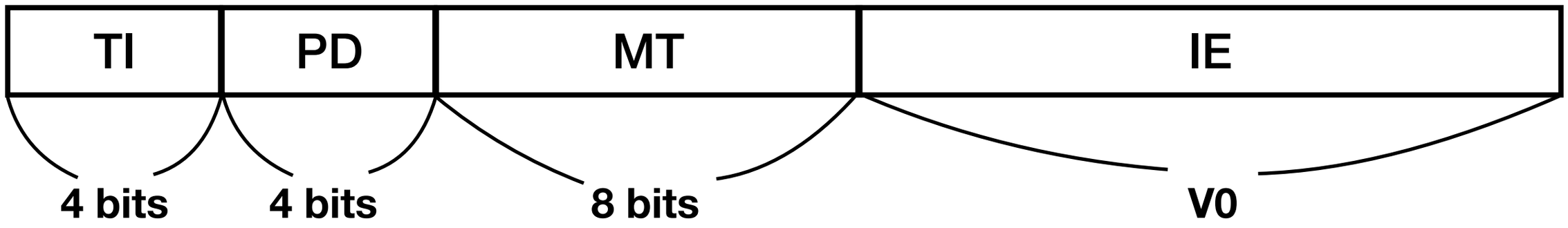
Baseband Security

Baseband Security Overview

- Code-base baseband is introduced in 1990s.
- GSM protocols have many length field
- There's no exploit mitigations
  - Stack canary, heap protection (safe unlink), DEP, ASLR, ...
- Cellular phone/baseband's firmware is not open-source
  - But, in 2004, Vitelcom TSM 30 firmware was leaked
    - It helps to understand GSM baseband stack architecture

Baseband Security

Layer 3 Message Format



- Transaction Identifier (TI)
- Protocol Discriminator (PD)
- **Message Type (MT):** specify message type of given PD
- **Information Elements (IE):** contain information options and data by given **MT**. **V0** is different by **MT** and **IE**'s option
  - IE can be combination of **T**, **L** and **V**. (V, LV, T, TV,TLV)
    - T=tag (1 byte), L=length (1 byte), V=value

Part 3. How to Find Bug

- Targets
- Analysis methods
  - Fuzzing
  - Code auditing
  - Reverse engineering

How to Find Bug

Targets

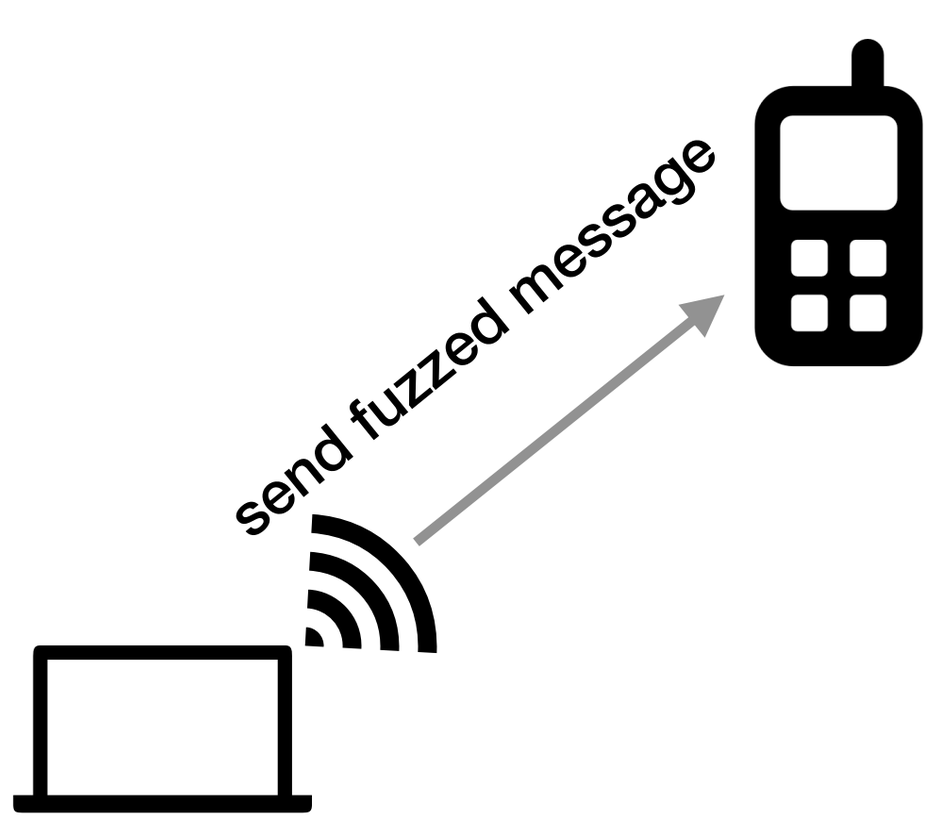
Apple iPhone 4
(Intel Infineon
baseband, iOS)

HTC Dream G1
(Qualcomm
baseband,
Android)

How to Find Bug

Analysis methods - Fuzzing

- Fuzzing
  - From a previous related work, numerous crashes occur leading denial-of-service
  - But there was no easy way to find out whether the crash can lead memory corruption



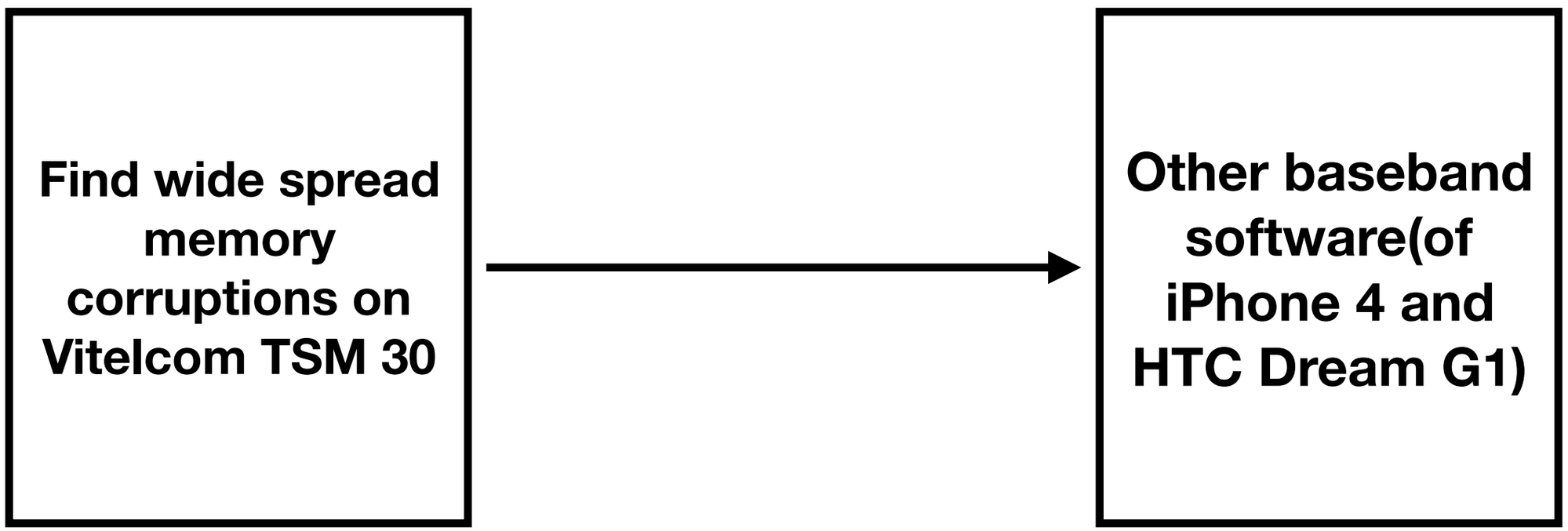
C. Miller and C. Mulliner, Fuzzing the phone in your phone, BlackHat, 2009.

How to Find Bug

Analysis methods - Code auditing

- There's no source code of the targets publicly available
- But there's source tree of Vitelcom TSM 30's firmware

Is there such a kind of memory corruptions in target baseband software?



How to Find Bug

Reverse engineering - Obtaining firmware

- iPhone 4 (iOS)
  - OTA update file
    - It's .ipsw extension file
      - Unpacking .ipsw is required

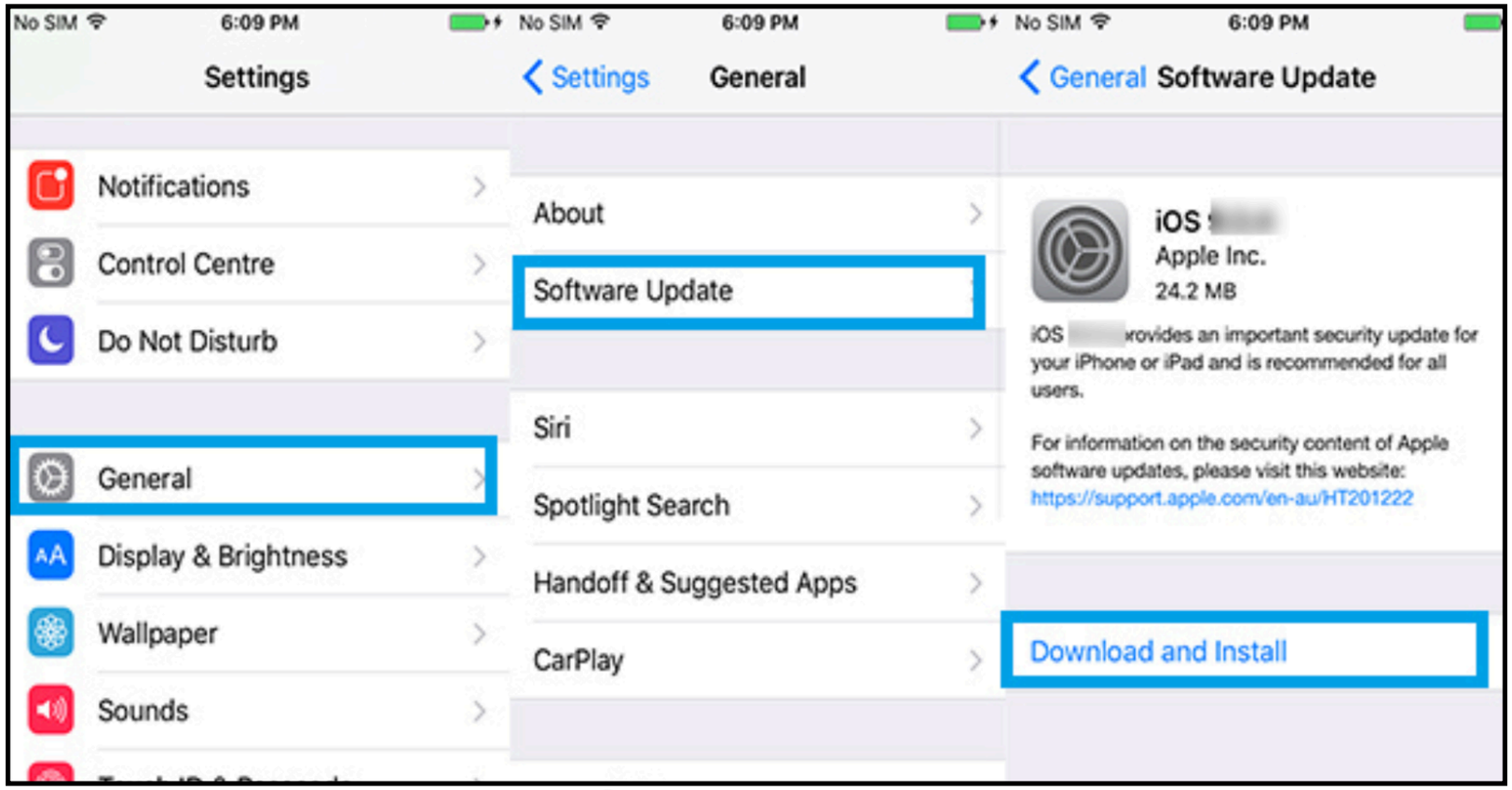


Figure. OTA update of iPhone

How to Find Bug

Reverse engineering - Obtaining firmware

- HTC Dream G1 (Android)
  - By dumping memory/flash using JTAG
    - Baseband image exist in the firmware It contains ELF and loader
  - JTAG can be used to dynamic debugging

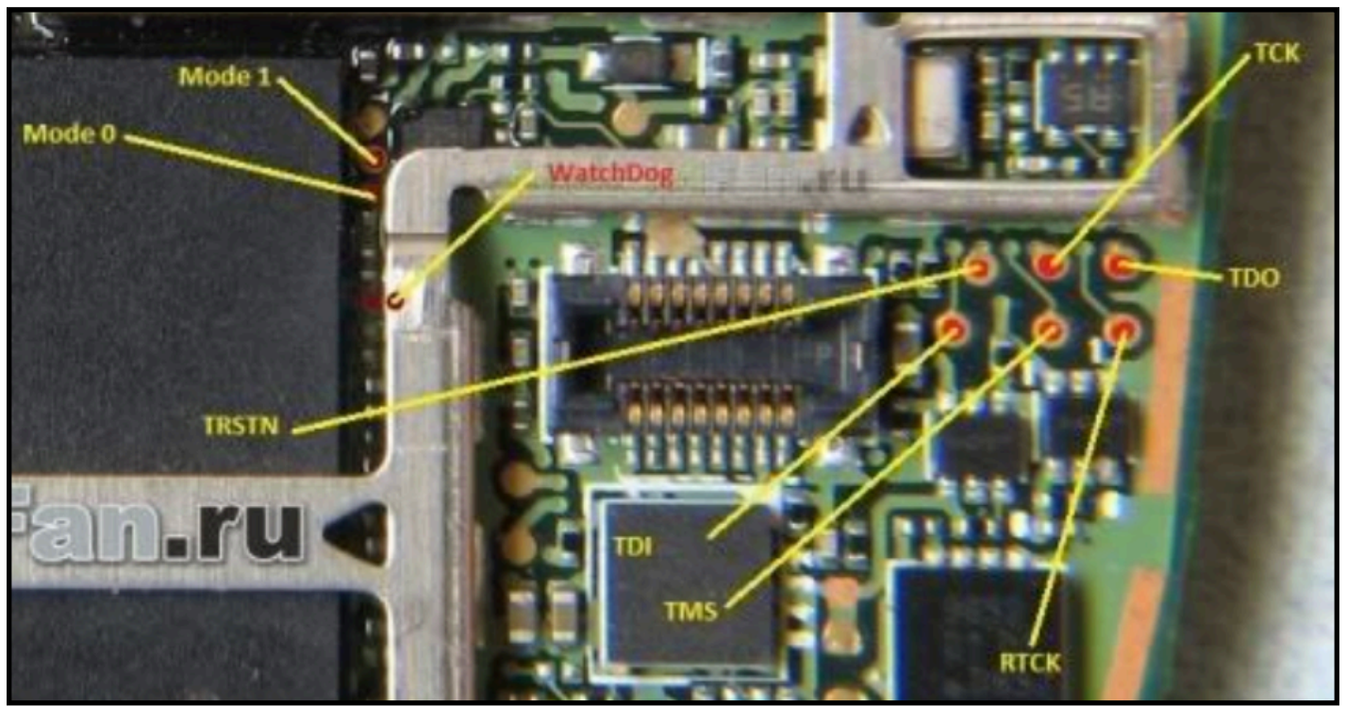


Figure. HTC Dream G1 JTAG pins on mainboard

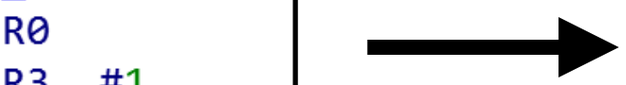
How to Find Bug

Reverse engineering - Analyzing binaries

- ARM binaries are supported by IDA Pro
  - Hex-Rays
    - Decompiler plugin of IDA Pro

```
00010FB4                STMFD   SP!, {R11,LR}
00010FB8                ADD     R11, SP, #4
00010FBC                BL      init
00010FC0
00010FC0 loc_10FC0
00010FC0                BL      menu
00010FC4                BL      read_int
00010FC8                MOV     R3, R0
00010FCC                SUB     R3, R3, #1
00010FD0                CMP     R3, #4
00010FD4                LDRLS   PC, [PC,R3,LSL#2
00010FD8                B       loc_11018
00010FD8 ; ---------------------------------------
00010FDC                DCD loc_10FF0
00010FDC                DCD loc_10FF8
00010FDC                DCD loc_11000
```

decompiled by hex-rays



```
int v3; // r0
int v4; // r0

v3 = init(argc, argv, envp);
while ( 1 )
{
  v4 = menu(v3);
  switch ( read_int(v4) )
  {
    case 1:
      v3 = info();
      break;
    case 2:
      v3 = login();
      break;
```

How to Find Bug

Reverse engineering - Analyzing binaries

- Symbol identification
  - Zynamics's BinDiff, a binary diffing tool, can be used
  - Memory copy function symbols can be identified
    - memcpy(), memmov(), bcopy() and so on

Similarity

	nfid...	Address	Primary Name	Type	Address	Secondary Name	Type	Basic Blocks			Jumps		
0.76	0.78	00438C...	sub_438C9C	No...	00650518	sub_650518	No...	0	3	0	0	3	0
0.76	0.78	00438C...	sub_438CBC	No...	006DD4...	sub_6DD4B4	No...	0	3	0	0	3	0
0.76	0.78	004A7948	sub_4A7948	No...	004C9B...	sub_4C9B7C	No...	0	3	0	0	3	0
0.76	0.78	004AA638	sub_4AA638	No...	0064F3E8	sub_64F3E8	No...	0	3	0	0	3	0
0.76	0.78	004CAA54	sub_4CAA54	No...	005502B4	sub_5502B4	No...	0	3	0	0	3	0
0.76	0.78	004D1B...	sub_4D1B90	No...	006492E0	sub_6492E0	No...	0	3	0	0	3	0
0.76	0.78	004CA1...	sub_4CA1C0	No...	004D5B...	sub_4D5B4C	No...	0	3	0	0	3	0
0.77	0.78	004B40...	sub_4B40C4	No...	004CAB...	sub_4CAB38	No...	0	3	0	0	3	0
0.81	0.95	0060B6...	sub_60B65C	No...	0060B9...	sub_60B9CC	No...	0	6	2	2	6	5
0.85	0.85	0062B6...	sub_62B61C	No...	00551C...	sub_551CD8	No...	0	3	0	0	3	0
0.86	0.92	0062305C	sub_62305C	No...	004CA2A4	sub_4CA2A4	No...	0	3	0	0	3	0
0.87	0.92	00532534	sub_532534	No...	0065043C	sub_65043C	No...	0	4	0	0	5	0
0.88	0.92	004D1788	sub_4D1788	No...	004C7D...	sub_4C7D8C	No...	0	5	0	0	6	0
0.89	0.98	004DFF60	sub_4DFF60	No...	004E0044	sub_4E0044	No...	1	10	0	5	12	4
0.91	0.92	006C49...	sub_6C49B8	No...	006D60...	sub_6D60CC	No...	0	5	0	0	6	0

How to Find Bug

Reverse engineering - Analyzing binaries

- Analyzing iPhone 2G
  - iPhone 2G has no UMTS (3G) and GPS functions
  - The analyzed work can be ported to iPhone 4 through BinDiff

smaller than iPhone 4

too big!

How to Find Bug

Reverse engineering - Analyzing binaries

- Dynamic debugging
  - JTAG
    - obtaining machine code, setting breakpoint, obtaining register status, ...
  - In HTC Dream G1, second boot loader, which is OS boot loader, doesn't allow JTAG
  - But the the before getting into second boot loader, we can set breakpoint and can change the JTAG allowing flag

Part 4. Memory Corruptions Found

- Types of bug found
- Example in Intel Infineon baseband code (CVE-2010-3832)
- Example in Qualcomm baseband code
- Demo

Memory Corruptions Found

Types of bug found

- Insufficient length checks for memory copy
  - it can be found more easily by identifying symbols of memory copy functions
- Object lifecycle issue
  - GSM has complex state machine
    - allocation/freeing pair mismatching
      - use-after-free, uninitialized use, unhandled state
- Reaching code path not to be reached
  - code path for UMTS (3G) can be reached using GSM (2G)

Memory Corruptions Found

Example in Intel Infineon baseband code (CVE-2010-3832)

- Temporary Mobile Subscriber Identifier (TMSI)
  - It's supposed to be always 32 bits long value
  - but variable length field (1 byte) is used for TMSI
    - L in IE of layer 3 message
- No enough space to take TMSI (> 32 bits)
  - It trusts the variable length field and copies the TMSI sent by fake BTS
  - Heap buffer overflow occurs
- CVE-2010-3832
  - It allows attackers to execute arbitrary code remotely

Memory Corruptions Found

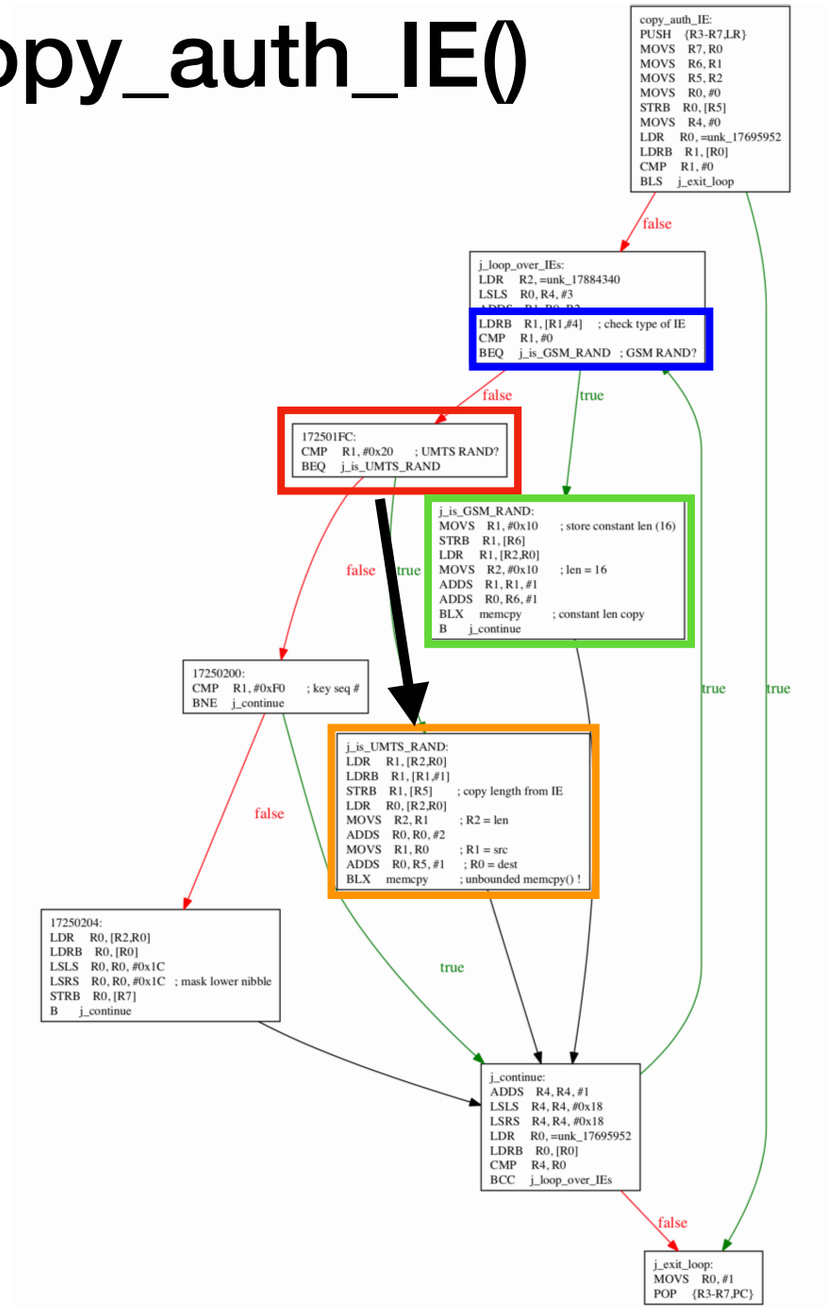
Example in Qualcomm baseband code

- During authentication, BTS send a challenge response
  - In GSM, RAND 16 bytes (which is constant)
  - In UMTS, AUTN 16 bytes (which has variable length field)
- Even if Qualcomm baseband in GSM mode accept AUTN
  - By changing RAND's IE type to AUTN
- Sending RAND (> 16 bytes) with AUTN IE type
  - Stack buffer overflow
    - Program counter can be overwritten
    - Saved registers can be overwritten
  - Remote code execution!

Memory Corruptions Found

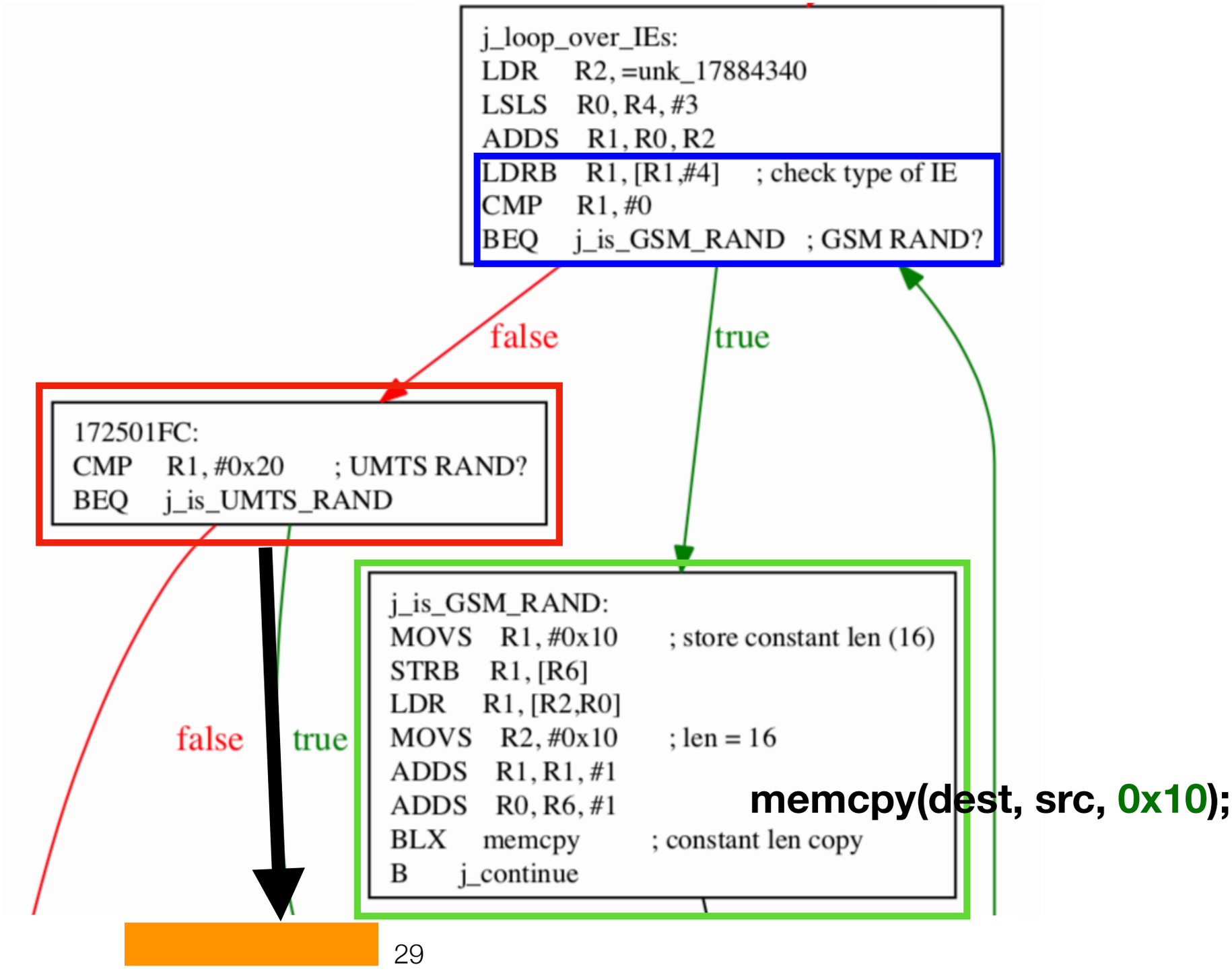
From bugs to exploitations - Qualcomm baseband code

control flow of copy_auth_IE()



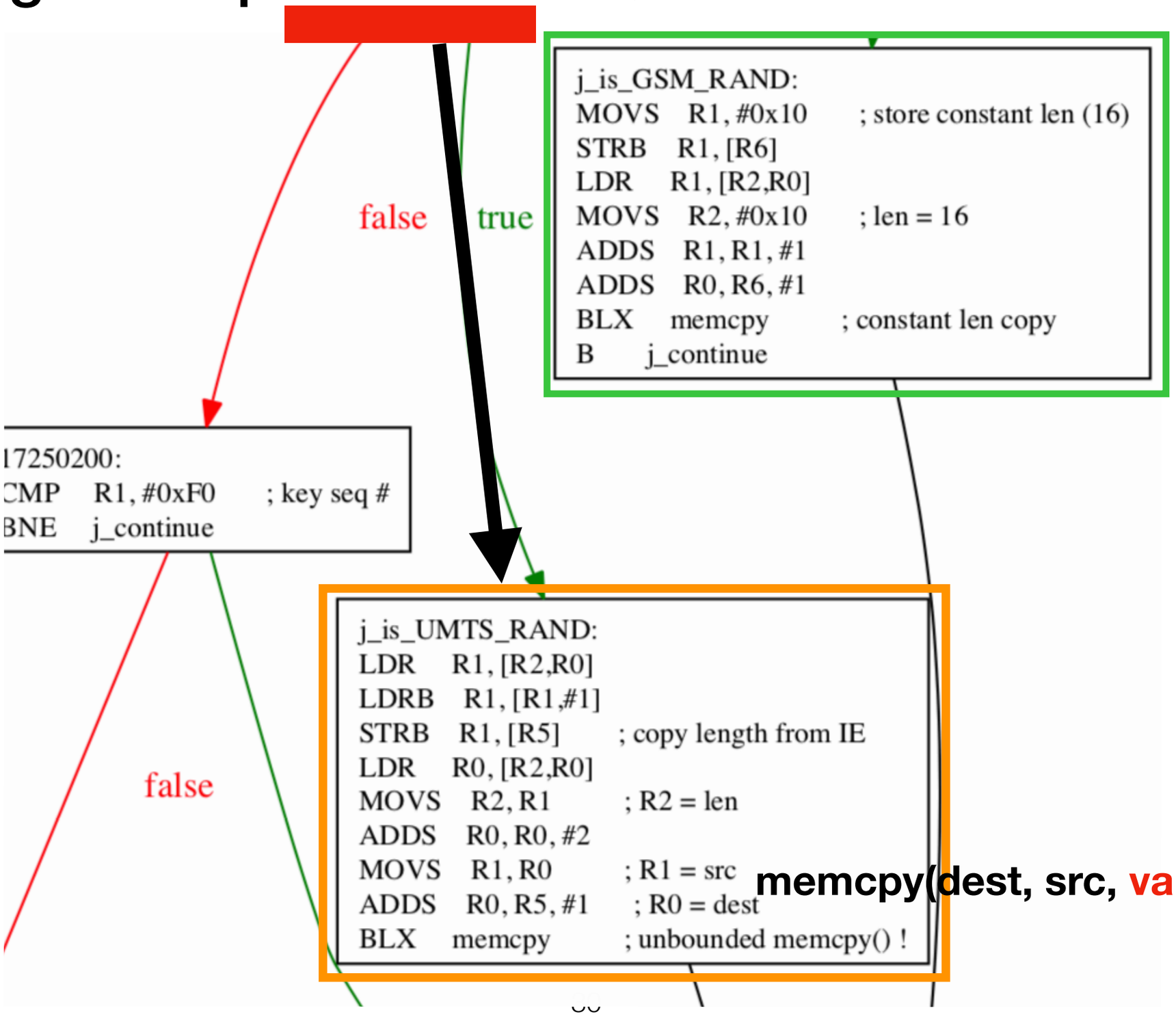
Memory Corruptions Found

From bugs to exploitations - Qualcomm baseband code



Memory Corruptions Found

From bugs to exploitations - Qualcomm baseband code



Memory Corruptions Found

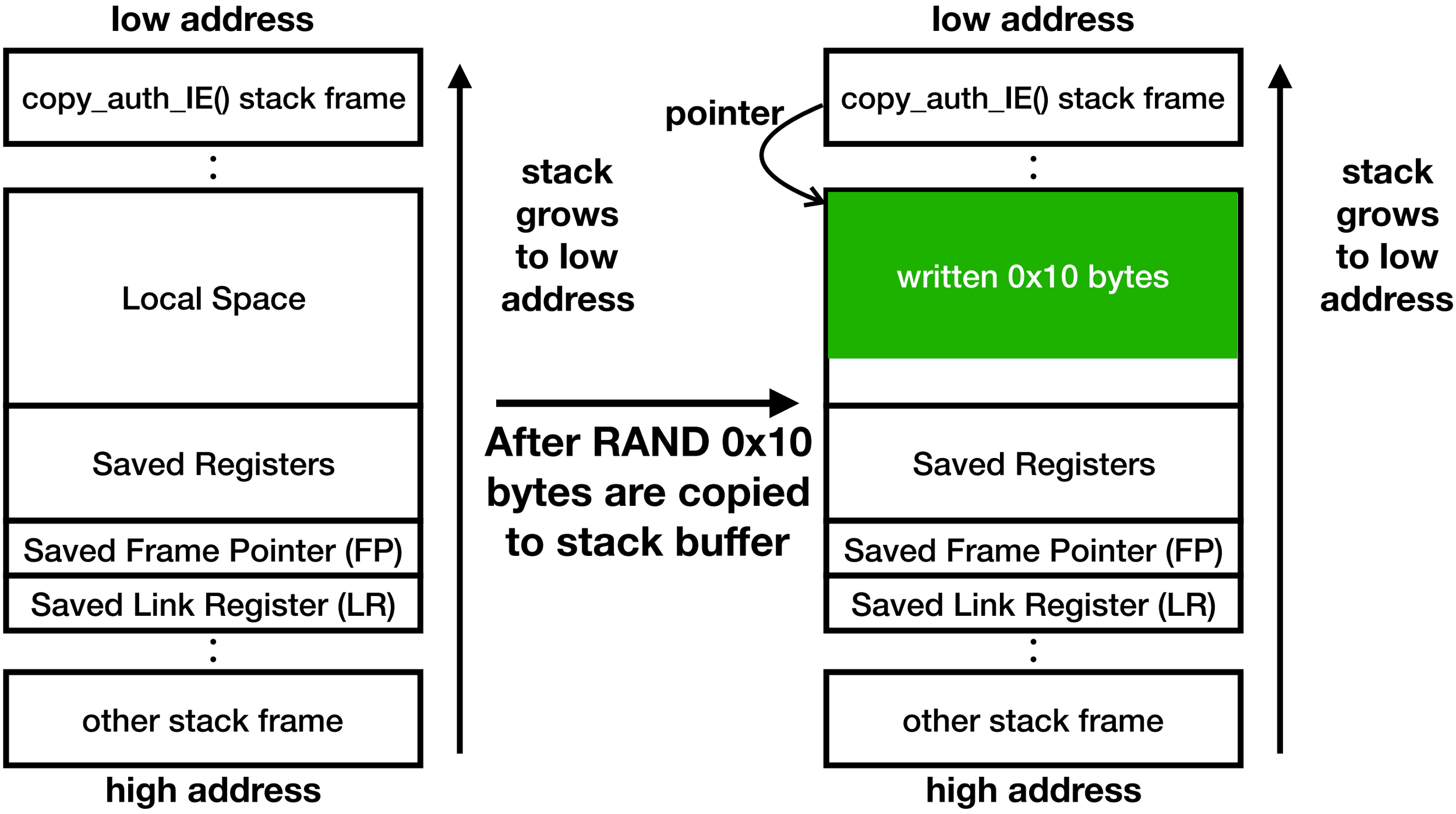
From bugs to exploitations - Qualcomm baseband code

- FakeBTS
  - Ettus Research USRPv1
    - It provides RF processing capability
  - Laptop with OpenBTS
    - Software-defined GSM access point

- Payload
  - Changing return address --> ATS0=n handler
  - Changing saved R0 register value --> 1 (ON)
  --> ATS0(0); is executed
  --> Auto-answer feature is turned on
  --> control flow hijacking can be proved

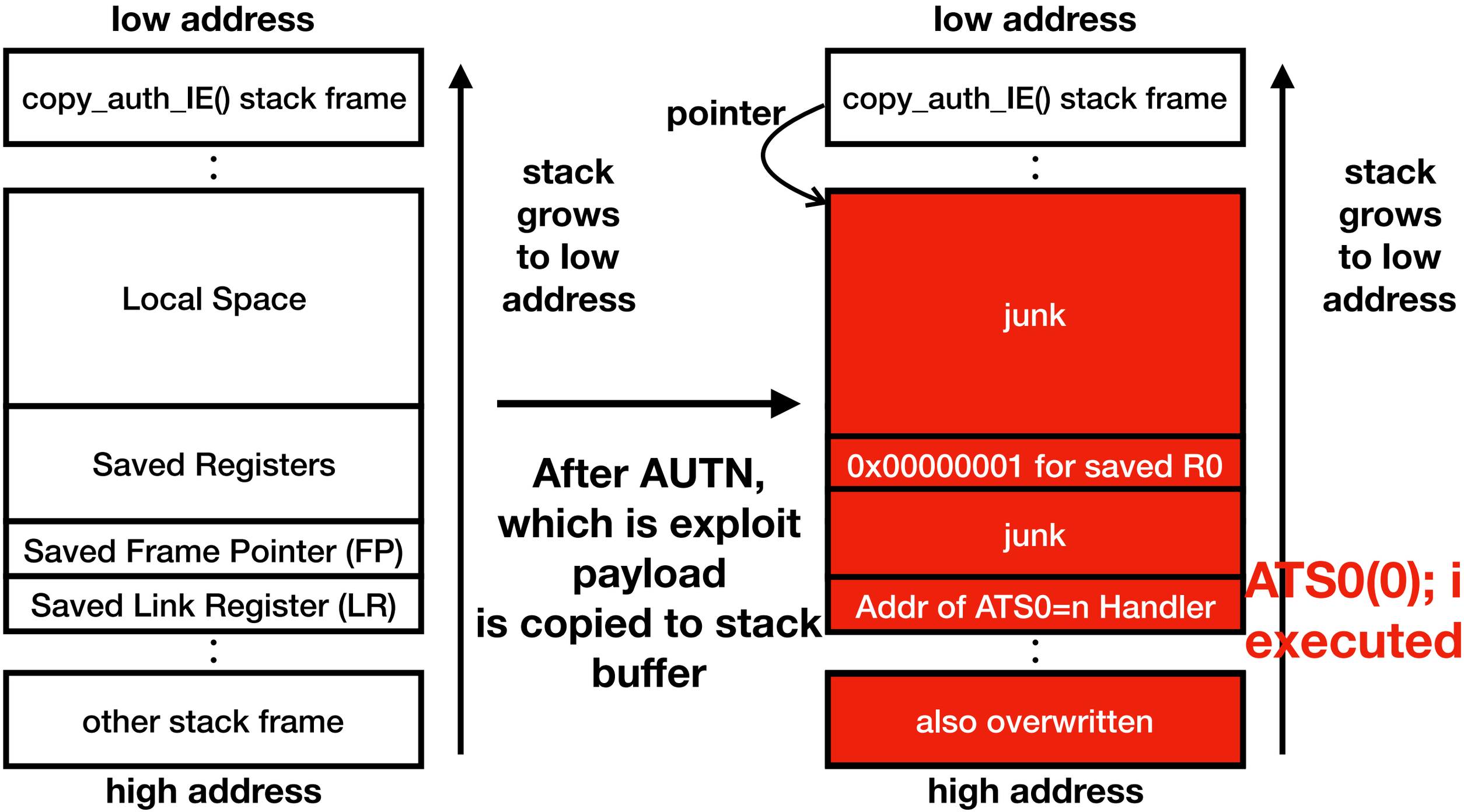
Memory Corruptions Found

From bugs to exploitations - Qualcomm baseband code



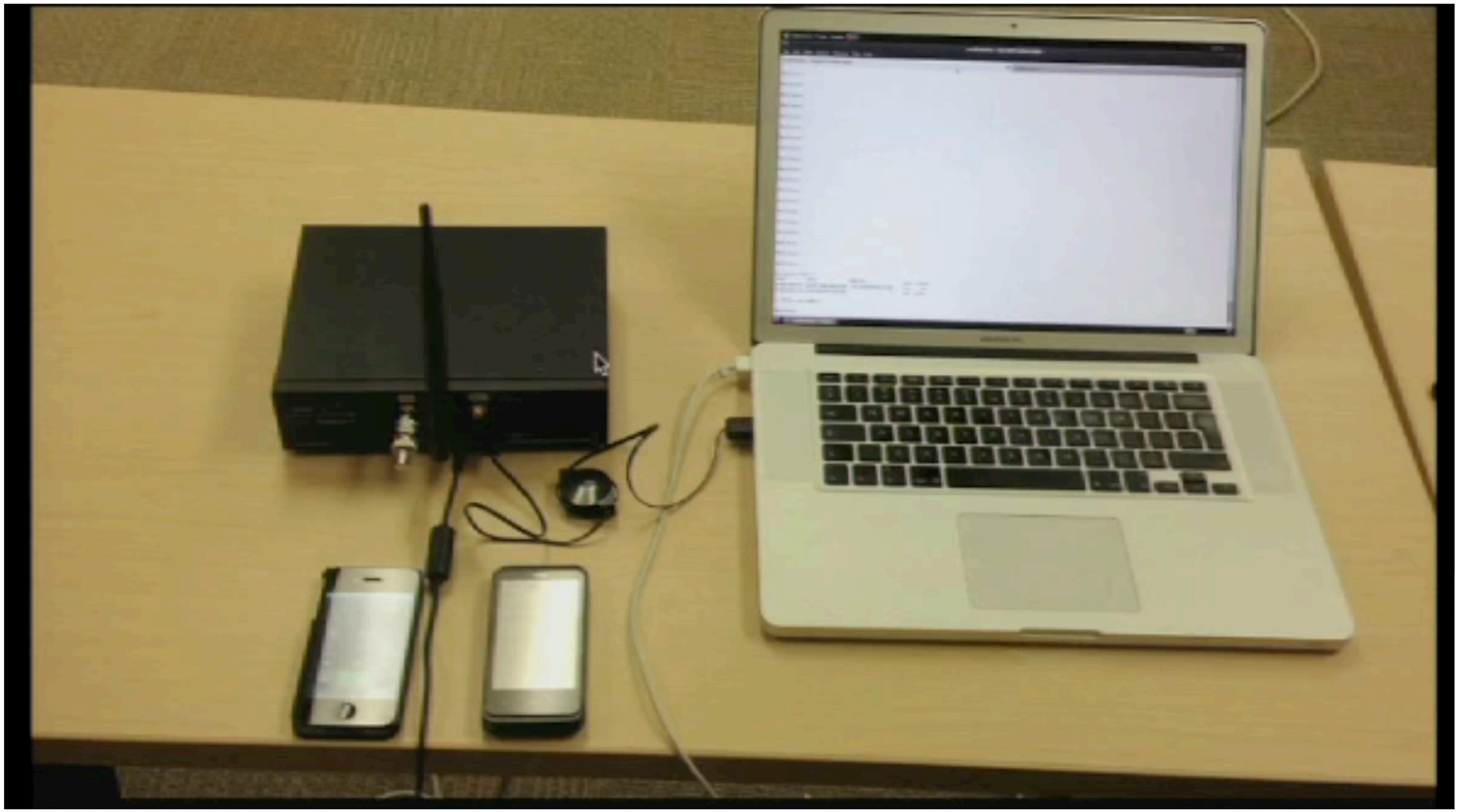
Memory Corruptions Found

From bugs to exploitations - Qualcomm baseband code



Memory Corruptions Found

From bugs to exploitations - Qualcomm baseband code



Part 5. Impact & Conclusion

- Impact
- Defense

Impact & Conclusion

Impact

- Billing issue
  - By controlling compromised baseband, adversary can send MMS or cause large data transfer
- Feasibility of eavesdropping
  - Audio routing is done by baseband stack
- Bricking phone
  - adversary can write something to NVRAM region which contain important data like IMEI
- In case of shared memory design in which single RAM is used for both application and baseband stack
- Replaying this attack somewhere crowded areas can gives critical damage

Impact & Conclusion

Conclusion

- Attack can be performed with reasonable budget
  -Laptop (with OpenBTS), USRP
- iPhone 4 (iOS 4.2)
  - TMSI overflow was assigned to CVE-2010-3832
- HTC Dream G1
  - No public documentation
  - But, length check is added for parsing AUTN
- 3G also is expected to be vulnerable
  - Malicious Femtocell
  - 1500 pages for layer 3 of 3G protocol specification

Impact & Conclusion

Conclusion - Solutions

- Strict software security assessment
  - Vendors should find and patch the bugs by code auditing and testing before attackers
- Mitigation techniques should be enabled
  - Stack canary, heap protections, DEP, ASLR, ...
- Mutual authentication between MS and BTS
  - But, SW/HW manufacturers agreement is required to patch their products to add more authentication phase

Part 6. Related works & Future works

- Related works
- Future works

Related works & Future works

Related works

- C. Mulliner, N. Golde, J. pierre Seifert, **"SMS of Death: From Analyzing to Attacking Mobile Phones on a Large Scale"**, USENIX, 2011.
- F. van den Broek, B. Hond, A. Cedillo Torres, **"Security Testing of GSM Implementations**", ESSoS, 2014.
- N. Golde, D. Komaromy, **"Breaking Band: Reverse Engineering and Exploiting The Shannon Base Band"**, Recon, 2016.

Related works & Future works

Future works

- Attack implementation for recent cellular phone
  - Recently, AP and CP have its own RAM respectively
  - Even in such hardened design
    - Is escalation to application from baseband possible?
      - With assumption baseband already is comprised
      - Is there any attack vector from baseband to application?

Thank you