

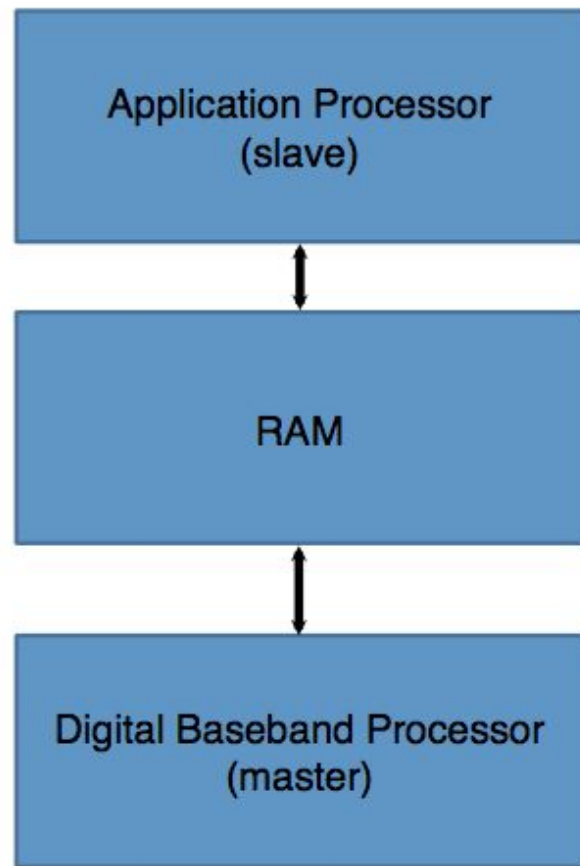
# the baseband bane

Neerad Somanchi

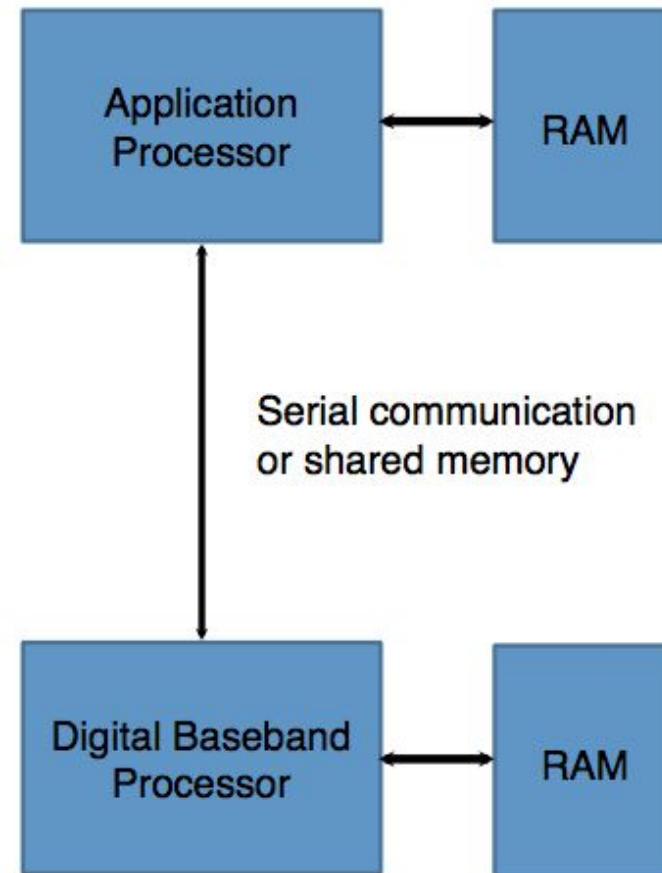
Anjana Guruprasad

# A Typical Smartphone

- How many processors?
  - The application processor (AP) – The one advertisements talk about
  - Qualcomm Snapdragon, Samsung Exynos, Apple A series
  - The communications processor (CP) – The Internet!
  - Low power ARM CPUs – Protocol Stack Processor
  - Qualcomm X12, Samsung Shannon S333
- How many Operating Systems?
  - AP – Android, iOS
  - CP – RTOS to handle time critical operations
    - Proprietary, closed source
    - Nucleus OS, ThreadX, Shannon OS



***Shared memory architecture***

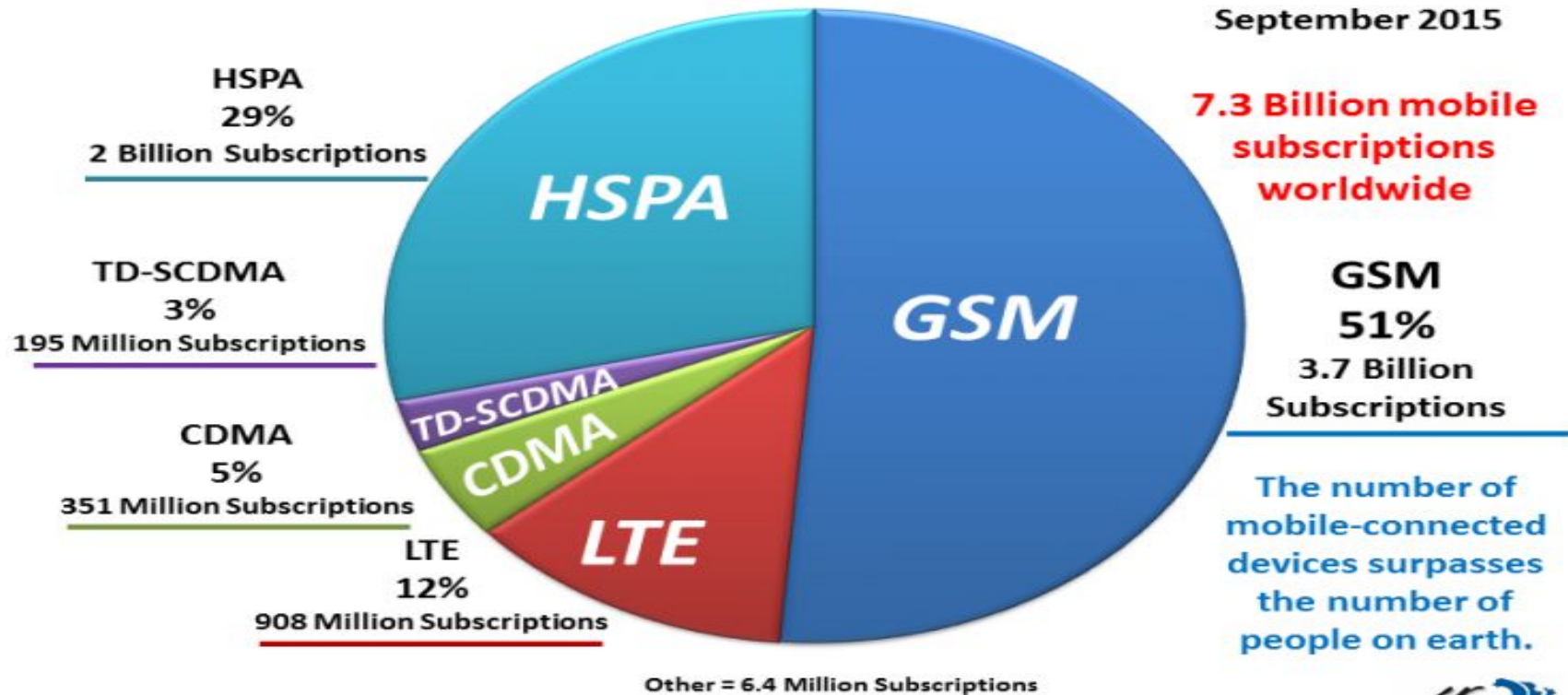



***Baseband as modem***

# Radio access technology

## Global Mobile Subscribers and Market Share by Technology

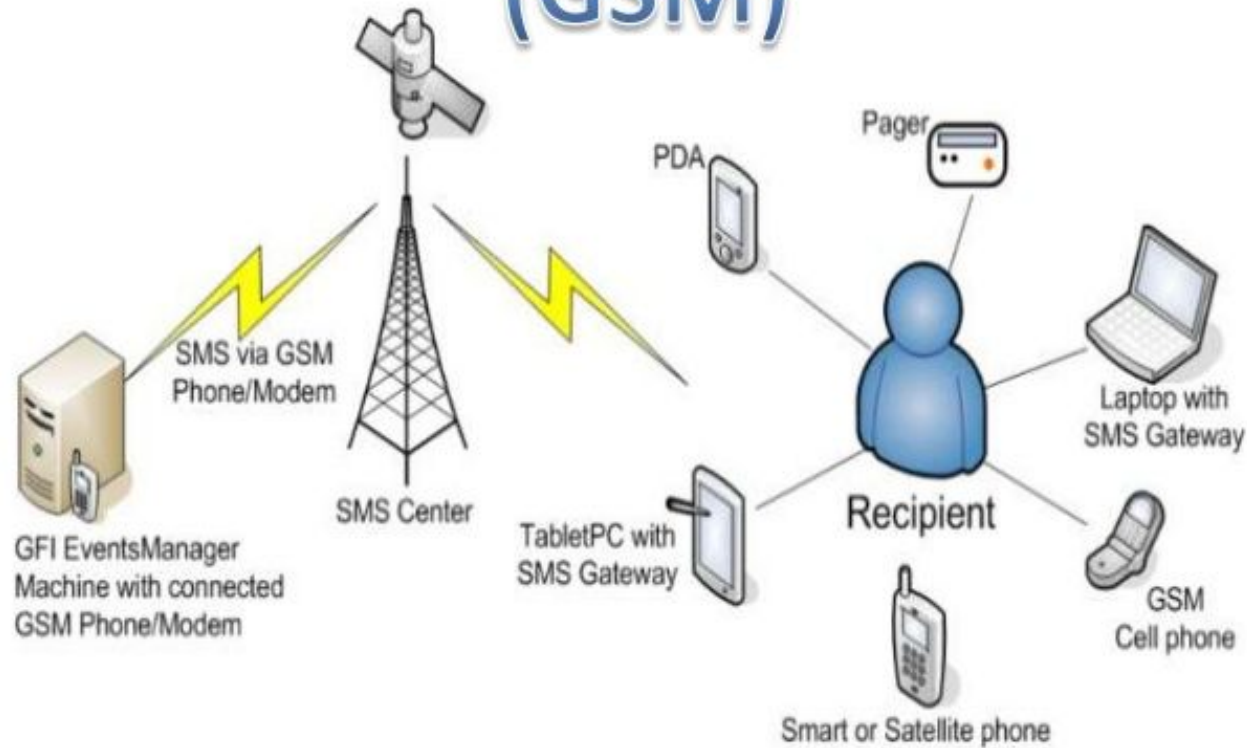
September 2015



Source:  September 2015

  
www.4gamericas.org

# Global System for Mobile (GSM)



# GSM PROTOCOL STACK



# Baseband Protocol stack

- Code base created in the 1990s... With a 1990s attitude towards security
- Network elements like Base Transceiver Station(BTS) are considered trusted
  - Very expensive back then
  - Now - Rogue BTS can be fashioned for as little as 1500 USD
- Layer 3 Protocol – Specified in GSM 04.08
  - Allows for variable length messages
  - Maximum Length: 255 Bytes (Length Field: One Byte)
- Some messages specified to be encoded as variable length messages
- ...even though the message description implies that it is of fixed length
- Potential Exploit!

# Finding Bugs

- Fuzzing – Providing invalid, unexpected and random data as protocol messages
  - Baseband crashes, but no way to glean any information from crash logs
- Static Analysis – Analyze code without executing it
  - No source code publicly available
  - Exception – Vitelcom TSM30 source code was leaked in 2004
  - Helped understand the general architecture of the GSM protocol stack code
- Conclusion – Reverse engineer binaries
- OTA firmware updates contain baseband firmware as well



# Reverse engineering binaries

- Tools for identifying interesting code paths – IDA Pro Disassembler and Google BinDiff
- Disassembler translates machine language into assembly language – inverse of assembler
- BinDiff compares and identifies identical and similar functions in disassembled code
- BinDiff generates function “fingerprints”
- Run both tools on target binary and a known code base – VSM30 to the rescue!
- Helped identify functions like memcpy() and memmov()
- Then identify functions that used variable-length memory copies
- Check if they employed sufficient length checking for the copied or moved data

# The bugs!

- Insufficient length checks, aka, unchecked memory copies
  - Found in binary once memcpy() et al. are identified
- Object/structure lifecycle issues
  - Generous use of state machines in GSM
  - Use-after-free bugs, uninitialized variables, unhandled states
  - Harder to exploit these bugs
- Code path pains
  - Code paths used for 3G (UMTS) can be triggered using GSM messages

# Example (Infineon Code base)

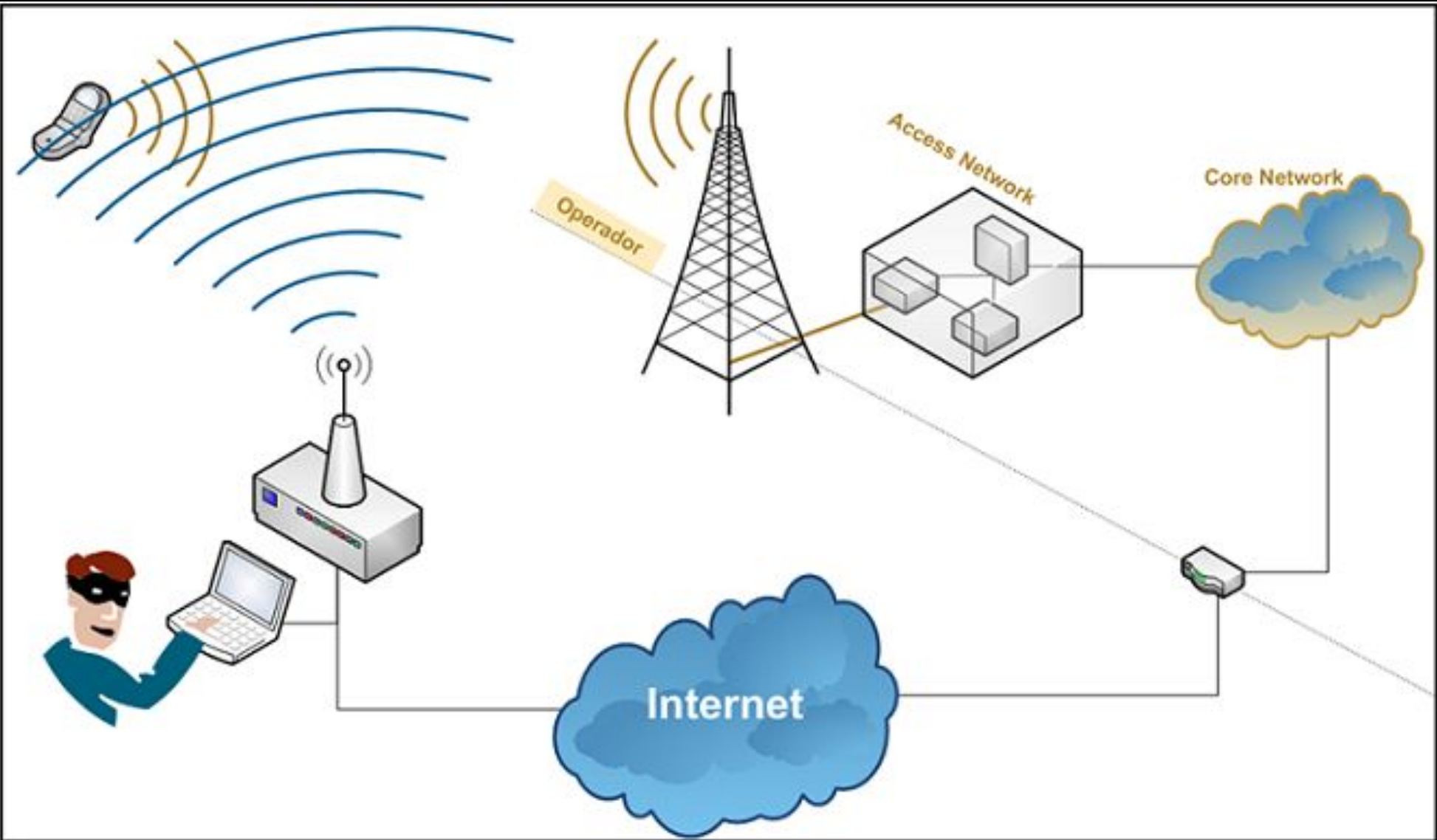
- TMSI – Temporary Mobile Subscriber Identity
  - Always a 32 bit value
  - For some reason, encoded with a length field
- Engineer A allocates only 32 bits for the TSMI value
- Engineer B trusts the length field and copies the value sent by rogue BTS into location above
- Results in a heap overflow
- Tricky to exploit in a stable way – leads to a modem crash
- Issue identified in iPhone 4 – Fixed in the subsequent point update to the OS

# Example (Qualcomm code base)

- For authentication in GSM, BTS transmits a 16 byte challenge value called RAND
- 3G (UMTS) uses a variable length message called AUTN, but is specified to also be only 16 bytes long
- Qualcomm stack accepts the AUTN challenge even when operating in GSM mode
- Apparently a workaround used by Qualcomm for compatibility reasons
- Rogue BTS sends AUTN message of length > 16 bytes
- Stack overflow as only 16 bytes are provisioned for RAND challenges
- Result – Remote code execution (before successful authentication)
- Qualcomm fixed it after disclosure

# 'AT + s0 = n' feature exploited

- Hayes AT command set – a specific command language developed for modems in 1981
  - Short text strings combined to produce commands for operations like dialing, hanging up and changing connection parameters
- AT + S0 is a Hayes command to turn on auto-answer
- Code exists in Infineon and Qualcomm stacks to enable this feature
- For instance, \*5005\*AANS# enables auto answer on the iPhone 4
- First, locate the AT command handler function for setting S0 register
- Requires examining memory and register contents at runtime
- Enter, JTAG
  - **Joint Test Action Group** – developed standards for on-chip instrumentation
  - **Processors use JTAG-specified port to provide debugging information**
  - **Software Patch allows JTAG access in HTC dream (Qualcomm baseband)**



# Target – HTC Dream (Qualcomm)

- Rogue BTS - Ettus Research USRPv1, provides RF processing capability
  - Supports two daughter RF boards, for transmit and receive
  - OpenBTS, running on a laptop, modified with patches to perform the exploit
- Phone tries to authenticate with the rogue BTS
- Use the AUTN exploit previously discussed, which causes a stack buffer overflow
- Overwrite the program counter and register r0 of the stack frame
- PC with the entry point of s0 register handler, r0 with value 1
- Overwrite the subsequent stack frame's PC as well to ensure smooth execution (no crash)
- With the rogue AUTN message, less than 100 bytes long, this exploit is possible
- Auto answer is enabled without the user being aware

# Impact

- Place Rogue BTS in crowded/sensitive areas
- Audio routing on most chipsets is done on baseband CPU
- Which means it has access to the built-in mic
- Baseband processors have large quantities of RAM available
- Record audio, store in ram, piggy back onto the next data connection
- Shared memory architecture – AP can also be compromised
  - Higher layer security features are bypassed
- Brick phones permanently



# Solutions?

- Open source baseband stack
  - Quicker at identifying bugs
  - But still hard to patch them as phones need to be carrier certified – long process
- Isolation
  - Cut off baseband access to the mic when not on a call
- Stringent quality control by CP manufacturers
  - Use tools like coverity to check for possible buffer overflows
- Problem is worse with 3G
  - Radio Resource Control protocol specifications almost 1800 pages long
  - Messy, complicated
  - LTE is cleaner

# References

- Baseband Attacks: Remote Exploitation of Memory Corruptions in Cellular Protocol Stacks - Ralf-Philipp Weinmann, *University of Luxembourg*