Reverse engineering a Qualcomm baseband

Guillaume Delugré
Sogeti / ESEC R&D
guillaume(at)security-labs.org

28C3 - Berlin

The baseband world

What is a baseband processor?

- The main chipset of your phone
- Responsible for handling telecommunications
- Directly interfaced to hardware (microphone, speakers, ...)
- Includes stacks for telephony protocols
- Smartphones also include an application processor running a separate OS (e.g. Android, iOS, ...)

Largest suppliers

- Qualcomm (HTC phones, iPhone 4S)
- MediaTek
- Infineon (iPhone)

The baseband world

Getting into the baseband world

- Hacking phones is something hard to reach
- Quite a closed industry
- Network side: reading the 3GPP specs is a life achievement
- System side: Aside from OsmocomBB, everything is closed

Yet good reasons to look into basebands

- Understanding how a phone really works
- Very big and old code base, good potential for vulnerabilities
- Unlocking

The baseband world

Exploitation on baseband

- Probably not a lot of exploit mitigation techniques
- Ralf-Philipp Weinmann reported vulnerabilities in Infineon and Qualcomm basebands
- But exploitation is almost impossible if you don't know the environment in which you are running
- Clear lack of litterature

About this talk

- Describe the RTOS used on a Qualcomm baseband
- Implementation of a debugger

Plan

Targeted device

Targeted device

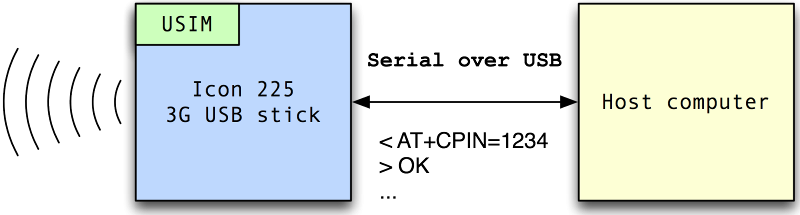
Icon 225

- USB 3G stick
- Qualcomm baseband inside, model MSM6280
- Processor ARM926EJ-S (ARMv5)
- Two proprietary Qualcomm DSPs (audio & modem)

Evolution

- Model dating back to 2008
- REX kernel running as OKL4 guest in newer generations

3G USB stick normal operation mode



Hayes commands set as defined by the 3GPP TS 27.007 specification

- The stick emulates a serial line over USB
- Communication with the host through standard AT commands
- Registers to the cellular network, carries packet data over 3G

First contact

USB stick entry points

- Plugging the stick creates 3 emulated serial ports
    1. AT commands / packet data (multiplexing mode)
    2. Packet data (no multiplexing)
    3. Channel to a Qualcomm diagnostic task

Enabling the diagnostic channel

- Directly accessible on the Icon 225 stick
- Might need to send the AT command `AT$QCDMG` on some models

DIAG task protocol

0x7e	CMD	*Variable-length data*	CRC-CCITT	**0x7e**

Diagnostic protocol

- Undocumented but simple protocol
- Partly reversed in ModemManager (libqcdm)
- Begin-end markers (`0x7e`)
- One byte for command type
- Variable parameters (with escaped `0x7e` and `0x7d` bytes)
- 16-bits CRC-CCITT

DIAG commands

Reading and writing to memory

- The diagnostic task seems to support a lot of commands
- Some of them offering direct access to memory
- Command `0x02` reads a byte in memory
- Command `0x05` writes a byte in memory

Dumping the system memory

- Dump the primary bootloader at `0xffff0000`
- Dump the whole system memory from 0

Downloader mode

Downloader mode

- Presence of a subset mode with only two commands available
  - Write data to memory
  - Execute at address
- Access limited to a small hardcoded memory range
- Enabled:
  - through command 58 of the diagnostic mode
  - when the system crashes
  - on some HTC phones, `VolUp+VolDown+Power` during boot (5 vibrations)

Live memory snapshot

Live memory snapshot

- PBL does not check QC-SBL signature (not the case anymore)
- Memory snapshot is 32MB long, entry point at 0x80000
- The memory snapshot can be directly analyzed in IDA Pro

System characteristics

- MMU maps direct access to physical pages, first 13MB as read-only
- All tasks share the same address space
- Everything is running in ARM supervisor mode
- Presumably compiled with official ARM toolchain, no SSP
- ARMv5 does not support XN bit

Plan

1. Analysis of a 3G USB stick
2. REX, the Qualcomm real-time kernel
3. Live debugging on the baseband

REX

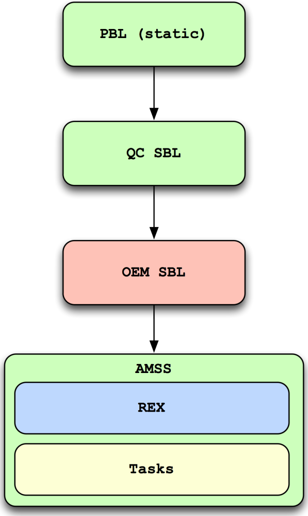
The Qualcomm RTOS

- Qualcomm has developed their own proprietary RT kernel, called REX
- Operating system is named AMSS
- The system is made of 69 concurrent tasks
- Tasks for
  - Hardware management (USB, USIM, DSPs, GPS...)
  - Protocol stacks at each layer (GSM L1, L2, RR, MM...)

Necessary reverse engineering

- **The kernel API**
- The C library
- Softfloat/arithmetic builtins

System boot



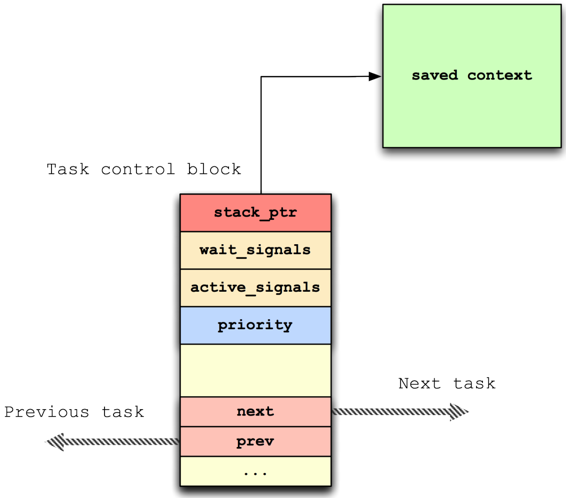
REX tasks

REX core tasks

- `SLEEP`: Idle task.
- `DPC`: Routes APCs across tasks.
- `MAIN`: Launches all system tasks, then handles timer events.
- `DOG`: Watchdog. Constantly checks that tasks are alive.
- `DS`: Data Services task. Unified data gathering task for all protocol layers.
- `CM`: Call Manager task.
- `PS`: Packet-switched Services. Network stacks at upper layers (TCP/IP, PPP...)
- `DIAG`: Provides the diagnostic interface.

Plan

Main fields of the task structure



REX scheduler

REX task

- All tasks share the same address space, ARM supervisor mode
- Double-chained task list, ordered by priority
- Context switch
  - Task context saved on the stack
  - Context switch $\Rightarrow$ stack switch + restore context

Synchronization

- Tasks can wait for a signal (up to 32 signals)
- Scheduler support for critical sections

REX APC/DPC mechanism

Asynchronous procedure calls

- Push a new context on the stack of a task
- When the task is scheduled, the call is executed
- Original context is then restored

Deferred procedure calls

- The task `DPC Task` is dedicated to dispatch APCs
- High priority task
- Allows a task to execute code in the context of another task

REX Timers

Timers

- Tasks can set a specific action to occur at regular interval
- Timers are handled by the `Main task`
- Timer lists ordered by deadline

Timer events

- Timer actions (non-exclusive)
  - Send a signal to a task
  - Execute an APC
  - Execute a direct routine call (main task context)

Plan

REX IPC

Inter-task communication

- Tasks primarily communicate through the mean of signals
- A task is put into a wait state until the right signal is fired

Common inter-task communication

1. Task `A` is waiting for data to process
2. Task `B` pushes data into a shared FIFO queue
3. Task `B` sends a signal to task `A`
4. Task `A` is scheduled, pop and process the data
5. Optionally, task `B` sends a signal to task `A` to acknowledge

Data pipes

Pipes

- The `DS` (Data Services) task implements data pipes
- Tasks can push and fetch data as a contiguous stream of memory
- Widely used
  - DS task seems to gather and route data from many layers into pipes
  - Implementation of sockets

Plan

REX heap

Heap

- Heap structure very simple
- Chunk metadata: `size`, `free?`
- Tasks generally use their own separate heap
- Presence of a global system heap, almost unused

Plan

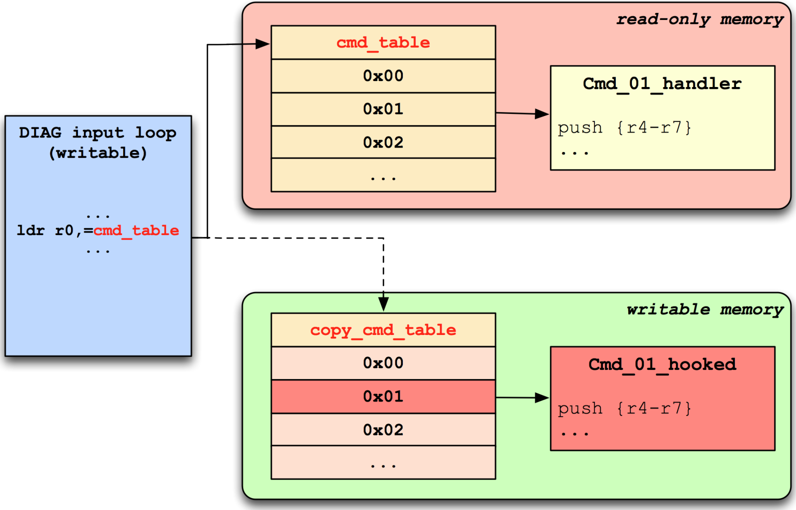
Plan

Code execution

Getting code execution

- Since arbitrary reads/writes are possible, code execution is easily achievable
- We want to be able to communicate with our injected code
  - Either hook an AT command
  - Or hook a DIAG task command
- Preferably hook a DIAG command and execute code in the context of the DIAG task
- This way we can still debug AT command handlers
- Communication with the payload over USB using the DIAG protocol (reuse of the DIAG task API)

Hooking the DIAG task



Plan

Debugger breaks

Stopping a task

- A task should be stopped
  - when the debugger instructs it to do so
  - after hitting a breakpoint / exception
- The task has to be unscheduled and resumed on-demand
- Use the kernel function `rex_wait` to pause the task
- The signal must be used *only* by the debugger
- The task is resumed by setting the debug signal (`rex_set_task_signals`)
- The debugger can force a task to stop by sending a DPC

Plan

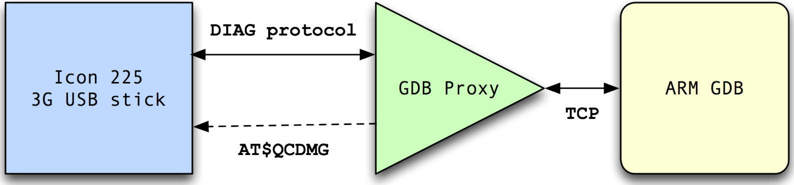
Single-step on ARM

Single step

- ARM has no native support for single-step
- Multiple implementations possible for single-stepping
- **Standard**: compute next $pc value, set a BP and continue
  - ⇒ Not multithread safe.
- **Emulated**: emulate the instruction in the thread context.
  - ⇒ Needs a full ARM/Thumb emulator...
- **Displaced**: copy the instruction into a separate buffer, set a BP after it and jump on the buffer
  - ⇒ Needs JIT assembler to perform instruction relocation.
- **Virtualized**: Implements separate virtual address space for each task and uses standard method
  - ⇒ Needs to patch the scheduler

Plan

Debugging architecture



Debugger architecture

- Proxy to bridge GDB requests and the DIAG protocol
- GDB used in non-stop mode (multithread support)
- REX tasks are shown as threads to GDB

Demo

Demo

Conclusion

Conclusion

- Baseband systems massively used but still poorly known
- Icon 225 key uses a non-secure bootloader
- Those chips are also nice for code execution and analysis
- More recent versions uses a REX/OKL4 hybrid kernel
- Interactions with application processor on real phones

Baseband related works

Related talks

- R.-P. Weinmann, *The Baseband Apocalypse*, 27C3
- Luis Miras, *Baseband playground*, Ekoparty 7th edition

On the web

- `tjworld.net`
- `bb.osmocom.org`

Thank you for your attention!

Questions?