Comsecuris

Baseband exploitation in 2013: Hexagon challenges

Ralf-Philipp Weinmann
<ralf@comsecuris.com>

Chaos Communication Congress
2013-12-27, Hamburg, Germany

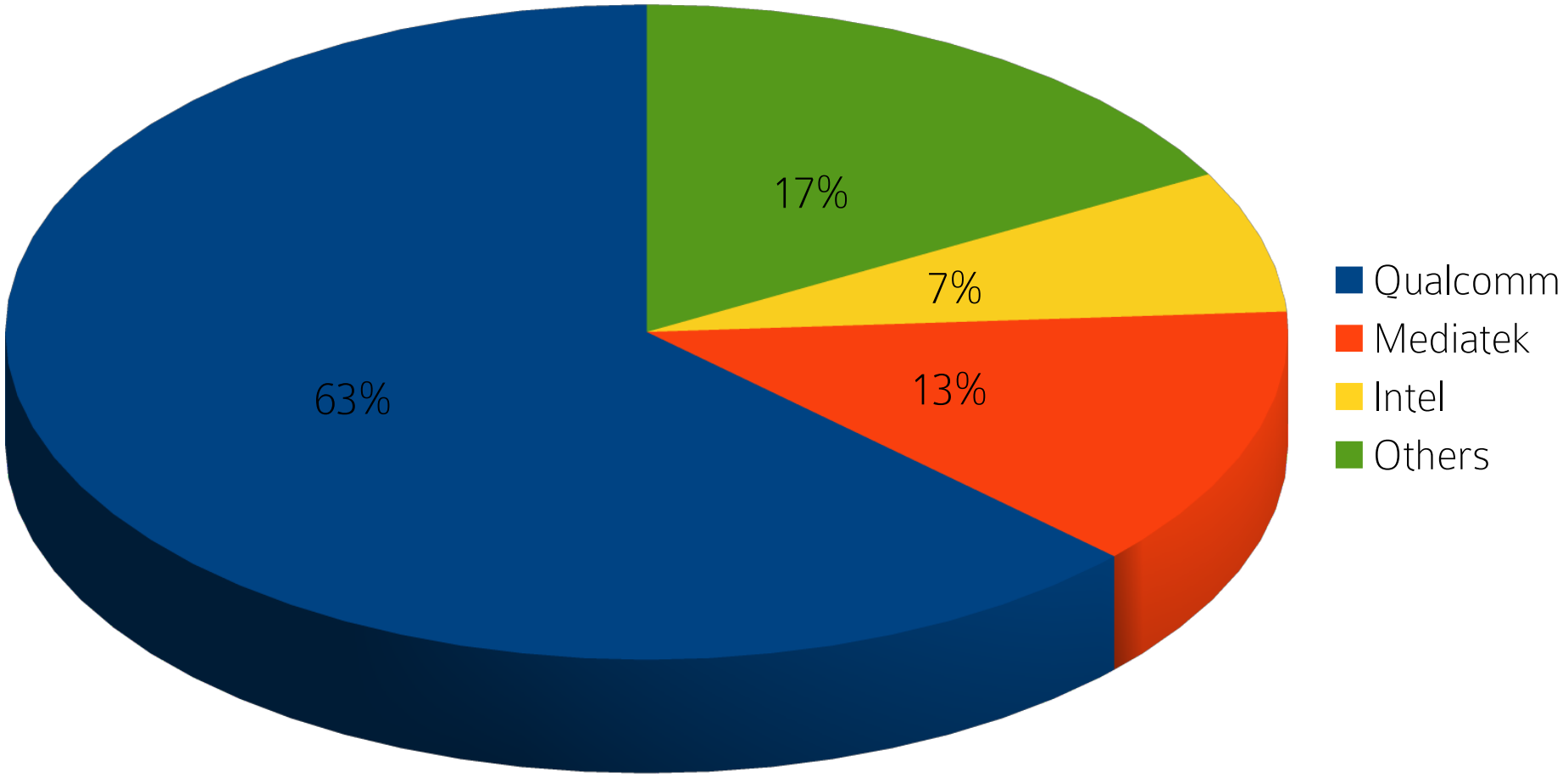
Who am I?

- Security researcher from Germany
- Previously in academia (University of Luxembourg)
- Now working for my own company
- Keen interest in security of mobile, wireless and embedded systems
- First to demonstrate remotely exploitable vulnerabilities in baseband stacks (3 years ago, e.g. at 27c3)

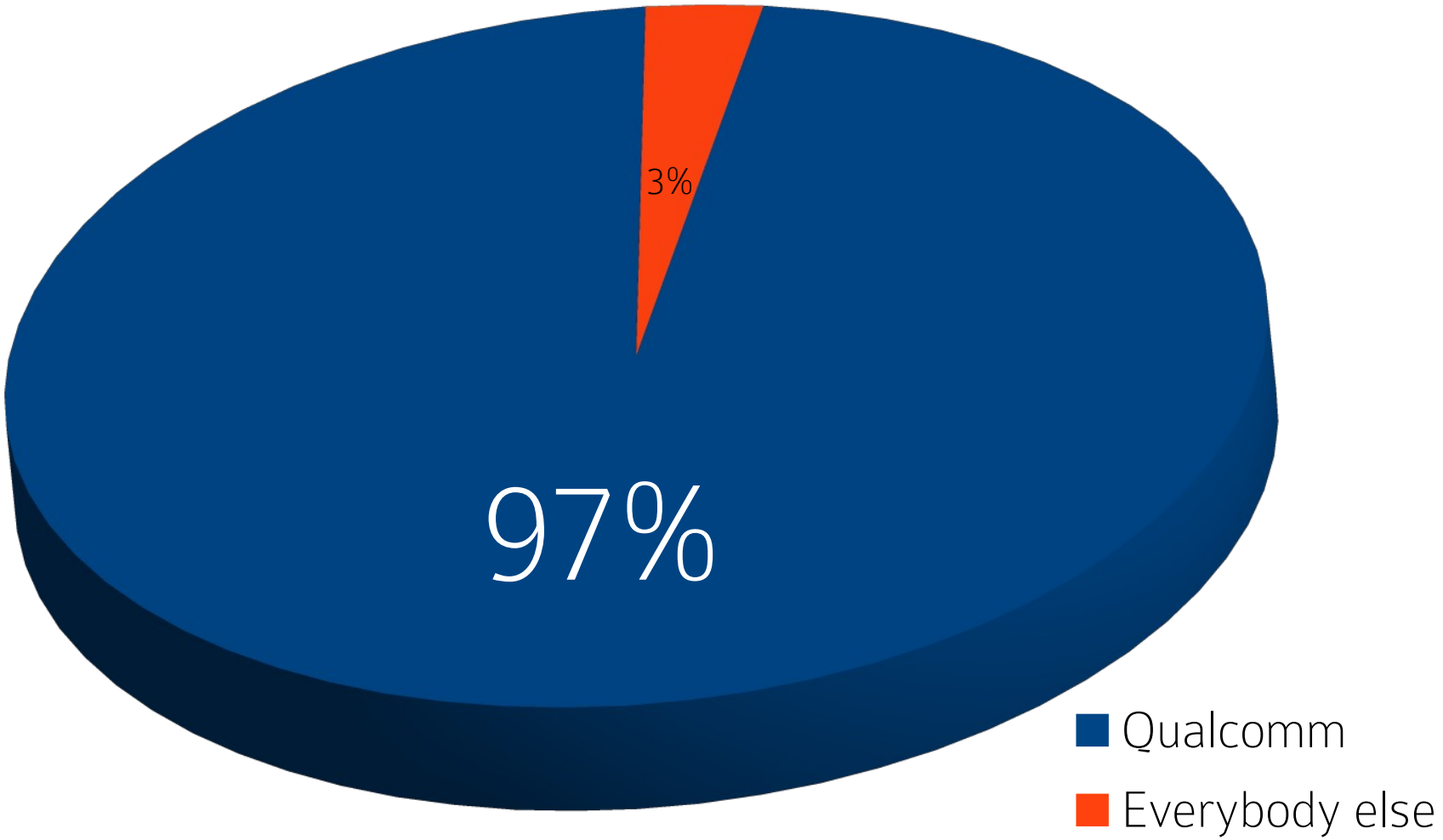
Overview

- Importance of Hexagon for mobile exploitation
- Intro to the QDSP6 architecture
- Past issues with BLAST
- On the complexity of ROP and similar techniques
- An example vulnerability
- Conclusions

Cellular baseband market 2013



LTE: Baseband market share distribution



Other players

- Intel (former Infineon stack):
  - shipping LTE-capable baseband XMM7160 (Galaxy Tab 3)
- Mediatek: announced LTE-capable baseband for 2014
- NVidia (former Icera stack)
  - prototypes of LTE-capable modem chipsets
  - Tegra 4i: integrated modem / app CPU SoC announced for early 2014
- Spreadtrum: TD-LTE + GSM chipsets available [no WCDMA] (SC9610)
- Broadcom: just bought Reneases in mid-2013 (former Nokia baseband)
  - MP6530 (ARM Big.Little + 3G/4G modem), allegedly ready for sale in 2014
- Ericsson: Future of Novathor unclear after ST-Ericsson breakup

Hexagon architecture

- Originated from QCOMs general purpose DSP
  - Used for only audio processing and L1 in early days
- VLIW architecture [1-4 instructions per cycle]
- Barrel processor (interleaved multithreading)
- 32-bit unified address space for code and data
  - Byte addressable
- 32 general registers (32-bit)
  - also usable pairwise: 64-bit register pairs
- Supports nestable (hardware) loops
- Many addressing modes (specific to DSP usage cases)

Design Goals

- Optimization for low-power
  (constraints orthogonal to security!)
- VLIW instead of OOO: lower footage and power
- Don't increase clock rate, increase work done per tick
- No speculation (branch/data prefetch)
- Avoid memory stalls
- Parallellism
  - Instruction-level
  - Data-level
  - Thread-level

Instruction packets

- Atomic units grouping instructions executed in parallel
- 4 parallel pipelines (called *slots*)
- Different ins. types assigned to different slots
- Constraints for grouping apply
  - HW resources cannot be oversubscribed
- Manuals: no branching into middle of packet
  - Empirically: you can return into middle of packet

Chipset evolution

- QDSP6v1: MSM8600
  - Pantech Racer Vega (anyone?!?)
- QDSP6v2: QSD8650 (v1/v2), MSM8200 (v1/v2), CSM8900, MDM8900
  - e.g. Sharp IS03/IS05
- QDSP6v3: MDM900 (v1/v2), CSM8700, FSM9000, QSD8650a, MDM8200a, MSM8660, QSD8x72
  - e.g. Sony Xperia acro HD IS12S
- QDSP6v4: MSM8960, MDM9x15
  - e.g. Samsung Galaxy S4 (GT-i9505), Apple iPhone 5, BlackBerry Z10
- QDSP6v5: MSM8974
  - e.g. LG G2, Sony Xperia Z Ultra

ISA Documentation & Toolchains

- Hexagon Programmer's guide available for v2, v4 & v5
- Building tools from scratch
  - Complex ISA (comparable to x86, but less docs)
  - Testing?
- Start from publicly released toolchain
  - GNU toolchain
  - LLVM Hexagon support looks very rough at the moment

QDSP6v2 vs QDSP6v3

- No reference manual for QDSP6v3 available
  - No changelog, diff opcode headers in Hexagon gnutools
- Instructions added
  - pause (for up to 263 cycles)
  - vrcmpys (vector reduce complex multiply by scalar)
- Undocumented:
  - tlblock / tlbunlock (obvious)
  - rteunlock (potentially related to runtime exceptions)
  - k0lock (no clue)
  - l2cleaninvidx (L2 cache cleaning/invalidation)
  - setimask (no clue)

QDSP6v3 vs. QDSP6v4 (according to doc)

- Virtualization instructions added
  - where?
- Support for SDR
  - Some of that was added in v3 already
- Debug & trace enhanced
- Larger address space

QDSP6v5 and up

V5:

- Floating point support
- “Enhanced data cache prefetch”

V55:

- Cycle count register
- Vector add and select maximum halfwords

Useful instructions

- Transfer: rX = rY || immediate
- ALU: Rd = add(Rs, Rt || immediate)
  [16 bit signed immediate for arithmetic, 10 bit for logical]
- combine: Rdd=combine(immediate, immediate)
  [8 bit signed immediates]
- MUX: Rd = mux(Pu, Rs||immediate, Rt||immediate)
  [8 bit signed immediates]
- NOP: `7f xx xx xx`

Leaked documentation

- Some while ago, archive of chipset docs on MSM8960 appeared on XDA Developers site
  - Someone put 7 AMSS security bulletins into this
- Performance: Leaked docs claim up Hexagon spends up to “3x fewer cycles than ARM9 on control code”

Control registers

- LC0 [C1], SA0 [C0],
  LC1 [C3], SA1 [C2]: Loop registers
- PC [C9]: Program counter
- USR [C8]: User status register
- M0 [C6]
  M1 [C7]: Modifier registers (circular addressing)
- P3:0 [C4]: Predicate registers
- UGP [C10]: User General Pointer (TLS)
- GP [C11]: Global Pointer

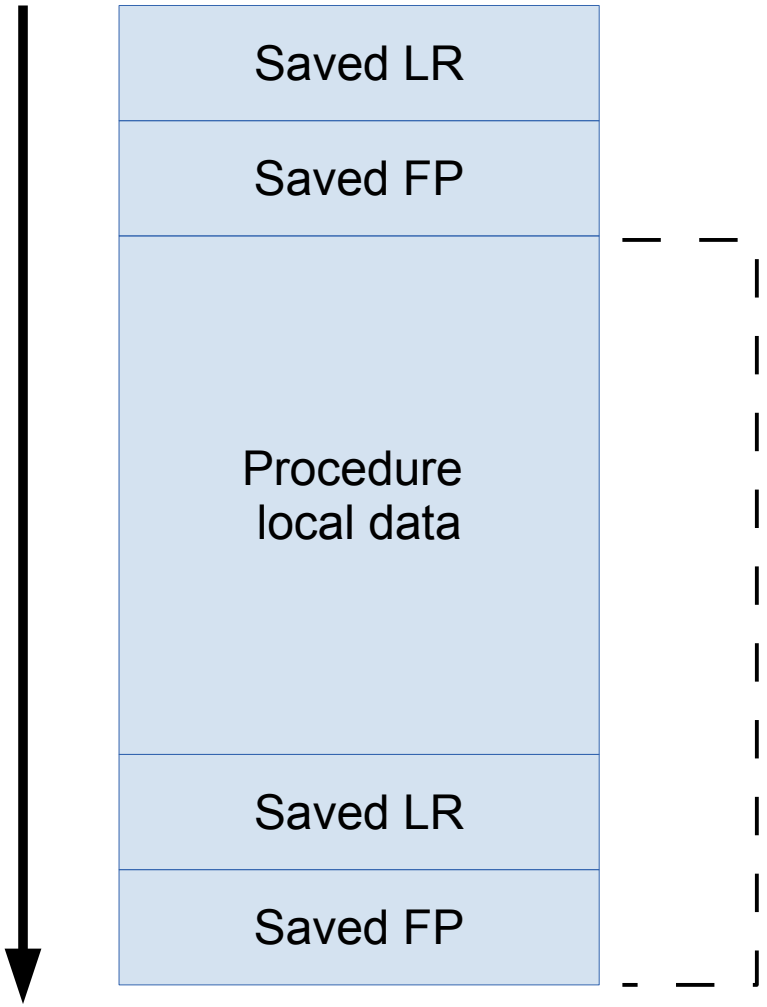
Calling conventions

R0 - R5	Parameters	Can be modified by callee
R6 - R15	Scratch	Caller [R14/R15 used for PLT]
R16 - R27	Scratch	Callee
R28	Scratch	Caller [used for PLT]
R29 - R31	Scratch	Callee [(de)allocframe]
P3:0	Processor state	Caller

- fill left to right R0-R5
- parameters up to 32 bit are passed in a register
- 64-bit parameters passed in register pairs
  pair is always (even,odd), skip registers to achieve
- remains on stack

Calling conventions

- `allocframe`(*size* [u14])
  - Push `LR` and `FP` to top of stack.
  - Subtract *size* [8-byte aligned] from `SP`
  - `FP` = addressof(oldFPonStack)
- `deallocframe`
  - Load saved FP and LR values from address referenced at FP
  - Restore SP to previous frame



Hexagon code, simple examples

```
some_func:
01 02 03 A3:  memw (r0 + #0xC) = r3 ; memw (r0 + #8) = r1
00 30 02 A4:  memw (r0 + #0x10) = r2 ; memw (r0 + #0) = #0
00 40 9F 52:  {   jumpr r31
80 C0 40 3C:      memw (r0 + #4) = #0 }


[...]
60 46 04 7C   {   r1:0 = combine (#0x33, #8)
46 42 33 04       immext (#0x43309180)
82 45 00 78   r2 = ##filename       @ "/local/mnt/" …
43 C1 03 78   r3 = #0x60A }
51 42 33 04   {   immext (#0x43309440)
A4 46 00 78       r4 = ##message        @ "<PRESENCE" …
00 40 5D 3C       memw (r29 + #0) = #0
80 C0 5D 3C       memw (r29 + #4) = #0 }
4A 63 64 5A   {   call logmsg
00 C1 5D 3C       memw (r29 + #8) = #0 }
[...]
```

Security of chip fabric

- Old(er) Qualcomm chipsets (e.g. MSM7200):
  - baseband was master (access to AP memory & flash)
- Current-gen chipsets have separate ARM7 core for bringup (RPM)
  - Modem firmware now is loaded by HLOS (e.g. Android, iOS)
- Chipset fabric has “hardware firewalls”
  - No documentation leaked on these
- Unclear whether baseband → AP escalation is possible
  - What about DMA?

New RTOS

- Very old Qualcomm chips use proprietary OS REX
- Later, REX was propped onto OKL4
  - commercial microkernel based on L4
- Hexagon-based baseband firmwares abandon OKL4
  - BLAST/QuRT apparently redesigned from scratch
  - Some remnants of REX for compatibility can be found

Security mitigations

- Stack cookies, generated by build toolchain
- Non-executable stack/heap
  - albeit, according to QCOM security advisory 80-N3172-14 (May 2012): "Enable Data Execution Prevention support in QuRT/BLAST-based images"
- Kernel/user-mode separation in QuRT/BLAST [also 80-N3172-14]
- Safe unlinking for heap
- No ASLR

" The customer must verify that any performance impact is acceptable."

[customer = OEM]

ROP & Roll

- Note that deallocframe sets FP
  - very similar to popping SP off stack on other architectures
- Instruction packets can be split
  - as long as they are not in cache
- Compound instructions are annoying
  - create constraints for gadgets
- For automation: use SMT solver to handle constraints
  - See BH 2010 talk & WOOT paper on same subject
- Still some way to go
- Manual gadget search works, but very labor-intensive
- Alternate gadgets ending in jump r31 and deallocframe gadgets to get work done

Hands-on training

- Smartphones: most modem firmwares signature checked at boot time (mostly older MDMs, though)
- USB modems: firmware freely modifiable
  [caveat, there may be exceptions: haven't seen any yet]
- Some Samsung Galaxy S4s (GT-i9505) with MSM8960: no signature check on modem firmware
  - Secure boot type: Samsung
- According to leaked docs modem bringup and sigcheck done by Krait core
  - SBL hacks may help with getting around checks

Tools

- QDSP6v5 toolchain released by QUIC
  - based on GCC 4.4
- Can be used to compile C/C++ code for Hexagon and inspect using objdump
- Modem firmware: empty ELF section header
  - need to populate to make objdump disassemble
- IDA Pro Hexagon plugin by GSMK (QDSP6v4)
  - also based on released binutils
  - very rudimentary at the moment
  - crashes on some firmwares (e.g. iPhone 5 baseband)

Leaked bugs: An example (CR 310629)

- Classic stack buffer overflow
- In LTE air interface
- Occurs when processing Test Loopback messages
  - Simple L3 messages > 100 bytes trigger this problem
- Mitigated by use of `-fstack-protector`
- Appeared in May 2012 security advisory
  - Detailed description given
- Still, surprising to see such straightforward bugs
  - Possible explanation: LTE stack was still “young”

Hexagon example code (fixed)

```
emm_process_incoming_tlb_msg:
allocframe (#0x90)
{ r2 = #0x78                @ sizeof(lte_tlb_dl_info_ind_s)
  call callee_save_r16_to_r19
  r3 = memw (gp + #0xB030) } @ retrieve stack canary
{ r17:16 = combine (r1, r29)
  r18 = r0
  r0 = r29 }
{ call memset_trampoline_0xA34440
  r1 = #0 ; memw (sp + #0x7C) = r3 } @ write stack canary
{ immext (#0xF00)
  r1:0 = combine (##0xF19, r29)
  immext (#0x4270400)
  r2 = ##0x4270403 }
call msgr_init_hdr_trampoline_0xA57000
{ p0 = cmp.gtu (r17, #0x64) @ if (tlb_msg_len > 100)
  if !p0.new jump:t tlb_msg_is_small_enough }
{ immext (#0x434C1880)
  r1:0 = combine (r17, ##ERRMSGSTRUCT_tlb_msg_too_large)
  call logmsg
  r3:2 = combine (#0, #0) }
jump check_canary
```

Hexagon example code (cont'd)

```
tlb_msg_is_small_enough:
{ r0 = add (r29, #0x12)
  memh (r16 + #0x10) = r17 }
r1 = r18 ; r2 = r17
call memcpy
[...]
check_canary:
r1 = memw (r29 + #0x7C)
r0 = memw (gp + #0xB030)
{ p0 = cmp.eq (r1, r0) ; if (p0.new) jump:t stack_is_good
  call stack_is_smashed }
stackis_good:
jump return_from_function
```

Analysis of more complex bugs

- Example given was shallow
- For more complex bugs, often messages need to be traced across tasks
  - Already became obvious during research for SUPL talk at BlackHat 2012
- Difficult without source code
- Given malleable baseband, perform dynamic analysis
- Monitor IPC (Message Router):
  - hook `msgr_send()`
  - uses UIDs
  - UID structure not publicly disclosed

The Way Forward: Offense

- New architecture raised bar of entry significantly
- However, Qualcomm dominates market
  - Attackers will have interest in their chips
- Well-funded attackers will adapt
- Public leaks of vulnerability information make attackers' task easier
  - Takedown possible, but the internet “doesn't forget”
  - Don't find bugs, find bug description
  - OEMs sometimes have slow patch cycles
- ROP exploitation needs automation
  - Not as difficult as assumed

The Way Forward: Defense

- Killing bugs & hardening is only one strategy
- After 3 years, still see the same silly problems
- Architectural changes:
  - Baseline: assume compromise, minimize
  - Why should baseband have access to microphone on smartphone? Or camera? Or GPS receiver?
  - Route through app CPU
- Shared-memory separation untested (externally)
  - Hard to make this verifiable without disclosing heaps of internals.

References

- Qualcomm: *Hexagon Application Binary Interface Specification*, 80-N2040-23 Rev. A, August 2013.
- Qualcomm: *Hexagon V4 Programmer's Reference Manual*, 80-N2040-9 Rev. A, August 2013.
- Qualcomm: *Hexagon V5/V55 Programmer's Reference Manual*, 80-N2040-8 Rev. A, August 2013.