OpenBSC network-side GSM stack

running on top of Linux

Harald Welte <laforge@gnumonks.org>

gnumonks.org
gpl-violations.org
OpenBSC
airprobe.org
hmw-consulting.de

Linux Kongress 2009, October 2009, Dresden/Germany

Outline

1. GSM/3G security
   - The closed GSM industry
   - Security implications
   - The GSM network
   - The GSM protocols
2. OpenBSC: Implementing GSM protocols
   - Getting started
   - OpenBSC software architecture
   - Code Reuse
3. Security analysis
   - Theory
   - The Baseband
   - Observations
4. Summary
   - What we've learned

About the speaker

- Always been fascinated by networking and communications
- Using + playing with Linux since 1994
- Kernel / bootloader / driver / firmware development since 1999
- IT security specialist, focus on network protocol security
- Board-level Electrical Engineering
- Always looking for interesting protocols (RFID, DECT, GSM)

GSM/3G protocol security

- Observation
  - Both GSM/3G and TCP/IP protocol specs are publicly available
  - The Internet protocol stack (Ethernet/Wifi/TCP/IP) receives lots of scrutiny
  - GSM networks are as widely deployed as the Internet
  - Yet, GSM/3G protocols receive no such scrutiny!
- There are reasons for that:
  - GSM industry is extremely closed (and closed-minded)
  - Only about 4 closed-source protocol stack implementations
  - GSM chipset makers never release any hardware documentation

The closed GSM industry

Handset manufacturing side

- Only very few companies build GSM/3.5G baseband chips today
    - Those companies buy the operating system kernel and the protocol stack from third parties
- Only very few handset makers are large enough to become a customer
    - Even they only get limited access to hardware documentation
    - Even they never really get access to the firmware source

The closed GSM industry

Network manufacturing side

- Only very few companies build GSM network equipment
  - Basically only Ericsson, Nokia-Siemens, Alcatel-Lucent and Huawei
  - Exception: Small equipment manufacturers for picocell / nanocell / femtocells / measurement devices and law enforcement equipment
- Only operators buy equipment from them
- Since the quantities are low, the prices are extremely high
  - e.g. for a BTS, easily 10-40k EUR

The closed GSM industry

Operator side

- Operators are mainly banks today
- Typical operator outsources
  - Billing
  - Network planning / deployment / servicing
- Operator just knows the closed equipment as shipped by manufacturer
- Very few people at an operator have knowledge of the protocol beyond what's needed for operations and maintenance

The closed GSM industry
Security implications

The security implications of the closed GSM industry are:

- Almost no people who have detailed technical knowledge outside the protocol stack or GSM network equipment manufacturers
- No independent research on protocol-level security
  - If there's security research at all, then only theoretical (like the A5/2 and A5/1 cryptanalysis)
  - Or on application level (e.g. mobile malware)
- No open source protocol implementations
  - which are key for making more people learn about the protocols
  - which enable quick prototyping/testing by modifying existing code

Security analysis of GSM
How would you get started?

If you were to start with GSM protocol level security analysis, where and how would you start?

- On the handset side?
  - Difficult since GSM firmware and protocol stacks are closed and proprietary
  - Even if you want to write your own protocol stack, the layer 1 hardware and signal processing is closed and undocumented, too
  - Known attempts
    - The TSM30 project as part of the THC GSM project
    - mados, an alternative OS for Nokia DTC3 phones
  - none of those projects successful so far

Security analysis of GSM

How would you get started?

If you were to start with GSM protocol level security analysis, where and how would you start?

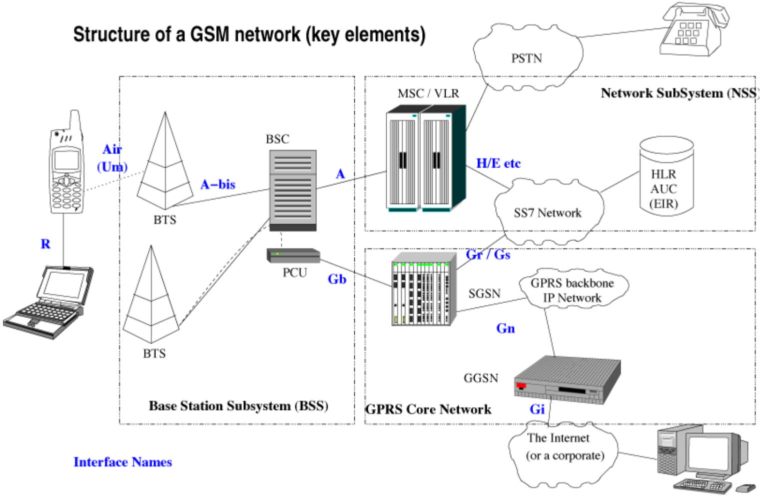
- On the network side?
  - Difficult since equipment is not easily available and normally extremely expensive
  - However, network is very modular and has many standardized/documented interfaces
  - Thus, if equipment is available, much easier/faster progress

Security analysis of GSM

The bootstrapping process

- Read GSM specs day and night (> 1000 PDF documents)
- Gradually grow knowledge about the protocols
- Obtain actual GSM network equipment (BTS)
- Try to get actual protocol traces as examples
- Start a complete protocol stack implementation from scratch
- Finally, go and play with GSM protocol security

The GSM network



GSM network components

- The BSS (Base Station Subsystem)
  - MS (Mobile Station): Your phone
  - BTS (Base Transceiver Station): The *cell tower*
  - BSC (Base Station Controller): Controlling up to hundreds of BTS
- The NSS (Network Sub System)
  - MSC (Mobile Switching Center): The central switch
  - HLR (Home Location Register): Database of subscribers
  - AUC (Authentication Center): Database of authentication keys
  - VLR (Visitor Location Register): For roaming users
  - EIR (Equipment Identity Register): To block stolen phones

GSM network interfaces

- Um: Interface between MS and BTS
  - the only interface that is specified over radio
- A-bis: Interface between BTS and BSC
- A: Interface between BSC and MSC
- B: Interface between MSC and other MSC

GSM networks are a prime example of an asymmetric distributed network, very different from the end-to-end transparent IP network.

GSM network protocols

On the Um interface

- Layer 1: Radio Layer, TS 04.04
- Layer 2: LAPDm, TS 04.06
- Layer 3: Radio Resource, Mobility Management, Call Control: TS 04.08
- Layer 4+: for USSD, SMS, LCS, ...

GSM network protocols
On the A-bis interface

- Layer 1: Typically E1 line, TS 08.54
- Layer 2: A variant of ISDN LAPD with fixed TEI's, TS 08.56
- Layer 3: OML (Organization and Maintenance Layer, TS 12.21)
- Layer 3: RSL (Radio Signalling Link, TS 08.58)
- Layer 4+: transparent messages that are sent to the MS via Um

Implementing GSM protocols

How I got started!

- In September 2008, we were first able to make the BTS active and see it on a phone
  - This is GSM900 BTS with 2 TRX at 2W output power (each)
  - A 48kg monster with attached antenna
  - 200W power consumption, passive cooling
  - E1 physical interface
- I didn't have much time at the time (day job at Openmoko)
- Started to read up on GSM specs whenever I could
- Bought a HFC-E1 based PCI E1 controller, has mISDN kernel support
- Found somebody in the GSM industry who provided protocol traces

Implementing GSM protocols
Timeline

- In November 2008, I started the development of OpenBSC
- In December 2008, we did a first demo at 25C3
- In January 2009, we had full voice call support
- In August 2009, we had the first field test with 2BTS and > 860 phones

OpenBSC: Overall architecture

- implement BSC, MSC, HLR, AUC, SMSC in a box
- Single-theaded, select-loop driven design
  - avoids locking/synchronization complexity
  - makes debugging much easier
  - amount of singalling traffic low, scalability on multi-core systems not a design goal
- Use Linux kernel coding style
- Have as few external dependencies as possible

OpenBSC: A-bis OML (GSM TS 08.59 / 12.21)

In order to fully boot and initialize a BTS, the OML (Organization and Maintenance Layer) needs to be brought up. It is implemented in OpenBSC `abis_nm.c`

- download/installation + activation of BTS software
- RF parameters such as ARFCN, hopping, channel configuration
- RF power level, calibration, E1 timeslot + TEI configuration

OpenBSC: A-bis RSL (GSM TS 08.58)

The Radio Signalling Link is the signalling layer between BTS and BSC, implemented in `abis_rsl.c`

- non-transparent messages for BTS-side configuration
  - channel activation on the BTS side
  - channel mode / encryption mode on BTS side
  - paging of MS
  - setting of BCCH beacons (SYSTEM INFORMATION)
- transparent messages to be passed through to MS

OpenBSC GSM Layer 3 (GSM TS 04.08)

The GSM Um Layer 3 is established between BSC and MS, the BTS transparently passes it through RSL DATA INDICATION / DATA REQUEST, implemented in `gsm_04_08_*.c`

- Radio Resource (RR)
- Mobility Management (MM)
- Call Control (CC)

OpenBSC: Input Drivers

- Concept of input drivers important, since there are many different E1 driver models and no clear standard (mISDN, VISDN, Sangoma, Zaptel)
  - We so far implement a socket-based input driver to the Linux kernel mISDN stack
  - Some proof-of-concept driver for Sangoma exists
- ip.access A-bis over IP interface is very different from E1 interface, but can still be supported by the input driver API
- Input drivers are not implemented as plugins, as we don't want proprietary plugins.

OpenBSC: mISDN integration

- Physical layer of A-bis is a E1 interface
- However, Layer 2 is slightly different to Q.921 on ISDN
  - static TEI assignments, no dynamic TEI's
  - different SAPI's are used for OML, RSL
  - multiple BTS can be connected to one E1 link, requiring multiple TEI manager instances to run in different timeslots on one E1 line
- Patches have been contributed to mISDN and are in mainline

OpenBSC: Multiple processes/Threads

- Currently, there is one single-threaded process for all of
  - The signalling part (BSC/MSC features)
  - Database access (HLR/VLR features)
  - Relaying/remultiplexing of speech data (TRAU + RTP frames)
  - SMS store-and-forward (SMSC features)
- Single-threaded select loop is great for signalling
- TRAU + RTP multiplexing / relaying should become separate media gateway process
- SMSC features should become independent process, too.

OpenBSC: Database model

- The HLR, EIR, SMSC are simple SQL tables
  - `subscribers` is the HLR (IMSI, phone number, tmsi, location area)
  - `equipment` is the EIR (IMEI, classmark1/2/3)
  - `sms` is the SMSC, one row for each SMS
- At the moment, only SQLite3 is used (simplicity)
- DBD layer will enable easy migration to postgresql or MySQL

OpenBSC: Code reuse

- Configuration file + interactive terminal: Reuse the VTY code from zebra/quagga project
  - "configure terminal; enable" style interface known to many network administrators
  - no need to handle persistent configuration different than run-time configuration
- Linked Lists: Imported code + API from Linux list_head
- Timers: Imported code + A PI from Linux kernel
- Core select loop handling: Stolen frm ulogd2 (netfilter/iptables)
- Database interface: Use dbi and dbd-sqlite3

OpenBSC: Voice call integration

- Integration with `lcr` (Linux Call Router)
  - Uses the OpenBSC codebase as library (libbsc.a)
  - Uses the 'call switching API' (MNCC) inside OpenBSC
  - Allows switching between ISDN and OpenBSC-based GSM
  - Has itself an interface for Asterisk VoIP
- Integration with Asterisk `chan_obenbsc`
  - Directly integrate OpenBSC as Asterisk channel driver
  - Ongoing effort by some community members
  - *Interesting* from a Licensing point of view !
- Integration with actual MSC
  - Allows OpenBSC to be used as true BSC in real GSM network

OpenBSC: GPRS support

- GPRS support is currently under active development
- Contrary to public belief, GPRS has very little relation to GSM beyond the physical layer
- OpenBSC is implementing SGSN and GGSN functionality for a *GPRS network in a box* apprach
- GPRS protocol stack of phone-originated HTTP request on a nanoBTS:
  - HTTP inside TCP inside IP (regular TCP/IP stack)
  - inside PPP, SNDCP and LLC (adaption of IP onto Um)
  - inside BSSGP and NS (Gb interf BTS - SGSN)
  - inside UDP inside IP inside Ethernet (ip.access encapsulation)

OpenBSC commercial interest

- On-Waves Inc. (Iceland), deploying small GSM networks like e.g. aboard ships
  - funding the development of a functional split between MSC/BSC to use OpenBSC as a true BSC (without MSC/HLR/SMSC/...)
  - funding the development of the A interface (the BSC-BTS network protocol stack)
- Netzing AG (Dresden/Germany), GSM networks for emergency applications
  - funding the development of ip.access nanoBTS support
- However, OpenBSC remains primarily a research tool for research use.

Known GSM security problems

Scientific papers, etc

- No mutual authentication between phone and network
  - leads to rogue network attacks
  - leads to man-in-the-middle attacks
  - is what enables IMSI-catchers
- Weak encryption algorithms
- Encryption is optional, user does never know when it's active or not
- DoS of the RACH by means of channel request flooding
- RRLP (Radio Resource Location Protocol)
  - the network can obtain GPS fix or even raw GSM data from the phone
  - combine that with the network not needing to authenticate itself

Known GSM security problems
The Baseband side

- GSM protocol stack always runs in a so-called baseband processor (BP)
- What is the baseband processor
  - Typically ARM7 (2G/2.5G phones) or ARM9 (3G/3.5G phones)
    - Runs some RTOS (often Nucleus, sometimes L4)
    - No memory protection between tasks
  - Some kind of DSP, model depends on vendor
    - Runs the digital signal processing for the RF Layer 1
    - Has hardware peripherals for A5 encryption
- The software stack on the baseband processor
  - is written in C and assembly
  - lacks any modern security features (stack protection, non-executable pages, address space randomization, ..)

Interesting observations

Learned from implementing the stack

While developing OpenBSC, we observed a number of interesting

- Many phones use their TMSI from the old network when they roam to a new network
- Various phones crash when confronted with incorrect messages. We didn't even start to intentionally send incorrect messages (!)
- There are tons of obscure options on the GSM spec which no real network uses. Potential attack vector by using rarely tested code paths.

Summary
What we've learned

- Until recently, there was no Open Source software for GSM protocols
- It is well-known that the security level of the GSM stacks is very low
- The GSM industry is making security analysis very difficult
- With OpenBSC and OpenBTS we now have tools for everyone
  - to learn more about and experiment with GSM protocols
  - to actually study protocol-level GSM security
  - to do penetration testing against GSM protocol stacks in phones

TODO

Where we go from here

- The tools for fuzzing mobile phone protocol stacks are available
- It is up to the security community to make use of those tools (!)
- Don't you too think that TCP/IP security is boring
- Join the GSM protocol security research projects
- Boldly go where no (free) man has gone before

Future plans

- Complete packet data (GPRS/EDGE) support in OpenBSC
  - GPRS is used extensively on modern smartphones
  - Enables us to play with those phones without a heavily filtered operator network
- UMTS(3G) support in OpenBSC
- Access to MS side layer 1
- Playing with SIM Toolkit from the operator side
- Playing with MMS
- More exploration of RRLP

Further Reading

- http://openbsc.gnumonks.org/
- http://airprobe.org/
- http://openbts.sourceforge.net/
- http://wiki.thc.org/gsm/