

# Injecting SMS Messages into Smart Phones for Security Analysis

**Collin Mulliner** <collin@sec.t-labs.tu-berlin.de>

**Deutsche Telekom Laboratories / Technical University Berlin**

Charlie Miller <cmiller@securityevaluators.com>

Independent Security Evaluators

USENIX WOOT August 10<sup>th</sup> 2009



# Agenda

- Contributions
- The Short Message Service
- The SMS Security Problem
- Analyzing SMS-Implementations
- SMS Delivery
- SMS Message Injection
- SMS Fuzzing
- Results
- Conclusions



# Contributions

- Novel method to test SMS-Implementations
  - Circumvent operator network
    - Deliver test original/unaltered messages to phone
    - Avoid bugs in telco equipment (test phone not network)
    - Don't crash infrastructure
    - Avoid paying per message fee
    - Operator doesn't see you testing
  - Framework for local SMS messaging injection
    - Lightweight software-only modification of the target phone
    - Higher speed than real mobile operator network



# The Short Message Service (SMS)

- Building block of the mobile phone service
  - Implemented on all networks and all devices
- Text messaging on the surface (for the end user)
  - Large revenue for the operators
- Binary messages for various services
  - Voice mail notification
  - OTA configuration
  - WAP, MMS, ring tones, ...
  - Custom applications



# The Problem with SMS

- Large and very complex feature set
- Implementation problems are common
  - Almost every mobile phone platform has known issues
  - So far issues only found by accident
- SMS-based attacks are hard to prevent
  - Real remote attacks as long as device is online
  - No user interaction required
  - SMS can't be switched off, phone side filters don't really exist
  - Network filters exist (operators don't like to talk about this)



# The Problem with SMS

- Large and very complex feature set
- Implementation problems are common
  - **Almost every mobile phone platform has known issues**
  - **So far issues only found by accident**
- SMS-based attacks are hard to prevent
  - Real remote attacks as long as device is online
  - No user interaction required
  - SMS can't be switched off, phone side filters don't really exist
  - Network filters exist (operators don't like to talk about this)
- **Need techniques and tools to analyze and improve the security of SMS-Implementations**



# Analyzing SMS-Implementations

- SMS analysis is difficult, multiple reasons:
  - SMS delivered through operator infrastructure
    - Tester would need to control or fight against infrastructure
  - Sending SMS messages costs money
    - Main cause why this has not yet been done in depth!
  - Most mobile phones are really closed systems
    - Source code is highly guarded company asset



# Analyzing SMS-Implementations

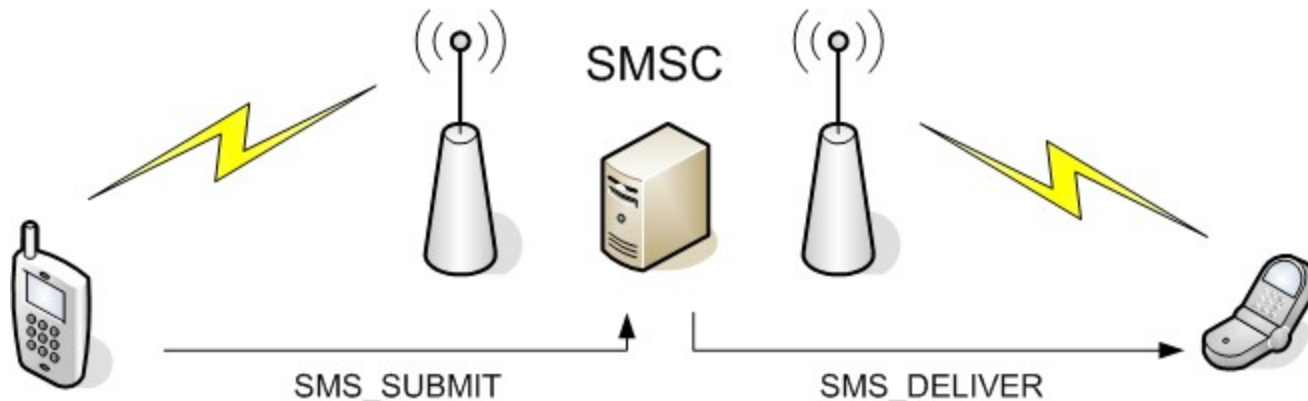
- SMS analysis is difficult, multiple reasons:
  - SMS delivered through operator infrastructure
    - Tester would need to control or fight against infrastructure
  - Sending SMS messages costs money
    - Main cause why this has not yet been done in depth!
  - Most mobile phones are really closed systems
    - Source code is highly guarded company asset
- **Removed the need for an mobile network infrastructure**
  - **Local SMS message injection**
- **Cost factor is cut out since operator is out of the game**
- **Fuzzing-based vulnerability analysis**
  - **Source code access not required**
  - **Quick results**





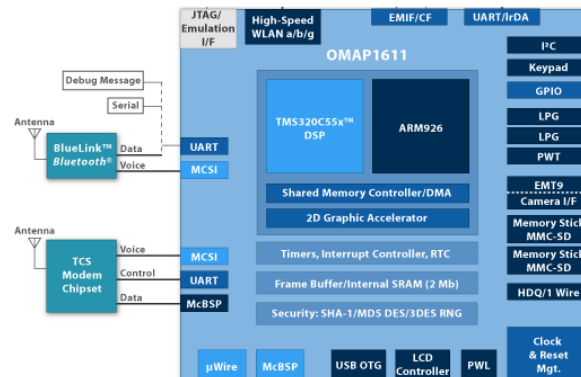
# SMS Delivery on the Network

- Store and forward message delivery
  - Sender submits to Short Message Service Center (SMSC)
  - Optional: sender's SMSC sends message to receiver's SMSC
  - SMSC delivers message to recipient
- Two SMS formats
  - SMS\_SUBMIT for sending (phone → SMSC)
  - SMS\_DELIVER for receiving (SMSC → phone)
- SMS\_DELIVER is what we use for testing



# SMS Delivery on the Phone side 1/2

- Smart phones are composed out of two processors
  - The application processor for the GUI and user applications
  - The modem handles the communication with the mobile phone network
  - The modem and the application processor are connected through a serial line interface
- The Telephony stack sits on top of the serial line
  - Controls the modem via the GSM AT command set
  - Provides API for applications (phone dialer, texting app, ...)



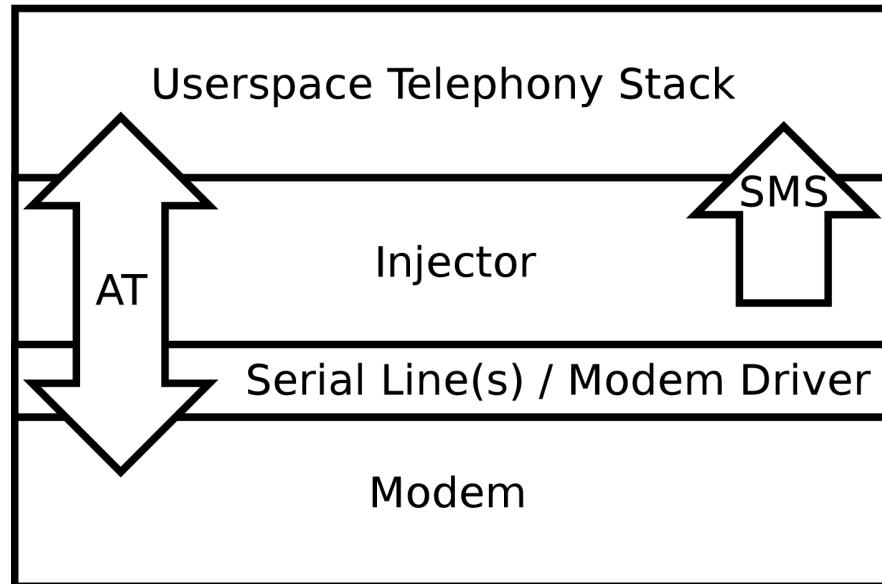
# SMS Delivery on the Phone side 2/2

- Unsolicited AT result code: +CMT
  - Modem issues the result on the serial line connected to application processor
  - +CMT: ,22  
07916163838450F84404D0110020009030329021810  
00704010200088000
- Telephony stack acknowledges message and pushes it up the stack to SMS applications
  - Text messaging and MMS app
  - System services (voice mail indication, OTA configuration, ...)
  - WAP stack
  - Custom SMS apps



# SMS Injection

- Man-in-the-Middle between modem and telephony stack
- Injects SMS message via +CMT result code
  - SMS messages are delivered to the injector via WiFi



# iPhone Injector

- Injector daemon opens modem lines
  - `/dev/dlci.[h5|spi]-baseband.3,4`
  - Publishes UNIX domain sockets for CommCenter
- CommCenter library injection via pre-loading
  - Hook `open(2)` to redirect serial lines to UNIX domain sockets that are connected to injector daemon



# Android Injector

- Single daemon to MITM on the serial line
  - Renames `/dev/smd0` to `/dev/smd0real`
  - Opens `/dev/smd0real`
  - Creates fake `/dev/smd0`
- Kill -9 33 (kills and restarts `/system/bin/rild`)
  - On restart rild will open fake `/dev/smd0`



# WindowsMobile Injector

- Replacement serial device driver
  - Based on open source AT command logging driver
  - Loads original serial driver to talk to the modem
- Log-driver was heavily modified for SMS injection
  - Added threading support for SMS message submission via TCP socket
- Installation requires multiple steps
  - Registry hacks (app unlock)
  - DLL signing, certificate installation, ...



# SMS Fuzzing

- Test case generation
  - Message generation using the Sulley fuzzing framework
  - Developed special SMS crafting library in Python
- Test cases
  - Standard text messages containing “problematic strings”
  - Multipart SMS
  - Voice mail notification
  - iPhone visual voice mail (non-standard app)
  - Port addressing (SMS supports TCP/IP like ports 0-65535)
    - Sending garbage to a random port, e.g. WAPpush at 2948
    - Port scanning
- Send test cases to phone via WiFi
  - Injection daemon on device reads test cases from TCP:4223





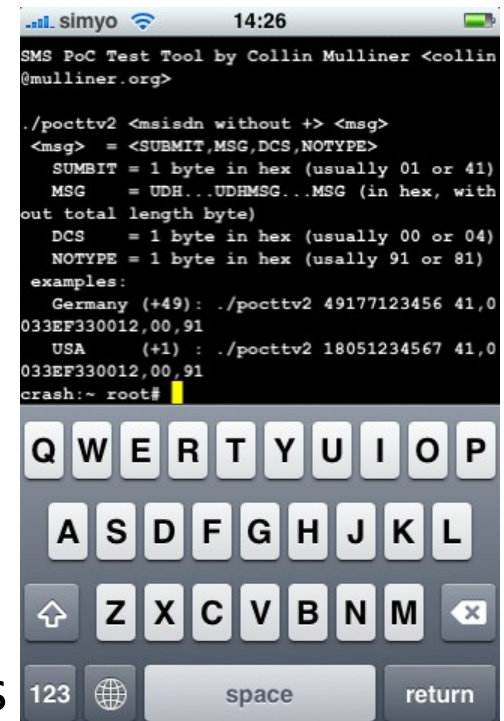
# SMS Fuzz Monitoring

- Need to monitor fuzzed app to catch the actual bugs
  - Best case: fuzz over night and collect results in the morning
- Device specific monitoring required
  - iPhone
    - Look for crash dumps from crash reporter
  - Android
    - Monitor device via Android Debug Bridge (ADB)
  - WinMobile
    - Attach debugger to SMS apps (tmail.exe, Manila2D.exe, ...)
    - Unfortunately only manual crash recovery
- Check for problems not related to a crash
  - Send valid SMS and inspect if it arrives in SMS database



# From Bug to Attack

- Not all bugs found through fuzzing can be send over the network
  - Test-send fuzzing results (the beef) over the network
  - Messages that go through are real attacks
- Small application that runs on an iPhone
  - Easy testing while logged in via SSH
  - Awesome demo tool using mobile terminal
- Test different operators
  - Not all operators allow all kinds of messages
    - May not be able to attack people on all networks



```
simyo 14:26
SMS PoC Test Tool by Collin Mulliner <collin@mulliner.org>

./pocppv2 <msisdn without +> <msg>
<msg> = <SUBMIT,MSG,DCS,NOTYPE>
SUBMIT = 1 byte in hex (usually 01 or 41)
MSG = UDH...UDHMSG...MSG (in hex, with out total length byte)
DCS = 1 byte in hex (usually 00 or 04)
NOTYPE = 1 byte in hex (usually 91 or 81)
examples:
Germany (+49): ./pocppv2 49177123456 41,0 033EF330012,00,91
USA (+1) : ./pocppv2 18051234567 41,0 033EF330012,00,91
crash:~ root#
```



# iPhone Fuzzing Results

- Multiple Denial-of-Service attacks and one code execution
  - OS versions 2.2, 2.2.1, 3.0
- CommCenter memory corruption
  - Allows to control program counter (code execution)
    - Needs 519 SMS messages (user only sees 1 message)
  - Crashing CommCenter kicks phone off the network (DoS)
    - Also kills all other network connections (WiFi & Bluetooth)
    - Phone call in progress is interrupted!
- SpringBoard crash (nullptr dereference)
  - Locks iPhone (user has to: slide to unlock)
  - Blocks iPhone for about 15 seconds



# Android Fuzzing Results

- Denial-of-Service against com.android.phone
  - Kicks Android phone off the mobile phone network
  - Restart of com.android.phone locks SIM card if SIM has a PIN set, phone can no longer register with network
  - Attack is silent, user does not see or hear it
    - User is unreachable until he checks his phone!
- Attack possible with different bugs
  - OS versions 1.0, 1.1, and 1.5

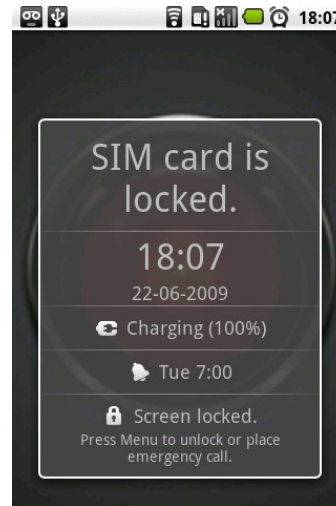
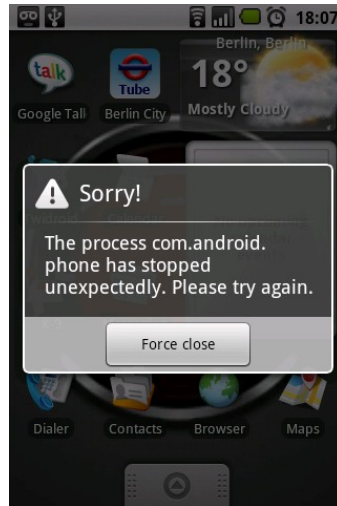


# Windows Mobile Fuzzing Results

- Format String Vulnerability
  - HTC Touch 3G (Windows Mobile 6.1)
  - Manial2D.exe (TouchFLO by HTC)
  - Classic %n
  - Allows to control PC → code execution
  - Denial-of-Service
    - App doesn't restart as long as the *bad SMS* is in the inbox
    - TouchFLO interface is completely blocked
    - In this case the fix is easy (if you know what to do)
      - Just delete the *bad SMS* using the Windows Mobile SMS app instead of using TouchFLO



# Results



# Conclusions

- We have developed a novel way for performing vulnerability analysis of SMS-Implementations
  - Removes cost factor → enables large scale fuzz-based testing
    - We sent more than 500K SMS messages during testing
  - No interference with mobile operator network → results are reproducible and conclusive
- We identified a number of new vulnerabilities that can be used for Denial-of-Service attacks and code execution
  - DoS is a real problem in the mobile communication world
  - Found security issues for all our test platforms
- Future work
  - Port the framework to other platforms
  - Injector provides cost free and unfiltered path to send SMS messages to a phone → use it for other kinds of tests



# End, Any Questions?

Thank you for your time!

