Fuzzing the Phone in your Phone

Authors: Charlie Miller and Collin Mulliner
Presented by: Jevin Sweval

Presentation Overview

• Background information
  – Smartphone overview
  – SMS overview
• Fuzzing Framework
• Results
There is a Phone *in my* Phone!?

- Today’s smartphones contain two processors
- **Application processor**
  - Runs main OS (iOS, Android, etc.) and applications
  - Fast (up to 1 GHz dual core) ARM processor
- **Baseband processor**
  - AKA: modem, mobile station modem (MSM)
  - System-on-Chip with low-power ARM processor and RF modules (GSM, 3G, WiFi, Bluetooth)
    - May be on same package as application processor
  - Runs a RTOS (Nucleos OS on iPhone)

Baseband Overview

- The baseband is usually tasked with handling all of the radio communications on your phone
- Baseband interfaces to main CPU using old-school serial ports!
  - Still use 1970s AT modem commands
  - Serial ports may either be physical UARTs or logical ports over another interface (SPI, I2C, etc)
Why focus on the Baseband?

• The operation of the baseband is largely outside of the main OS’s control
  – Not subject to security measures within the main OS
• The baseband responds to and processes unsolicited radio messages
  – No user interaction needed
  – Interface using the legitimate carrier’s network or even a malicious microcell tower
  – A great attack vector!

SMS Overview

• How do you attack the baseband?
  – Use SMS
• Short Message Service
  – Utilizes a sideband in the cell service’s control channel
  – Messages can be 140 bytes (160 7-bit characters)
    • Originally used for carrier housekeeping
      – e.g. Pay-as-you-go balance info, voicemail notifications
    • Expanded use to “text messaging” where it became wildly popular
SMS Overview

- SMS is not firewalled
  - Necessary for normal operation of the phone
- SMS is processed without explicit user interaction
- Allows targeting with a static phone number
  - Want to attack a high value target? Just look them up in a directory!

Format of a SMS Message

- Arrives unsolicited over the serial line as an AT command result
  +CMT: ,30
  8791947196004034040091947196466656F80000901082114215400AEB329BF4697D9D9EC377
- The long hex string is the actual SMS, encoded in PDU (protocol data unit) format
  - Just like any other message protocol, PDU contains many fields related to network control and functionality
- Some PDU fields control how the message is interpreted
- A sample of message types identified in the paper
  - SMS
  - Voicemail indicator
  - Visual Voicemail
  - Push notifications
Fuzzing SMS

- Authors performed basic fuzzing by mutating fields in SMS PDUs
  - Mutate observed, valid PDUs or generate/mutate new PDUs from RFC specs
  - More targeted than generating random bitstrings
  - Authors used custom tool to generate fuzzed PDUs
- Send the PDU to the target
- Watch and log faults (possible vulnerabilities)

Delivering SMS

- Fuzzing is probabilistic by nature and requires many, many individual fuzzing attempts (sending a SMS)
- Sending these SMS over the carrier is problematic
  - Costs money
  - Sending lots of SMS may violate TOS or get throttled
  - Carrier may modify/filter certain SMS
- Could you send the SMS over your own carrier?
  - This is possible using software radio + OpenBTS/OpenBSC
  - Authors did exactly this for a later presentation
- Inject the SMS into the phone via software
Micro-cell used by authors for CanSecWest 2011 Presentation

SMS Injection

• Use software to inject SMS for free and without carrier interference
• Create a MITM proxy between the baseband and application processors
  – Passes uninteresting traffic without modification
    • Also allows for sniffing, reverse engineering
  – Allows for modification and creation of SMS messages
SMS Injection on the iPhone

• The CommCenter process on the application CPU handles all communication with the baseband
  – Runs as root and without sandboxing!
  – Connects to baseband using serial character devices in /dev/
• Hook open() within CommCenter and detect when the baseband device is opened
• When the device is opened, return a file descriptor of a UNIX socket, not the real device

SMS Injection on the iPhone

• The proxy runs as a daemon, proxying traffic between the real device and the UNIX socket
• You can inject SMS messages simply by writing the PDU to the UNIX socket
• The proxy listens on a TCP port to receive commands and fuzzing data over WiFi
Hooking open()

- Darwin’s (OS X, iOS kernel) dynamic runtime linker offers library pre-loading
  - Add your hooked version of open() to a library specified in the DYLD_INSERT_LIBRARIES env-var
  - Calls within the application to open() actually call your hook library
  - Can’t use pre-loading with statically linked executables
    - Luckily, all iPhone executables are dynamically linked
- Same technique is available on Linux (LD_PRELOAD) and Windows (DLL injection)

iPhone SMS Injection Details

- Detect crashes using iOS’s built-in CrashReporter facility
  - CrashReporter places crash logs into a directory on the phone
  - Check the directory for new crashes with SSH commands sent by the fuzzing computer
- The test suite must check that the phone is still processing new messages and hasn’t crashed
  - After every fuzzing test, send a valid SMS and check for correct delivery
Android and Windows Phone Injection

• Android
  – Rename serial device (no need to hook open())
  – “CommCenter” equivalent is written in Java
    • Harder to attack
• Windows Phone
  – Authors created a new kernel driver for a virtual, proxy serial port
    • Authors did not discuss why DLL injection wasn’t used

Results

• Not all interesting fuzzes can be sent over the carrier network
  – Intermediate network nodes may drop/modify the fuzzed SMS
  – Must verify all attacks on real networks
• Exploits created
  – DoS attacks for iPhone, Android, and Windows Phone
  – Remote Code Execution on iPhone
DoS Results

- iPhone
  - Malicious SMS crashes CommCenter, killing all network connections (WWAN, WiFi, Bluetooth)
  - A different malicious SMS can crash the window manager, freezing the device for about 15 seconds
- Android
  - Malicious SMS can crash the Phone app, knocking the device off the network
  - When the Phone app restarts, it locks the SIM card, blocking network registration!
- Windows Phone
  - Malicious SMS can crash HTC’s custom UI
  - Custom UI will not recover while the malicious SMS is still in the inbox!
    - Message must be deleted using Microsoft’s SMS app

iPhone Remote Code Execution

- A series of 519 SMS messages can enable arbitrary remote code execution
- Just one of these messages is visible to the user – user is unaware of the deluge of SMS
- Authors sent the 519 SMS at a rate of 1/second
  - Did not specify if they could send faster or if any carrier throttling occured
Jevin’s Takeaway

• There is more research to be done on smartphone baseband security
• SMS is a key attack vector for mobile phones
  – All phones have it, it isn’t firewalled, it requires little to no user interaction
• Critique: authors focus on vulnerabilities within the application CPU, not within the baseband CPU
  – Could you install a rootkit in the baseband that is undetectable from the application CPU?
• Injection and hooking can be useful for fuzzing and other synthetic testing situations