Fuzzing the Phone in your Phone

Collin Mulliner
Technical University Berlin
<collin@mulliner.org>

SEC-T
Stockholm, Sweden
September 10th 2009
About me

• PhD Student at TU-Berlin (Germany)
• Specialized in mobile and smart phone security
• Previous work:
  • MMS remote exploit for WinMobile in 2006
  • Hacked: WinMobile, Symbian, iPhone, NFC, Bluetooth, to name a few.
My Co-Author

- Charlie Miller
  - Security Researcher at Independent Security Evaluators
- Claim to fame:
  - First one to hack the iPhone and G1 Phone
  - Pwn2Own winner 2008 and 2009
Agenda

- SMS
- Fuzzing SMS
- iPhone injection
- Android injection
- WinMobile injection
- Some fuzzing results
SMS

- Uses extra bandwidth in control channel (used for establishing calls, status, etc.)
- Message data limited to 140 bytes (160 7-bit chars.)
- Commonly used for “text messages”
- Can also deliver binary data:
  - OTA configuration
  - ringtones
- Building block for the essential mobile phone service
Why pick on SMS?

- SMS is received by and processed by almost all phones
- No way to firewall it (and still receive calls/texts)
- SMS is processed with no user interaction
  - Server side attack surface with no firewall, I'm having a 1990's flashback!
- Can be targeted with only a phone number
- SMS firewalls/filters exist on the network but those on the phones are too high in the stack to protect against these attacks
The life of an SMS message

- Message is sent from the device to the Short Message Service Center (SMSC)
- The SMSC forwards to the recipient, either directly or through another SMSC
- SMSC will queue messages if recipient is not available
- Delivery is best effort, no guarantee it will arrive
On the device

- Phones have 2 processors, application processor and modem
- Modem runs a specialized real-time operating system that handles all communication with the cellular network
- Communication between CPUs via logical serial lines
- Text based GSM AT command set is used
Looking inside
Continued life of an SMS

- When an SMS arrives at the modem, the modem uses an unsolicited AT command result code
- This consists of 2 lines of text:
  - The result code and the number of bytes of the next line
  - The actual SMS message (in PDU mode)

```
+CMT: ,30
0791947106004034040D91947196466656F800009010821142
15400AE8329BFD4697D97D9EC377D
```
### A PDU

```
07 91947106004034040D91947196466656F80000901082114215400AE8329BFD4697D9EC377D
```
But there is more

- The previous PDU was the most simple message possible, 7-bit immediate alert (i.e. a text message)
- Can also send binary data in the UD field
- This is prefaced with the User Data Header (UDH)
UDH example

050003000301

<table>
<thead>
<tr>
<th>Field</th>
<th>Size</th>
<th>Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>UDHL</td>
<td>1 byte</td>
<td>05</td>
</tr>
<tr>
<td>IEI</td>
<td>1 byte</td>
<td>00</td>
</tr>
<tr>
<td>IEDL</td>
<td>1 byte</td>
<td>03</td>
</tr>
<tr>
<td>IED</td>
<td>Variable</td>
<td>000301</td>
</tr>
</tbody>
</table>

Collin Mulliner                        SEC-T 2009                        Fuzzing the Phone in your Phone
UDH example

050003000301

- Concatenated messages
  - Can send more than 140/160 bytes
  - IEI = 0 → concatenated with 8 bit reference number
  - IEDL = 03 → 3 bytes of data
  - Reference number = 00
  - Total number of message parts = 03
  - This message number = 01
Other common UDH IEIs

- IEI 01 = voice mail indicator
- IEI 05 = port numbers (applications can register)
  - Port 5499 = iPhone visual voicemail
    - allntxacds12.attwireless.net:5400?
      f=0&v=400&m=XXXXXX&p=&s=5433&t=4:XXXXXX:A:IndyAP36ms:ms01:client:46173
  - Port 2948 = WAP push
PDU Spy

Fuzzing SMS
Fuzzing 101

• Create malformed input
  • Take existing input and “mutate” it
  • Create inputs from scratch (from RFC, for example)
• Send to target
• Monitor for faults
• Goto step 1
Unmanned fuzzing exploration

• The ultimate goal of a fuzzing harness is complete automation
  • Record interesting events for human analysis
  • Detect and restart if service hangs/crashes
  • Handle dialogue boxes and other UI
  • Reboot if necessary
Creating test cases

- Can take some sample PDUs and mutate
  - These aren't exactly easy to find!
- Might as well use our knowledge of protocol to generate intelligent test cases
- We can use Sulley fuzzing framework
  - This is how Charlie did it
- Build a SMS crafting library to generate messages
  - This is how I did it
SMS crafting library

- Support SMS_DELIVER and SMS_SUBMIT
  - DELIVER is used for fuzzing!
  - Can generate and parse PDUs
- UDH support, IEIs:
  - Port Addressing 8 + 16 bit
  - Multipart messages
  - Indication (voice mail, etc...)
- All PDU fields can be auto-filled or set by hand!
Some SMS test cases

- Multipart messages
- Port addressing
  - "Portscanning" → send random data to every port
  - WAP Push → send "less" random data to port 2948
- UDH bomb
  - Build a number of UDHs with valid length fields and random data, put all UDHs in same SMS message
- Voicemail indication
SMS library

- Add-on utilities to store, load, and send test cases to/from a file
- Written in Python
- Released at SEC-T
- http://www.mulliner.org/security/sms/
Sending the test cases

- Could send over the air
  - Costs $$$$/€€€€
  - Telcos get to watch you fuzz
  - You might (make that WILL) crash Telco's equipment
- Could build your own transmitter
  - That is hard!
- Could inject into the process which parses
  - Would be very device/firmware dependent
SMS injection

- We Man-in-the-Middle the channel between the application processor and the modem
- Can send messages quickly
- It's free
- Requires no special equipment
- The receiving process doesn't know the messages weren't legit
- Telco (mostly) doesn't know it's happening
- Warning: results have to be verified over the carrier network
SMS injection

![Diagram showing the flow of SMS injection]

- Userspace Telephony Application
- Injector
- Virtual Serial Line(s)
- Modem

Collin Mulliner
SEC-T 2009
Fuzzing the Phone in your Phone
Get SMS sniffing for free

- Log AT commands as you forward them
- Useful for RE'ing apps that register SMS ports, vendor, specific SMS data, etc...

```
ssfd3 connected
/dev/dlci.spi-baseband.3 opened
ssfd4 connected
/dev/dlci.spi-baseband.4 opened
csf3 to fd3 write 5 bytes
---
ate0^M
+++ 
csf4 to fd4 write 5 bytes
...
csf3 to fd3 write 35 bytes
---
00100b8.....
```
Speaking of free...

• Free to test with the injector
• We sent thousands of fuzzed SMS's during fuzzing
• We sent thousands of fuzzed SMS's during exploit dev
• Injector makes this whole thing possible
iPhone injection
iPhone SMS fun fact

- The CommCenter process is responsible for handling SMS and Telephone calls. It runs as root with no application sandbox.
iPhone SMS

- CommCenter communicates with the modem using 16 virtual serial lines
  - `/dev/dlci.h5-baseband.[0-15] (2G)
  - `/dev/dlci.spi-baseband.[0-15] (3G)`
Man-in-the-Middle

• Use Library Pre-loading to hook basic API

• com.apple.CommCenter.plist:

```xml
...<key>EnvironmentVariables</key>
  <dict>
    <key>DYLD_FORCE_FLAT_NAMESPACE</key>
    <string>1</string>
    <key>DYLD_INSERT_LIBRARIES</key>
    <string>/System/Library/Test/libopen.0.dylib</string>
  </dict>
...
```
#define FD3 "/tmp/fuzz3.sock"

Int open(const char *path, int flags, …)
{
    real_open = dlsym(RTLD_NEXT, "open");
    if ((strncmp("/dev/dlci.h5-baseband.3", path, 23) == 0 ||
         (strncmp("/dev/dlci.spi-baseband.3", path, 24) == 0)) {

        struct sockaddr_un saun;
        fd = socket(AF_UNIX, SOCK_STREAM, 0);
        saun.sun_family = AF_UNIX;
        strcpy(saun.sun_path, FD3);
        int len = offsetof(struct sockaddr_un, sun_path) + strlen(FD3);
        connect(fd, &saun, len);
        fd3 = fd;
    } else {
        fd = real_open(path, flags);
    }
    return fd;
}
The injection

- CommCenter thinks it opened the serial line, but actually it opened up a UNIX-domain socket
- A daemon runs which opens up the real serial line and copies all data to and from the UNIX-domain socket
- Daemon also listens on TCP port 4223 and writes all data read from the port on the socket
- Therefore, can inject AT command over TCP
Sending PDUs

def send_pdu(ip_address, line):
    leng = (len(line) / 2) - 8
    buffer = "\n+CMT: %d\n%s\n", % (leng, line)
    s = connect((ip_address, 4223))
    s.send(buffer)
    s.close()
def check_for_crash(test_number, ip):
    Commcenter = '/private/var/logs/CrashReporter/
    LatestCrash.plist'
    Springboard = '/private/var/mobile/Library/Logs/
    CrashReporter/LatestCrash.plist'
    command = 'ssh root@'+ip+' "cat %s 2>/dev/null; cat %s
    2>/dev/null"' % (commcenter, springboard)
    c = os.popen(command)
    crash = c.read()
    if crash:
        clean_clogs()
        print "CRASH with %d" % test_number
        print crash
        time.sleep(60)
    else:
        print ' . '
    c.close()
Final checks

- To make sure the device is still handling SMS messages send a legit message between each test case and make sure it is processed

- SMS message show up in the sqlite database /private/var/mobile/Library/SMS/sms.db

- Display contents of last message received:

  ```
  # sqlite3 -line /private/var/mobile/Library/SMS/sms.db
  'select text from message where ROWID = (select MAX(ROWID)
  from message);'
  ```
iPhone IEI support

0x0, 0x1, 0x5, 0x8, 0x22

Collin Mulliner        SEC-T 2009        Fuzzing the Phone in your Phone
Android injection

New text message at 11:49 AM
Hey want to grab lunch today?

Close  Reply
Android fuzzing fun-fact

- Process which handles SMS is a Java app :(
Android MITM

- Rename serial device from: `/dev/smd0` to `/dev/smd0real`
- Start injector daemon, daemon will create fake `/dev/smd0`
- Kill -9 33 (kills `/system/bin/rild`)
- When rild restarts it talks to the injector daemon via `/dev/smd0`
Sending test cases

- Identical to iPhone case, use TCP 4223

Inject SMS messages over WiFi
Crash monitoring

- Monitor output of ADB (Android Debug Bridge)
  - `logcat -d` gives you the logdump
  - "*** *** ***" indicates a CRASH
  - "uncaught exception" indicates a Java crash
  - Automized with a small Python script...
Valid test case injection

• Same as iPhone except the sqlite command is:

/system/xbin/sqlite3 -line /data/data/com.android.providers.telephony/databases/mmssms.db 'select body from sms where id = (select MAX(_id) from sms);'
Android is not sturdy

• It is easy to make the SMS app unresponsive (in fact it is hard not to)

• When things hang:
  • /data/busybox/killall -9 com.android.phone
  • /data/busybox/killall -9 com.android.mms

• When things are really broken (this is almost a reboot):
  • /data/busybox/killall -9 system_server
Windows Mobile injection
Not surprisingly

- Things are a little different in Windows Mobile
- Need all kinds of hacks
- “app unlock” device (registry hacks)
MITM Kernel Style

- Add new serial driver
- Driver provides same interface as original driver
- Uses original driver to talk to modem
- Open TCP port 4223
- Built on top of Willem Hengeveld's log-driver
  - Thanks for your help!
SMS injection

• Same as iPhone and Android :-)
Monitoring

- Done with IDA Windows Mobile remote debugger
- Multiple process to monitor
  - tmail.exe → SMS/MMs app from Microsoft
  - Manila2D.exe → TouchFLO GUI from HTC
Some fuzzing results

[Image: A cartoon with two characters. One says, "It's a bug! It's a bug!"]

[Image: Another character says, "It's a piece of fuzz!"]
From potential bug to attack

• Not all bugs found through injection can be sent over the phone network
  • Test-send fuzzing results over the network
  • Messages that go through are real attacks
• We built a small application that runs on the iPhone
  • Easy testing while logged in via SSH
  • Awesome demo tool via mobile terminal
• Test different operators
  • Not all operators allow all kinds of messages
  • May not be able to attack people on all networks
Send over the network

- Open /dev/tty.debug
- Read/write AT commands to send message
iPhone SMS DoS

- iPhone
  - Crashing CommCenter kicks phone off the network
  - Kills all other network connections (WiFi + Bluetooth)
  - Phone call in progress is interrupted!
  - Repeat as necessary

- SpringBoard crash
  - Locks iPhone (user has to: slide to unlock)
  - Blocks iPhone for about 15 seconds
Digging the DoS
Android SMS DoS

- 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!
DoS

The process com.android.phone has stopped unexpectedly. Please try again.

Collin Mulliner  SEC-T 2009  Fuzzing the Phone in your Phone
Windows Mobile DoS

- HTC Touch 3G (Windows Mobile 6.1)
  - Manil2D.exe (TouchFLO by HTC) crashes
    - App doesn't restart as long as the bad SMS is in the inbox
    - TouchFLO interface will not restart
- 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 TouchFLO
Windows Mobile DoS

Es gab ein Problem mit Manila2D.exe.

Bitte leiten Sie dieses Problem an Microsoft weiter. Ihnen entstehen dabei keine Kosten. Diese Informationen dienen ausschließlich der Verbesserung unserer Produkte.

Fehlerdetails anzeigen
Fehlerbericht deaktivieren
The Demo we did at Black Hat

• Send iPhone CommCenter DoS SMS for 1 hour
  • One message every 10 seconds
• Victim was not able to use his iPhone during the talk and for about 2.5 hours after the talk
  • SMS messages queued up at the SMSC
  • Everytime the phone came back online it got the next message that was waiting for him → bang offline again
iPhone SMS code exec summary

- I'm not Charlie, I can write exploits but haven't done it for the iPhone.

- The story:
  - 519 SMS's (@ 1/sec), only one shows up to the user
  - Can control program counter (PC)
  - Could only be found with “smart” fuzzing
Android DoS

- Send any SMS to port 2948 (WAP push)
- Get java.lang.ArrayIndexOutOfBoundsException
- Knocks phone off network for a few seconds
- Works on European carriers, not on AT&T
  - Bug would not have been found if we had tested only in the US and on AT&T!
ADB logcat output

I/ActivityManager(56): Stopping service: com.android.mms/.transaction.TransactionService
D/dalvikvm(7099): GC freed 2614 objects / 148896 bytes in 134ms
W/AudioFlinger(35): write blocked for 97 msecs
D/WAP PUSH(7085): Rx:
0606436b46673774261b69195d187d2b1610370c39456f5b3b58540e3c650b21542141630b6c214764240e707e5c533e0b1143090c4078de77705714193c1a2937066d75141c1835144753565d602f6a67152a7807106d35334a721454177456925640a11335a3b30461145307d04df7b
D/AndroidRuntime(7085): Shutting down VM
W/dalvikvm(7085): threadid=3: thread exiting with uncaused exception (group=0x4000fe70)
E/AndroidRuntime(7085): Uncaught handler: thread main exiting due to uncaused exception
E/AndroidRuntime(7085): java.lang.ArrayIndexOutOfBoundsException
E/AndroidRuntime(7085):  at com.android.internal.telephony.WspTypeDecoder.decodeExtensionMedia(WspTypeDecoder.java:200)
E/AndroidRuntime(7085):  at com.android.internal.telephony.WspTypeDecoder.decodeConstrainedEncoding(WspTypeDecoder.java:222)
E/AndroidRuntime(7085):  at com.android.internal.telephony.WspTypeDecoder.decodeContentType(WspTypeDecoder.java:239)
Windows Mobile results

- Format string bug in Manila2D.exe (TouchFLO)
- This is the user interface for HTC devices
- A simple text message containing “%n” crashes TouchFLO
- Format string should make it exploitable!

07919471173254F6040C91947167209508000099309251619580022537
As seen in IDA Debugger
Conclusions

- SMS is a great vector of attacks against smart phones
- SMS fuzzing doesn't have to be limited by equipment or cost of sending SMS
- Can inject SMS using software only by MITM the modem
- Can find some bugs, keep on fuzzing!
Firmware Updates

- Android CRC1 also fixes our WAP push DoS bug
  - Released about 2 weeks after we reported the bug
- iPhone OS 3.0.1 was released on July 31th
  - ONLY fixes our CommCenter bug :-(
- HTC told us the bug in TouchFLO is fixed
  - ROM Build 1.00.19153530.00 (this is the HTC Touch 3G)
  - Haven't found a way to download/install it :-(

Collin Mulliner  SEC-T 2009  Fuzzing the Phone in your Phone
Thanks

- Charlie Miller for being a cool co-author :-) 
- Willem Hengeveld for his WinMobile log-driver
Questions

- All tools and slides at:
  - http://www.mulliner.org/security/sms/
- Contact: collin@mulliner.org