Evolution of the iPhone Baseband and Unlocks

@MuscleNerd
iPhone Dev Team
Hack in the Box, Amsterdam
May 24, 2012
My background

• Member of iPhone Dev Team
  • [http://blog.iphone-dev.org](http://blog.iphone-dev.org) (133 million visits to date!)
• Initially just interested in baseband, but now also maintain and extend “redsn0w” jailbreak utility
  • custom ramdisks, blob stitching, downgrades, etc
• Tech editor for *iOS Hacker’s Handbook* by Miller, Blazakis, DaiZovi, Esser, Iozzo, Weinmann (2012)
  • `<musclenerd@iphone-dev.org>`
General BB environment

- Communication with BB is via UART, internal USB or cellular
- There’s little independent monitoring and control of its embedded OS in production mode -- can be hard to trigger, detect, and analyze crashes
  - Similar to exploiting bootrom in DFU mode, when direct feedback is limited or delayed
- However, as the BB is crashing, it saves a limited crash report into its NVRAM which can be retrieved after the subsequent reboot
3G/3GS BB crash log

System Stack:
0x406AE300
0x00000008
0x40245C90
0x40322284
0x40442F00
. . .
. . .
. . .
0x4032180C
0x2014E055

Date: 18.06.2011
Time: 06:49
Register:
 r0: 0x00000000  r1: 0x00000000  r2: 0xFFFF2318
 r3: 0x00000001  r4: 0x34343434  r5: 0x35353535
 r6: 0x35353535  r7: 0x50505050  r8: 0x00000000
 r9: 0x00000000  r10: 0x406AD320  r11: 0x406B3320
r12: 0xFFFFFFFF  r13: 0x406AE318  r14: 0x201C0A75
r15: 0x50505050
SPSR: 0x40000013  DFAR: 0xFFFFFFFF  DFSR: 0x00000005
iPhone 4 BB crash log

Trap Class: 0xBBBB (HW PREFETCH ABORT TRAP)
Date: 27.06.2010
Time: 21:21:09
Magic: 55809
Task name: atc:1
System Stack:
0x00000000
0x00000000
0x00000000
0x009D0A8
0x00000002
0x00000001
.
.
.
0x00000000
0x00000000
Fault registers:
DFAR: 0x00000000 DFSR: 0x00000000
IFAR: 0x50505050 IFSR: 0x00000005
Abort Mode registers:
  r13: 0x50505054 SPSR: 0x40000053
  r15: 0x5050504C CPSR: 0x400001D7
FIQ Mode registers:
  r8: 0x90B0C9A1 r9: 0x9D0C8303 r10: 0x44309330
  r11: 0x918ABD44 r12: 0x428206C4 r13: 0x60BDDE10
  r14: 0x970583DF SPSR: 0x00000010
SVC Mode registers:
  r13: 0x72883C50 r14: 0x601DBFED SPSR: 0x20000053
IRQ Mode registers:
  r13: 0xFFFF2F20 r14: 0x601EA118 SPSR: 0x60000053
System/User Mode registers:
  r0: 0x00000000 r1: 0x00000000 r2: 0x00000000
  r3: 0x00000001 r4: 0x34343434 r5: 0x35353535
  r6: 0x35353535 r7: 0x50505050 r8: 0x00000000
  r9: 0x00000000 r10: 0x72881000 r11: 0x00000000
  r12: 0x601AF047 r13: 0xFFFF3B00 r14: 0x6CB91B48

Thursday, May 24, 2012
General BB environment

- Large portions of BB are executed from flash addresses
  - Those code segments are not modifiable while BB is running (simply by virtue of being flash, which requires erase cycles)
  - There’s no need for ASLR, or W^X checks in flash space
  - Much smaller partitions of BB flash are writeable (nvram and secpack) but that's for data, not code
- Scatter loading relocates various code+data up to RAM
  - Especially code that’s called frequently (reduces execution time due to lower latency of RAM vs flash?)
  - The relocations are to pre-determined linked addresses (not malloc’d or randomized)
• Security related routines seem to often *not* be relocated to RAM -- they stay in flash
  • Possibly kept there just by chance (usually not frequently called anyway)
• AT parser *does* remain in flash (but possibly just because it's so huge)
• Apple has occasionally pulled code or data from RAM back into flash only (example later)
Hidden changelogs

- Throughout the first dozen 3G/3GS BB updates, we were able to monitor exactly what fixes Apple was making to BB
  - They were accidentally embedding the changelogs **directly** in the baseband images
  - Apparently part of the "ClearCase" configuration step
  - Was in gzipped form at a known offset into the image
  - Was actually programmed to flash too (!)
  - The comments about where the trouble areas were helped direct where to look for bugs
# SMS00743609 Sometimes MA traces aren't transferred
element /vobs/dwddrv/XDRV/src/xdrv_driver_if.c /main/sms736266/5

# SMS00750464 FTA TC 18.1 (Temporary Reception Gap) Fails
element /vobs/dwddrv/DSP/src/fw_sgold.c /main/dwd_sgold3/aa_ifwd_sms00743767/5

# SMS00751055 Unlocking provisioned BB crashes BB
element /vobs/dwddrv/EE_DRV/src/ee.c /main/dev_eep_static_backup/9

#5697224 SMS00726764 BB / SW:port allocation table for EVT2 to be reflected by sw
(SMS00726764)
element /vobs/dwddrv/XDRV/src/xdrv_driver_if.c .../sms736266/4
element /vobs/dwddrv/XDRV/src/xdrv_req.c /main/ifwd_sms00731097/cnnbg_ice2_int/1

# SMS00745331 N82: Critical battery level notifications are not sent
element /vobs/dwddrv/CHR/src/chr_meas.c /main/dwd_mpeuplus_globe_int/ifwd_ice2/ifwd_sms00745331/2

# SMS00706345 Generate battery curves
element /vobs/dwddrv/EPP/src/eep.c /main/nbg_mpe_driver/dwd_mpeu/dwd_ec_old_spinner_structure/dwd_mpeplus/ifwd_ice2_main/ifwd_sms00706345/4
Diagnostic and cal routines

• Basebands contain lots of unused diagnostic and calibration commands
• Some of the commands include memory writes and reads of big static buffers/arrays at fixed (linked) locations
• Normally enabled only on specially provisioned phones, but in the end it comes down to a simple flag
  • If you can tamper with that one flag via an exploit, you open up the routines and vastly simplify further exploit development
• The tables for these routines used to be scatter loaded into RAM (unlike the normal AT command tables)
  • This also made it easy to commandeer the command table entries, and use them to hooks to run arbitrary injected code
  • The tables were eventually removed from the scatter list and are now back in flash, so they’re harder to commandeer
  • Most of the commands are still there including the mem writes/reads
Quick help:
Wildcard-supported by '*' operator before and/or after sub-symbol-string e.g 'my_fun*

Queries by '?' operator:
- functions starting with 'rf' : rf()?
- function description for 'my_rf_func' : my_rf_func()??
- all enum types : *?
- 'my_enum' items : $my_enum??
- 'mystruct.myvar' variable value: mystruct.myvar?
- 'mystruct' elements : mystruct.*??
- 'mystruct.myarray[3,10]' : mystruct.myarray[3,10]??

(NOTE1: number of '?' determines query level  
   higher levels generally means more info)  
(NOTE2: after '?'s optionally put output format specifier  
   e.g. 'myvar??%x' for hex output)

Write variables:
- write 0x43 to mystruct.myvar : mystruct.myvar=0x43
- write 3290 to mystruct.myarray[4]: mystruct.myarray[4]=3290
- write elements of above array : mystruct.myarray[2,5]={5,0x30,4500}

Call functions:
- call myfunc(%d,%u,%d) : myfunc(-3,0x30,true)  
  (note: 'true' is of enum type $bool

- call myfunc(%d, %&qd[9]) : myfunc(50,{4,2,3,70,100})
  (note: array function arguments need not be completely filled)

variable type specifiers examples:  
%d=int %ld=long int %u=uint %c=char %hd=half int %qu=quarter uint (~u8), %s=string  
%d[n] int array of size <n>
Diagnostic routine example

```
at@gticom:  
    OK
at@seq_kill(2)  
    OK
at@seq_init(2,0)  
    OK
at@seq_insert(2,1,"print("iPhone DevTeam countdown to 3.0:")")  
    OK
at@seq_insert(2,2,"new("%d:i",1)")  
    OK
at@seq_insert(2,3,"i=3")  
    OK
at@seq_insert(2,4,"while(i>0)"  
    OK
at@seq_insert(2,5,"print(i)"  
    OK
at@seq_insert(2,6,"i=calc(i-1)"  
    OK
at@seq_insert(2,7,"endwhile"  
    OK
at@seq_insert(2,8,"print("CAN I HAZ YELLOWSNOW??!?)")  
    OK
at@seq_run(2)  
    OK
iPhone DevTeam countdown to 3.0:  
    3
    2
    1
    CAN I HAZ YELLOWSNOW??!?
```

```
at@m(0x403c6068,16,0xe92d5ffe,0xeb00002f,0xe8bd9ffe,0x4b21b530,0x681b2080,0x0180b084,0xf854f000,0x491f4b1e,  
   0x1c05681b,0x20002211,0xf84cf000,0x481d4b1c,0x4669681b,0xf00022ff,0x2300f845,0x9b00702b)
```
AT commands

• The 3G/3GS basebands still contain several vulnerable AT commands
• But Apple started to mask off unused commands (rather than audit or remove them)
• Unlike the diagnostic commands, these disabled commands aren't designed to be dynamically enabled
  • The bitmask is created once at BB startup and is never updated again
AT command disable bits

01.59.00 command disables =
11111111111111100100000001110100000000011011101100
0100111101011001101111000111110000111000000000000000001000
00001111111111111111010111111100101000000000100010010011111111111011
01111111111011101100111111111101111101101111111111111111101111111
01111111110101101011111011111011001010101101111100011011111010011
11001111111001010101011110101011001000110111010011111111111111111111
00001010101011001111011101011000010

02.10.04 command disables =
1111111111111110001011111111001000000000111010100000000011011101100
0100111101011001101111000111110000111000000000000000001000
00001111111111111111010111111100101000000000100010010011111111111011
01111111111011101100111111111101111101101111111111111111101111111
011111111110101101011111011111011001010101101111100011011111010011
11001111111001010101011110101011001000110111010011111111111111111111
00001110101010110001111011101011000010
iPhone2G SW unlock

- Bootrom invokes bootloader which then sigchecks baseband
- Bootloader was either version 3.9 or 4.6 depending on manufacture date
  - 3.9 vulnerable to Bleichenbacher RSA forged signature
    - secpacks vulnerable: could write arbitrary carrier lock tables ("iPhoneSimFree" -- commercial unlock)
    - main BB FW also vulnerable: could flash arbitrary BB, ignoring carrier lock tables completely ("AnySIM" from iPhone Dev Team, free)
  - 4.6 vulnerable to firmware update trick that could erase bootloader
    - could then flash stock Apple BL 3.9 images and use the 3.9 exploits
- Eventually: “BootNeuter” app (iPhone Dev Team, free)
  - flashed a BB modified to remove NOR "locked" attributes of BL pages and erase/reflash them directly
  - included a "Fakeblank" option for running custom code injected at BB boot time over serial port (because bootloader appeared "missing")
About 70 tasks run in the 3G/3GS BB, across a few dozen priorities levels. Most tasks don’t directly call each other.

- They pass short messages to each other via mailboxes, or longer via queues
- The messages involved with the carrier check are between the “sim” and “sec” tasks
- By watching the mailbox semaphore owners, we can chart the general activation/unlock operations
- ultrasnow 3G/3GS tampered with "compare_lock_data" message
  - sec code segment is in flash so can't directly patch it with an exploit
  - ultrasnow inserts a new task at a priority 0x44, one level higher than “sec”
  - We see the messages from the sim task before sec can
3G/3GS BB tasks

at
OK
at@devteam()
devteam 3gbb tool v1.1
70 tasks [with priorities]:
drv_cb__[3C]  gct[78]  soc1[78]  llu:1[05]
  umacul:1[0A]  umacd1:1[0B]  umacc:1[0C]  urlcul:1[14]
  mon:78  ata:54  ipr_rx1:54  ipr_rx2:54
  ipr_rx3:54  mux:3C  io_evt:3C  atcptest:45
sec:45  xdrv_dat:96  EE_task:FE  gate_rtr:FF
DMA:FF  sme:37
sim -> sec activation messages

```
sim:1 sent sec 0xb  msg from  get_lock_profiles
sim:1 sent sec 0xc  msg from  get_file_profile
sim:1 sent sec 0xc  msg from  get_file_profile
sim:1 sent sec 0xc  msg from  get_file_profile
sim:1 sent sec 0xc  msg from  get_file_profile
sim:1 sent sec 0xd  msg from  compare_lock_data
sim:1 sent sec 0x1  msg from  get_bcd_imei
sim:1 sent sec 0x13 msg from  get_tmsi
sim:1 sent sec 0x13 msg from  get_tmsi
sim:1 sent sec 0x13 msg from  get_tmsi
sim:1 sent sec 0x13 msg from  get_tmsi
```
void inject() {
    status = nu_TCCE_Create_Task((TC_TCB *)system_malloc(sizeof(TC_TCB))
        "devteam1"        /*task name*/,
        (void*)0x4042d9a0 /*fixed address of devteam1() below*/,
        0, 0,
        system_malloc(UNLOCK_STACK_SIZE), UNLOCK_STACK_SIZE,
        0x44 /*priority*/,
        0 /* time slice */, NU_PREEMPT, NU_START);
}

void devteam1() {
    MB_MCB *mbox = (MB_MCB *)SEC_MAILBOX;  // the mailbox structures are at fixed locations
    while (1) {
        // intercept any mailbox messages intended for SEC
        // (we were installed above at priority 0x44, SEC is at lower priority 0x45)
        nu_MBCE_Receive_From_Mailbox((void*)mbox, msg, NU_SUSPEND);
        if (msg[0]==0xd /*ACT*/) {
            // if the message to SEC was an activation query, short circuit the query
            uint32_t *p = (uint32_t *)msg[1];
            p[3] = 1;                           // do all the stuff that
            *(uint32_t *)(SECBASE+0x14) = p[0]; // SEC would have done if it were to decide
            *(uint32_t *)(SECBASE+0x18) = p[1]; // carrier was allowed by the lock tables
            uint32_t *pp = (uint32_t *)p[2];
            pp[0] = 0x0100ff00; pp[1] = 0x04020401; pp[2] = 0x04040403;
            msg[0]=0x20;  // change func_id from 0xd to 0x20
        }
        // deliver message whether it was tampered above or not
        nu_MBCE_Send_To_Mailbox((void*)mbox, msg, NU_SUSPEND);
    }
}
Similar message tampering technique was used in iPhone4 01.59.00 ultrasnow

Apple started looking for this message tampering (although they have typos all throughout their debug strings, calling it "tambering")

A much more challenging obstacle on the iPhone4 was the hardware-based DEP mechanism ("crossbar").
- As soon as you write to memory, hardware disables all execution rights for the address range containing it
- The solution @planetbeing and I developed for ultrasnow to overcome the crossbar is detailed in the iOS Hacker's Handbook
iPhone4 “tamber” check

SEC_compare_lock_data+1A	 MOV R2, SP
SEC_compare_lock_data+1C	 MOVSR0, #0xD
SEC_compare_lock_data+1E	 MOV R1, SP
SEC_compare_lock_data+20	 STMIA R2!, {arg0-arg2}
SEC_compare_lock_data+22	 BL send_msg_to_SEC_task
SEC_compare_lock_data+22
SEC_compare_lock_data+26	 CMP R0, #0xD
SEC_compare_lock_data+28	 BEQ ok3
SEC_compare_lock_data+28
SEC_compare_lock_data+2A	 MOVSR2, #0
SEC_compare_lock_data+2C	 MOVSR0, #2
SEC_compare_lock_data+2E	 ADR R1, aErrorFunc_idHasBe_3

"Error: func_id has been tambered"
SEC_compare_lock_data+30	 BL msg
SEC_compare_lock_data+30
SEC_compare_lock_data+34
SEC_compare_lock_data+34	 ok3
SEC_compare_lock_data+34	 MOV R3, SP
SIM interposer unlocks

- Commercial SIM interposer unlocks take advantage of timing or protocol quirks of the baseband, rather than trying to trigger a traditional exploit and custom code execution.
- They physically sit between SIM and SIM reader, so they can alter, delay or block communication between the SIM and BB.
- Early example of SIM interposer was “Turbosim”
  - BB quirk: when a SIM was inserted, BB would read the IMSI 3 separate times.
  - The first 2 times were solely for comparing that SIM’s IMSI against the carrier lock tables.
  - Turbosim would fake the IMSI sent those first two times, substituting in the MCC and MNC of the official carrier.
  - It would then send the real IMSI for the SIM when the BB needed it to actually access the carrier network.
SIM interposer unlocks

- SIMs don't have access to the same AT parser that the BB exposes to CommCenter (and ultrasnow)
- SIMs do have access to the BB’s SIM Toolkit interface
  - JerrySIM was an iPhone Dev Team unlock that exploited this SIM/STK interface
  - Apple fixed the STK bug before we could deploy it (we saw it mentioned in the hidden changelogs!)
- For an example of a network-side hack that exploits the baseband from further away than the SIM tray, see @esizkur’s remote listener example in the iOS Hacker's Handbook
Changelog_02.04.03.txt:

# SMS00788402/SMS00787413 (CL->MSAP)
satfuzz / "jerrysim" STK attack still crashes ICE2 (SMS00787413)

Changelog_02.04.03.txt:

# SMS00788406/SMS00780636: satfuzz / "jerrysim" STK attack still crashes ICE2 (SMS00780636)
iPhone 4 carrier activation

- Non-Apple baseband typically get unlocked via one-time "AT+CLCK"
  - Carrier gives customer unique NCK code when subsidy has been paid, etc
  - Baseband crypto verifies the NCK and sets a permanent flag
- The NCK vendor code is in iPhone BB, but it’s ignored (no permanent flag!)
- Apple instead implements "activation tickets"
  - No such thing as a permanent iPhone unlock
  - Activation ticket specifies which MCC/MNCs are valid. Signed by Apple's servers using typical public key signature techniques
  - The server populates and signs the activation ticket based on what carriers the Apple activation servers have on record for a given IMEI
  - Commcenter sends activation ticket to BB after every BB reset (it's not kept in BB flash)
  - Activation ticket is preserved in FS through an IPSW "update", but not "restore"
- On the i4, the activation ticket is TEA-encrypted using device's unique hardware thumbprint (NOR chip IDs, etc)
  - Most can't decrypt the i4 activation tickets because they don’t know these values
### iPhone 4 Activation Ticket

### Field Specifications

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
<th>Len</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>ticketVersion</td>
<td>0</td>
<td>4</td>
<td>must be 2 (always in plaintext)</td>
</tr>
<tr>
<td>certLen</td>
<td>4</td>
<td>4</td>
<td>must be 18c</td>
</tr>
<tr>
<td>certVersion</td>
<td>8</td>
<td>4</td>
<td>must be 1</td>
</tr>
<tr>
<td>pubKeyLen</td>
<td>c</td>
<td>4</td>
<td>must be 0x400</td>
</tr>
<tr>
<td>exponent</td>
<td>10</td>
<td>4</td>
<td>RSA exponent (3)</td>
</tr>
<tr>
<td>certificateKey</td>
<td>14</td>
<td>80</td>
<td>RSA modulus for ticket payload</td>
</tr>
<tr>
<td>certificateNonce</td>
<td>94</td>
<td>80</td>
<td>rest of certificate</td>
</tr>
<tr>
<td>certificateSig</td>
<td>114</td>
<td>80</td>
<td>certificate signature</td>
</tr>
<tr>
<td>ICCID</td>
<td>194</td>
<td>c</td>
<td>BCD, must match this SIM's ICCID (wildcarding allowed)</td>
</tr>
<tr>
<td>IMEI</td>
<td>1a0</td>
<td>8</td>
<td>BCD, must match this phone's IMEI (no wildcarding)</td>
</tr>
<tr>
<td>thumbprint</td>
<td>1a8</td>
<td>14</td>
<td>must match this phone's HW thumbprint</td>
</tr>
<tr>
<td>payloadSize</td>
<td>1bc</td>
<td>4</td>
<td>size of IMSI payload (will be multiple of c)</td>
</tr>
<tr>
<td>recordA</td>
<td>1c0</td>
<td>c</td>
<td>first IMSI record (wildcarding allowed)</td>
</tr>
<tr>
<td>recordB</td>
<td>1cc</td>
<td>c</td>
<td>OPTIONAL additional IMSI records (wildcarding allowed)</td>
</tr>
<tr>
<td>recordC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ticketSig</td>
<td>1cc</td>
<td>80</td>
<td>signature of ticket</td>
</tr>
</tbody>
</table>

### IMSIs Listed in Activation Ticket for i4 Locked to USA AT&T (starting at "recordA"):

```
3c 00 00 00 // size of below IMSI table
00 00 00 00 31 01 50 ee ee ee ee ef // 310 150 **********
00 00 00 00 31 01 70 ee ee ee ee ef // 310 170 **********
00 00 00 00 31 04 10 ee ee ee ee ef // 310 410 **********
00 00 00 00 31 11 80 ee ee ee ee ef // 311 180 **********
00 00 00 00 31 09 80 ee ee ee ee ef // 310 980 **********
```

- **MCC 310 = USA**
- **MNC**
  - 150 Cingular Wireless (discontinued)
  - 170 Cingular Orange
  - 180 West Central Wireless
  - 410 AT&T Mobility (standard)
  - 980 AT&T Mobility (not in commercial use)
iPhone4S carrier activation

- iPhone4S uses flow similar to iPhone4 with some minor changes
  - They don't bother to TEA-encrypt the ticket anymore
  - They encode the ticket using standard ASN.1 notation
  - Almost *everything* signed by Apple nowadays uses ASN.1, even APTickets
  - The recent **SAM unlock** took advantage of temporary **glitch in the activation servers**
    - If you requested a ticket using MCC/MNC of your iPhone model’s official carrier, the server erroneously associated your (non-official) SIM's ICCID with the official MCC/MNC
    - After the initial bogus request was made, you could then send a real ticket request using your actual MCC/MNC and ICCID. **The server would hand you back a signed ticket good for that ICCID**
    - Not quite a full unlock (because each ticket is tied to one ICCID only)
    - The issued tickets are good for 3 years, so can be manually saved and re-used
iPhone4S act ticket (locked)
iPhone4S act ticket (SAM)

d=0  hl=4  l= 411 cons: SEQUENCE

d=1  hl=2  l=  1 prim: INTEGER           :01

d=1  hl=2  l= 11 cons: SEQUENCE

d=2  hl=2  l=  9 prim: OBJECT            :sha1WithRSAEncryption

d=1  hl=2  l= 102 cons: SET

d=2  hl=3  l=  4 prim: cont [ 63 ]       BBSerNum             12345678

d=2  hl=3  l=  4 prim: cont [ 64 ]       BBChipID             e1005a00

d=2  hl=3  l= 20 prim: cont [ 75 ]       serverRandomness     19fb083b96acda80...

d=2  hl=4  l=  7 prim: cont [ 1005 ]      IMEI                  01291234567890

d=2  hl=4  l= 10 prim: cont [ 3004 ]      ICCID                 89011234567812345678

d=2  hl=4  l= 12 prim: cont [ 3005 ]      IMSI                  000000003102601234567890

d=2  hl=4  l=  4 prim: cont [ 3006 ]      00000000

d=2  hl=4  l=  4 prim: cont [ 3007 ]      01000000

d=2  hl=4  l=  4 prim: cont [ 3008 ]      00000000
3G/3GS baseband downgrades

- Until the i4, basebands could only be reflashed with newer versions
  - Unlike the main firmware, which has no version checking per-se
  - Policy enforced by the "emergency boot loader" EBL that's a normal part of Apple's BB update process
- EBL injected over serial, sig checked by bootrom
  - Executes entirely in RAM and controls the rest of the reflash, including sig checking the incoming main image and enforcing the no-downgrade rule
- The 5.8 bootloader of early iPhone3G can be exploited and tricked it into running a tampered EBL
  - "Fuzzyband" implements this exploit for iPhone3G with 5.8BL, allowing downgrades to ultrasnow-compatible basebands
  - The bug was fixed in version 5.9 of the iPhone3G bootloader
  - Cannot simply reflash the 5.8 bootloader into those newer units due to bootrom checks of the bootloader
This code is MEANT to verify the addr and size of the EBL:

```
getldr_from_uart_and_go+3F8 loc_8141C
getldr_from_uart_and_go+3F8
getldr_from_uart_and_go+3FC
getldr_from_uart_and_go+400
getldr_from_uart_and_go+404
getldr_from_uart_and_go+408
getldr_from_uart_and_go+40C
getldr_from_uart_and_go+410
getldr_from_uart_and_go+414
getldr_from_uart_and_go+418
getldr_from_uart_and_go+41C
getldr_from_uart_and_go+420
getldr_from_uart_and_go+424
getldr_from_uart_and_go+428
```

```
ADD R2, SP, #0x40+signed_size
ADD R1, SP, #0x40+signed_addr
LDR R0, =0x93D00
BLX rsa_chk_ldr_signature  //(must still be a signature)
CMP R0, #0
BNE die
```

Wrong:
```
LDR R0, [SP,#0x40+signed_addr]
CMP R0, #0x86000
BNE continue
LDR R0, [SP,#0x40+signed_size]
CMP R0, #0xDF00
BEQ die
```

To exploit this, put any valid signature there (but make sure that the signature still verifies whatever it was meant to).

For instance: use the signature for the current main FW

```
LDR R2, =0x20040C48
```

```
31
Thursday, May 24, 2012
```
iPhone3G BL 5.9

get_ldr_from_uart_and_go+3F8  loc_81F6C
get_ldr_from_uart_and_go+3F8  ADD  R2, SP, #0x40+signed_size
get_ldr_from_uart_and_go+3FC  ADD  R1, SP, #0x40+signed_addr
get_ldr_from_uart_and_go+400  LDR  R0, =0x93D00
get_ldr_from_uart_and_go+404  BLX  rsa_chk_ldr_signature
get_ldr_from_uart_and_go+408  CMP  R0, #0
get_ldr_from_uart_and_go+40C  BNE  die
get_ldr_from_uart_and_go+410  LDR  R0, [SP,#0x40+signed_addr]
get_ldr_from_uart_and_go+414  CMP  R0, #0x86000
get_ldr_from_uart_and_go+418  BNE  die
get_ldr_from_uart_and_go+41C  LDR  R0, [SP,#0x40+signed_size]
get_ldr_from_uart_and_go+420  CMP  R0, #0xDD00
get_ldr_from_uart_and_go+424  BNE  die
iPhone4 baseband downgrading

• Starting with the iPhone4, the "no downgrade" rule is no longer enforced by EBL
• Instead, the baseband reflash process is personalized for each unique iPhone with signed BBTickets
• Part of what's signed includes unique BB chip IDs for that phone, and a random nonce generated by the EBL
• After submitting all the personalized information to Apple's upgrade server, the EBL checks that the returned signed BBTicket is correct and then flashes it along with the incoming BB image
• As long as Apple is currently signing that baseband version, it will be flashed (even if it's a downgrade)
• This is useful mostly during iOS beta periods, when the app developers may need to come back down from a beta version (which often includes a different baseband)
  • By comparison, trying to downgrade 3G/3GS FW causes the iOS restore to fail due to downward BB version
• The signed i4 BBTicket is also verified on every BB boot
  • Unlike the main firmware APTickets, the BB verifies that the nonce hash in the BBTicket matches the nonce originally generated by the EBL
  • The actual nonce is kept in a secure hardware register in the BB chip, only written to by EBL
iPhone4S baseband

- iPhone4S has **no flash** to store the main BB FW or bootloader
- Enters a sort of emergency service mode every time it's reset
  - It has nothing to boot by itself -- needs main AP assistance
  - Compared to normal Qualcomm basebands, it's as if the bootrom failed to validate the 2nd-stage DBL in flash, and entered DLOAD mode (almost!)
- It won't accept arbitrary code -- must be signed
- Apple also modified the the normal Qualcomm bootrom to **require** that the very first thing sent in DLOAD mode is a BBticket
  - Apple calls this the **"Maverick"** protocol in Commcenter
  - Similar concept to the iPhone4 BBTicket, except now the BBTicket is stored over on the main AP filesystem, not in flash (remember there is no flash)
  - Restore process stores the personalized *.bbfw images and BBTicket on root filesystem (which is mounted read/write during the restore)
  - BBticket in the *.bbfw file must have nonce matching the one saved in persistent BB hardware register
Qualcomm has extensive debug commands in DIAG protocol
  - Apple disables them like the extraneous 3G/3GS disabled AT commands
- There's a bug in Apple’s Maverick protocol that allows unauthorized access to the bootrom space
- Each stage of the flash-less boot provides different angle for finding bugs
  - Maverick (bbticket.der), DBL, OSBL, AMSS
  - Can fuzz for bootloader-level bugs without lengthy (and dangerous) flashing -- it’s never been so quick and safe to do this on an iPhone baseband
- Any bugs in early boot stages likely more powerful
- Downside: AT parser is gone. Replaced by Qualcomm protocols and internal USB
- No chance to brick the BB by playing (every boot is an emergency boot!)
- iPad3’s Qualcomm baseband appears to move much of the codebase from ARM over to the QC Hexagon DSP...is the iPhone baseband next?
3G/3GS baseband downgrades

- 3G/3GS baseband can be "upgraded" to iPad1 BB version 06.15
  - Still vulnerable to the AT+XAPP exploit
    - EBL allows the upgrade, since it satisfies the "greater than" check
    - Normal 3G/3GS basebands are still down in the 05.xx range
- But 06.15 baseband has limited GPS functionality (assisted-GPS that primarily uses wifi and cellular tower location databases, not satellites)
- Now that Apple is officially unlocking many older USA 3G/3GS units, unlockers want to come back down to the normal 3G/3GS baseband
  - EBL won't allow this, but we still can run custom code within main baseband via the ultrasnow exploit
  - Compared to the EBL runtime environment, trickier to reflash from a running baseband because you can’t erase while you’re using that NAND partition
    - The baseband itself is partially executing from the flash
    - Need to do some kind of controlled shutdown of Nucleus (which isn't designed for that)
3G/3GS baseband downgrades

- 3GS phones are *still* being sold
  - Until a recent update by Apple to newer NOR+RAM chips, the 06.15 trick still worked
  - But the **06.15 BB doesn't recognize the newer RAM** and so it hangs during init, bricking the radio
    - EBL doesn't recognize this compatibility issue and so it happily updated/bricked to the 06.15 image the unlocker gave it
    - EBL itself can still be injected in this bricked state, but it will refuse to downgrade (as usual)
  - Some commercial unlock sellers retrofit new 3GS phones with the older BB+NOR+RAM boards (and then apply the 06.15 upgrade and ultrasnow unlock)
Baseband brickability

- iPhone2G
  - Brickable if the BL image flashed to NOR crashed due to bad code
  - recoverable via original A17 hardware hack (makes BL look empty)
- iPhone3G and iPhone3GS
  - Brickable if only one of the two bootloader page is empty (normal BL spans two NOR pages)
    - In this case, EBL is never given a chance to run
    - Looks like unintended side effect (unanticipated condition?)
- iPhone4
  - Not brickable even with a partially erased or tampered bootloader
  - Will just wait for an EBL image to be uploaded to fix it
- iPhone4S
  - Not brickable (no persistent bootloader at all!)
Questions?

Thanks!