Reverse-Engineering eines Embedded-Linux-Systems am Beispiel TomTom Go

„Embedded systems“ überall

OpenTom

Router

Navigation

Sat-Empfänger

Telefone

Spielekonsolen

MP3-Player

Mehrwert „offener“ Systeme

OpenTom

- Navigation
- MP3-Player
- Bluetooth Handsfree Kit
- mobile Kopierstation
- Fahrtenbuch
- ...

Betriebssystem

Komplexe „embedded systems“ benötigen ein Betriebs-system – meist kommen Standardlösungen zum Einsatz.

- Windows CE
- Linux
- VxWorks
- QNX
- ...

Für Systeme unter Linux sollte reverse engineering eigentlich nicht nötig sein.

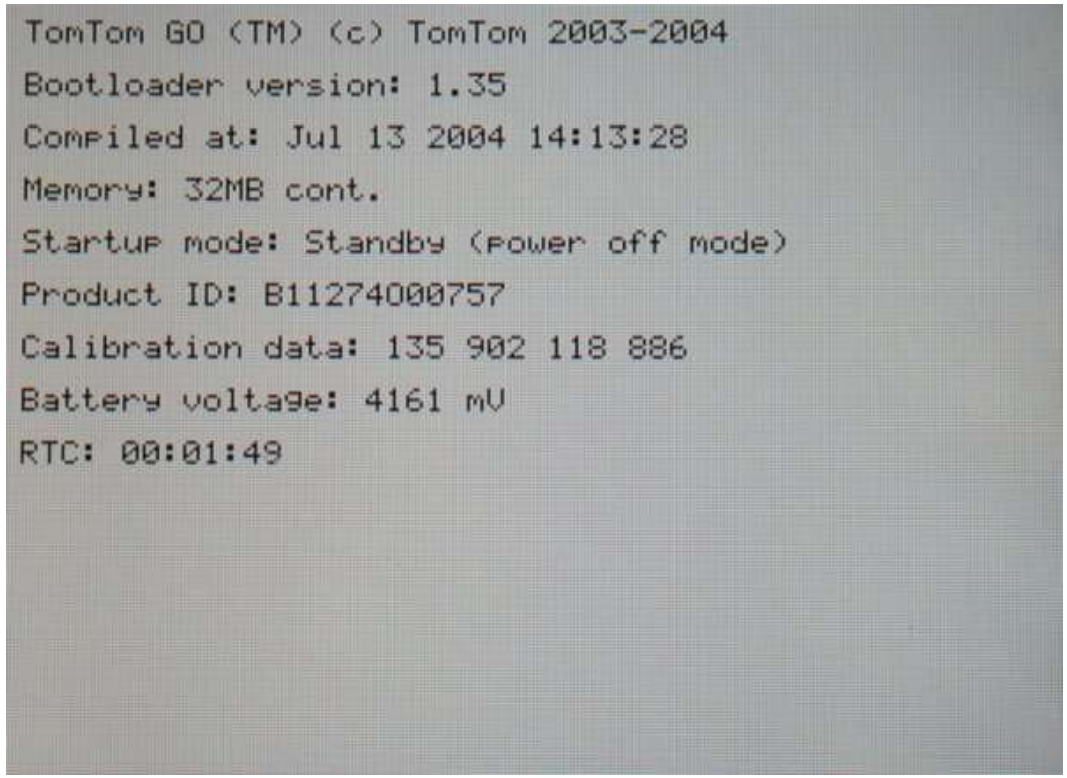
Die GPL verpflichtet den Hersteller zur Veröffentlichung aller zur Entwicklung eigener Funktionalität notwendigen Dinge.

- Quellen für Kernel & GNU-Tools
- Toolchain
- Systemsspezifische Build-Tools

Oft ignoriert oder übergeht der Hersteller die GPL. Dann ist „reverse engineering“ notwendig um dies nachzuweisen und das System zu „übernehmen“.

Durch einfaches „Herumspielen“ können dem System oft wertvolle Informationen abgerungen werden.

- Bootloader
- Debug-Tools
- Fehlermeldungen
- Abstürze



Analyse des Dateisystems

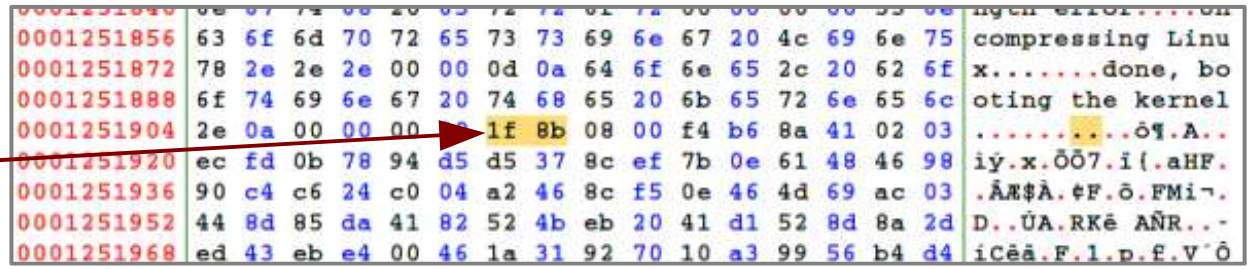
```
     33 Jan  1  2004 CurrentMap.dat
   2048 Oct 13 22:36 Deutschland-Map
   2048 Oct 13 22:36 TomTom-Cfg
 773676 Sep  3 16:04 data.chk
 794756 Mar 24  2004 data01.chk
 826460 Mar 24  2004 data02.chk
 683093 Mar 24  2004 data05.chk
 670562 Mar 24  2004 data12.chk
 735281 Apr 12  2004 data13.chk
 712656 Apr 11  2004 data15.chk
     28 Mar 23  2004 deuf.vif
     30 Mar 23  2004 deum.vif
     33 Mar 23  2004 engf.vif
     34 Mar 23  2004 fraf.vif
     37 Jun 10 12:33 install.bif
     30 Mar 23  2004 ita.vif
     35 Mar 23  2004 nldf.vif
 267536 Sep  3 16:04 system
      4 Jun  7 18:08 tomtom.aid
    220 Jan  1  2004 ttgo.bif
1908942 Sep  3 16:04 ttsystem
```

```
# strings ttsystem
[...]
ran out of input data
Malloc error
Memory error
Out of memory
incomplete literal tree
incomplete distance tree
bad gzip magic numbers
internal error, invalid method
Input is encrypted
Multi part input
Input has invalid flags
invalid compressed format (err=1)
invalid compressed format (err=2)
out of memory
invalid compressed format (other)
crc error
length error
Uncompressing Linux...
done, booting the kernel.
[...]
```

Extraktion des Kernels

OpenTom

zImage:



```
0001251856 63 6f 6d 70 72 65 73 73 69 6e 67 20 4c 69 6e 75 compressing Linu
0001251872 78 2e 2e 2e 00 00 0d 0a 64 6f 6e 65 2c 20 62 6f x.......done, bo
0001251888 6f 74 69 6e 67 20 74 68 65 20 6b 65 72 6e 65 6c oting the kernel
0001251904 2e 0a 00 00 00 1f 8b 08 00 f4 b6 8a 41 02 03 ..............A..
0001251920 ec fd 0b 78 94 d5 d5 37 8c ef 7b 0e 61 48 46 98 iý.x.ÕÕ7.ï{.aHF.
0001251936 90 c4 c6 24 c0 04 a2 46 8c f5 0e 46 4d 69 ac 03 .ÄÆ$À.¢F.õ.FMi¬.
0001251952 44 8d 85 da 41 82 52 4b eb 20 41 d1 52 8d 8a 2d D..ÚA.RKë AÑR..-
0001251968 ed 43 eb e4 00 46 1a 31 92 70 10 a3 99 56 b4 d4 íCëä.F.1.p.£.V´Ô
```

Alles ab den „magic bytes“ wird in eine neue Datei kopiert und mit gunzip entpackt.

Damit steht der „reine“ Kernel zur weiteren Analyse zur Verfügung.

Analyse des Kernels

Analyse mit „strings“ ergibt (auszugsweise):

```
root=/dev/ram0 console=ttyS0
Linux version 2.4.18-rmk6-swl5-tt201 (aya@achilles.intra.local)
<3>initrd (0x%08lx - 0x%08lx) extends beyond physical memory - disabling initrd
EXT2-fs: unable to read group descriptors
FAT: logical sector size too small for device (logical sector size = %d)
u.v.m.
```

Konsole auf seriellem Port

Init-Ramdisk

Linux 2.4

FAT für SD-Karten

Ext2 für Init-Ramdisk(?)

Beweisführung abgeschlossen

- Die Benutzung von GPL-Code (Linux Kernel) ist hiermit hinreichend nachgewiesen.
- Der Hersteller kann nachdrücklich auf den GPL-Verstoß hingwiesen werden.
- Anhand der Texte im Kernel kann die Verwendung einzelner Features nachgewiesen werden.
- Notfalls koennen Rechteinhaber dieser Features (z.B. Initrd) den Hersteller abmahnen.
- Führt dies nicht oder erst nach längerer Zeit zum Erfolg, muss das System weiter untersucht werden...

Extraktion der Init-Ramdisk

OpenTom

Die Init-Ramdisk wird vom Bootloader in den Speicher geladen – die Startaddresse wird dem Kernel übergeben.

Der Kernel sucht nach Filesystem- oder gzip-Magics.

ttsystem:

```
0000000000  54 54 42 4c 8e e5 12 00 00 00 00 31 1f 8b 08 00  TTBL.å.....1....
0000000016  29 78 65 41 02 03 ec 9d 09 78 14 45 de c6 6b 66  )xeA..i..x.EÞÆkf
0000000032  42 12 20 42 44 54 3e d1 35 5e c8 19 50 3c 40 50  B. BDT>Ñ5^È.P<@P
0000000048  12 0e 01 4f 14 fd d4 55 d7 1c 33 24 91 24 13 33  ...O.ýÔU×.3$.$.3
```

Offensichtlich befindet sich am anfach der ttsystem ein weiterer gzip-komprimierter Block – dieser enthält ein ext2-Dateisystem, die Init-Ramdisk.

Analyse der Init-Ramdisk

- Die Init-Ramdisk enthält ein Mini-Linux-System, basierend auf BusyBox.
- Auch die Navigations-Software (/bin/ttn) ist in der Init-Ramdisk enthalten.
- Das Init-Script (/etc/rc) sieht einen Debug-Modus mit Shell vor!

```
echo "* Starting ${product}"
if test -f ${debugf}
then
  ${ttnapp} > /dev/console 2>&1 &
  echo "* Starting shell"
  sh
else
  ${ttnapp} > /dev/null 2>&1 &
fi
```

- Die Konsole wurde im Kernel auf ttyS0 gesetzt.
- ttyS0 ist also vermutlich zugänglich.

Die TomTom-Hardware

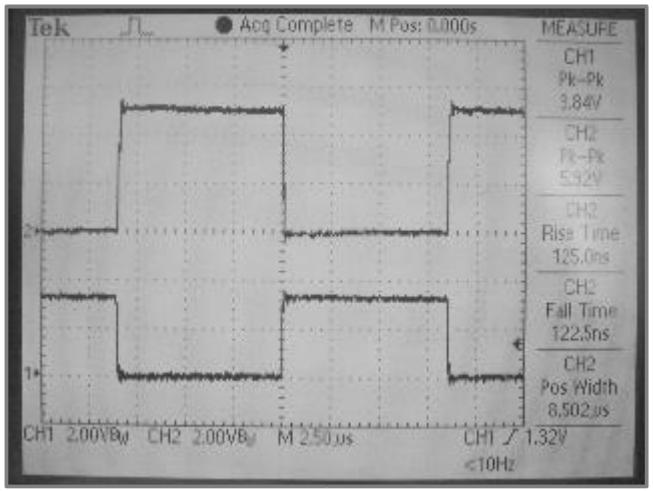
- 5V Stromversorgung
- USB
  (Host/Device umschaltbar)

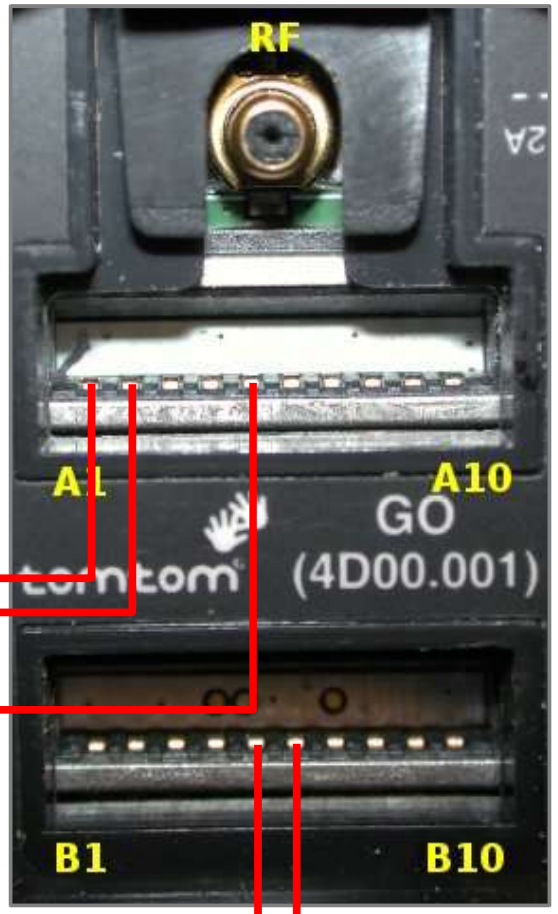
- Externe GPS-Antenne
- 2x10 Pins für Docking-Station & Carkit

Der serielle Port

OpenTom

```
Uncompressing
Linux.............................
.... done, booting the kernel.
Linux version 2.4.18-rmk6-swl5-
tt200 (aya@achilles.intra.local)
(gcc version 2.95.2 19991024
(release)) #1 Fri Sep 3 15:03:17
CEST 2004
CPU: SAMSUNG S3C2410(Arm920T)sid
(wb) revision 0
Machine: SAMSUNG ELECTRONICS Co.,
Ltd
On node 0 totalpages: 8192
zone(0): 8192 pages.
zone(1): 0 pages.
zone(2): 0 pages.
TomTom Go Boot, built at 2004-09-
03 15:04:15
* Mounting /proc
* Waiting for SD card to mount
```





Pegelkonverter
(3V <-> RS232)

TxD (A1)

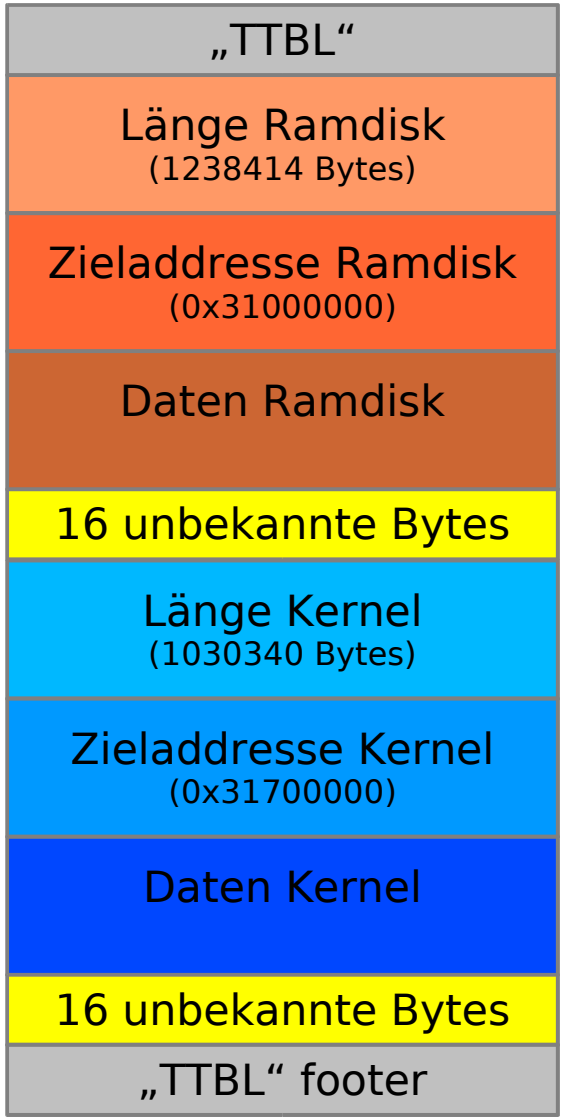
RxD (A2)

Ground (A5)

Left audio out (B5)

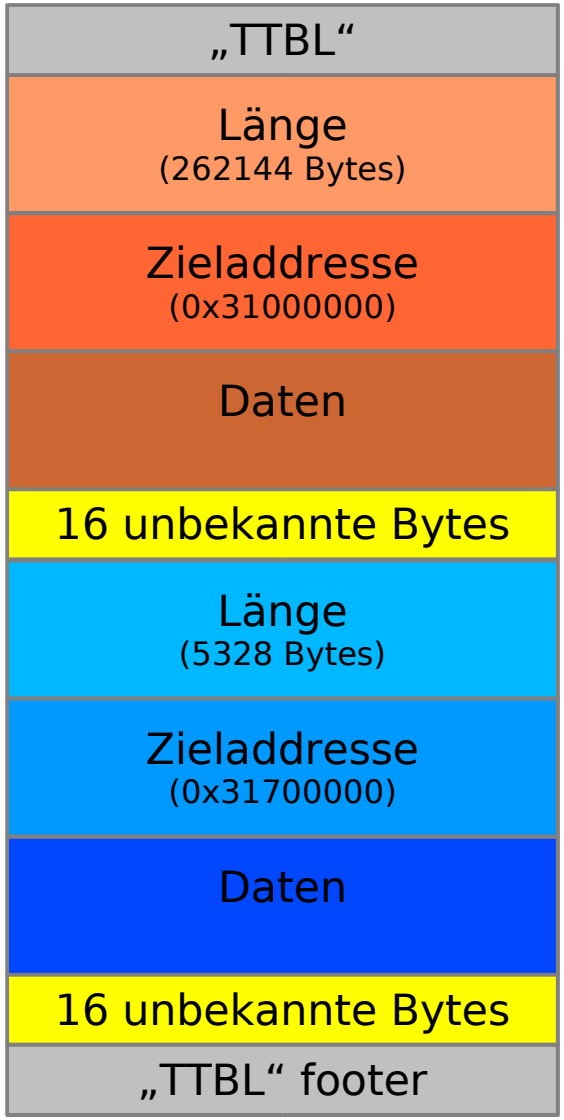
Right audio out (B6)

Der TomTom Bootloader



- Veränderung der Init-Ramdisk
- Erstellung eines „neuen“ TomTom-Images
- Anpassung des Längen-Parameters
- Eigenes Image wird vom Bootloader nicht akzeptiert
- Was haben die mysteriösen 16 Byte nach Kernel und Init-Ramdisk zu bedeuten???
- Trial-and-Error schlägt fehl
  (Kein MD4, MD5 oder ähnliches)

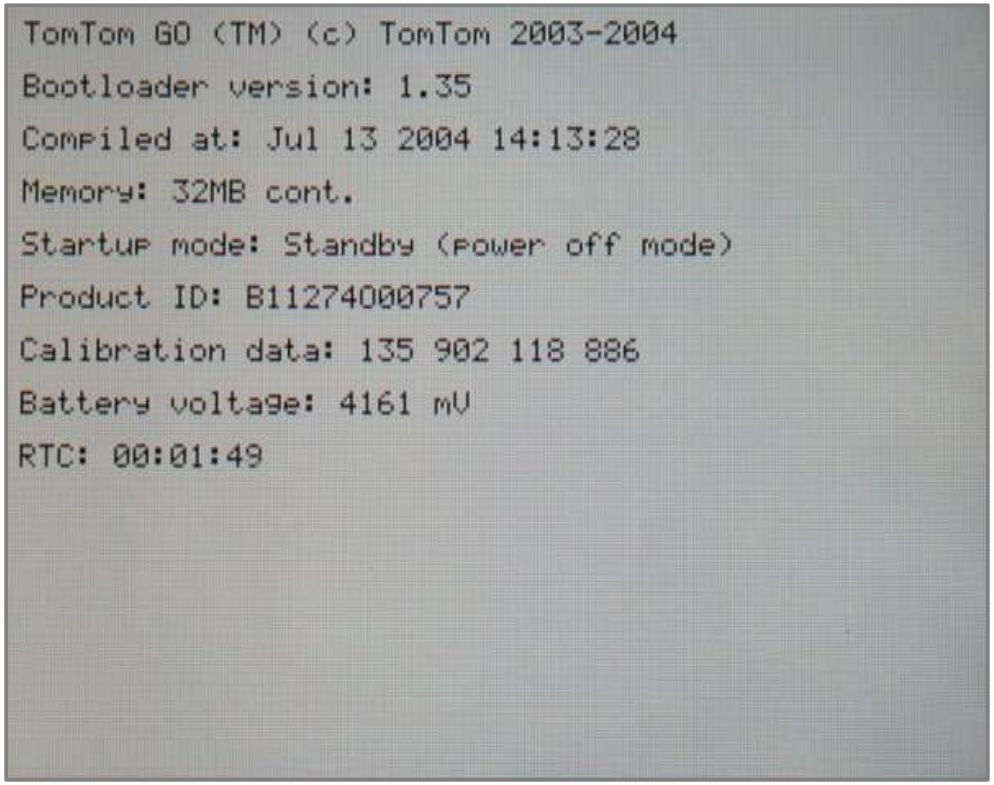
Analyse „system“



- Anstatt Init-Ramdisk sind exakt 256k Daten enthalten.
- Anstatt Kernel ist ein kleines Programm (ca. 5kB Code) enthalten
- Offensichtlich handelt es sich um eine Art Bootloader-Update.
- Weitere Nachforschungen ergeben:

Analyse Bootloader


```
# strings bootloader
[...]
TTBL
32MB not cont.
64MB cont.
32MB cont.
TomTom GO (TM) (c) TomTom 2003-2004
Bootloader version:
Compiled at: Jul 13 2004 14:13:28
Memory:
Startup mode:
Standby (power off mode)
Power down (reset)
Reboot (watchdog)
Product ID:
Calibration data:
not calibrated
Battery voltage:
RTC:
[...]
```



Die Datei „system“ enthält also tatsächlich den Bootloader-Code !

Analyse Bootloader II

Um den Verdacht auf MD5 zu prüfen, suchen wir nach typischen Konstanten.

md5.c:

```
MD5STEP (F1, a, b, c, d, in[0] + 0xd76aa478, 7);
MD5STEP (F1, d, a, b, c, in[1] + 0xe8c7b756, 12);
MD5STEP (F1, c, d, a, b, in[2] + 0x242070db, 17);
MD5STEP (F1, b, c, d, a, in[3] + 0xc1bdceee, 22);
MD5STEP (F1, a, b, c, d, in[4] + 0xf57c0faf, 7);
MD5STEP (F1, d, a, b, c, in[5] + 0x4787c62a, 12);
MD5STEP (F1, c, d, a, b, in[6] + 0xa8304613, 17);
MD5STEP (F1, b, c, d, a, in[7] + 0xfd469501, 22);
MD5STEP (F1, a, b, c, d, in[8] + 0x698098d8, 7);
MD5STEP (F1, d, a, b, c, in[9] + 0x8b44f7af, 12);
MD5STEP (F1, c, d, a, b, in[10] + 0xffff5bb1, 17);
MD5STEP (F1, b, c, d, a, in[11] + 0x895cd7be, 22);
MD5STEP (F1, a, b, c, d, in[12] + 0x6b901122, 7);
MD5STEP (F1, d, a, b, c, in[13] + 0xfd987193, 12);
MD5ST
MD5ST
```

```
0000047328  1e ff 2f e1 78 a4 6a d7 56 b7 c7 e8 db 70 20 24  .ÿ/áx¤j×V·ÇèÛp $
0000047344  ee ce bd c1 51 f0 83 0a 2a c6 87 47 13 46 30 a8  îÎ½ÁQð..*Æ.G.F0¨
0000047360  01 95 46 fd d8 98 80 69 51 08 bb 74 42 28 a3 76  ..FýØ..iQ.»tB(£v
0000047376  22 11 90 6b 6d 8e 67 02 72 bc 86 59 21 08 b4 49  "..km.g.r¼.Y!.´I
```

- Der Bootloader enthält offensichtlich eine MD5-Implementation.
- MD5 alleine ist aber nicht ausreichend.

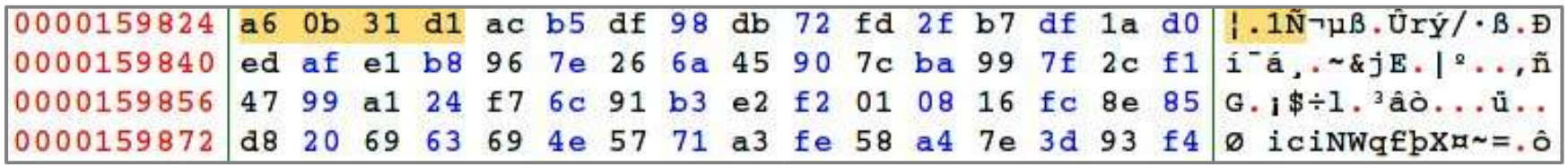
Wir suchen also nach Konstanten für Kryptografie...

...und werden bei Blowfish fündig!

blowfish.c:

```
/* precomputed S boxes */
static const u32 ks0[256] = {

0xD1310BA6,0x98DFB5AC,0x2FFD72DB,0xD01ADFB7,0xB8E1AFED,0x6A267E96,
0xBA7C9045,0xF12C7F99,0x24A19947,0xB3916CF7,0x0801F2E2,0x858EFC16,
0x636920D8,0x71574E69,0xA458FEA3,0xF4933D7E,0x0D95748F,0x728EB658,
0x718BCD58,0x82154AEE,0x7B54A41D,0xC25A59B5,0x9C30D539,0x2AF26013,
[...]
```



0000159824	a6	0b	31	d1	ac	b5	df	98	db	72	fd	2f	b7	df	1a	d0	¦.1Ñ¬µß.Ûrý/·ß.Ð
0000159840	ed	af	e1	b8	96	7e	26	6a	45	90	7c	ba	99	7f	2c	f1	í¯á¸.~&jE.|º..,ñ
0000159856	47	99	a1	24	f7	6c	91	b3	e2	f2	01	08	16	fc	8e	85	G.¡$÷l.³âò...ü..
0000159872	d8	20	69	63	69	4e	57	71	a3	fe	58	a4	7e	3d	93	f4	Ø iciNWq£þX¤~=.ô

Analyse Bootloader IV

Die weitergehende Analyse des Bootloaders mittels Disassembler führt zum Blowfish-Key:

```
D8 88 d3 13 ed 83 ba ad 9c f4 1b 50 b3 43 fa dd
```

Mit diesem Schlüssel signierte Images werden vom TomTom-Bootloader akzeptiert.

Die „richtige“ Arbeit kann jetzt beginnen!

Fertig!

Vielen Dank!

Weitere Informationen über das Projekt „OpenTom“ unter http://wiki.opentom.org