WHITE PAPER

OpenWRT on the Belkin F5D7230-4

Understanding the Belkin extended firmware for OpenWRT development

CONTROL PAGE

Document Approvals

Approved for Publication:

Author Name: Ian Latter
7 November 2004

Document Control

Document Name: OpenWRT on the Belkin F5D7230-4; Understanding the Belkin extended firmware for OpenWRT development
Document ID: openwrt on the belkin f5d7230-4.doc-Release-0.4(31)
Distribution: Unrestricted Distribution
Status: Release
Disk File: C:\papers\OpenWRT on the Belkin F5D7230-4.doc

Version	Date	Release Information	Author/s
0.4	*07-Nov-04*	Release / Unrestricted Distribution	Ian Latter
0.3	*26-Oct-04*	Release / Unrestricted Distribution	Ian Latter
0.2	*24-Oct-04*	Release / Unrestricted Distribution	Ian Latter

Distribution

Version	Release to
0.4	MidnightCode.org (correction of one grammatical error)
0.3	MidnightCode.org (correction of two minor errors)
0.2	MidnightCode.org

Table of Contents

1 Overview

1.1 In brief

In this paper the Belkin F5D7230-4 is explored for its availability as a fully integrated wireless firewall router and VPN end-point.

This work must be done in order to achieve a best-practice security solution in the Small Office / Home Office (SOHO) price-point. Where, while almost all of the casual risks are equivalent to those experienced by large enterprise, no mitigation technologies are available at an appropriate cost.

By collating a mass of publicly available information, the paper concludes by providing a root shell into the device, and a number of Linux-based reports on the hardware make-up of the router. It is hoped that this information can be used to adapt the OpenWRT embedded Linux distribution, for this Belkin router.

1.2 History

This paper represents one week's work (and a second week's documentation), in order to assess the Belkin F5D7230-4 hardware within the legally required 7-day refund period. The author has kept his router, in the hope of completing the work, and achieving the desired solution with an open source Linux configuration.

2 The problem and the solution

2.1 Buy Belkin

The Belkin F5D7230-4 Wireless (broadband) Router is a Broadcom based Linksys WRT54G clone. As such, it has 2Mbyte Flash (disk) and 16Mbyte RAM.

This handy little router has been found on sale in Australia at some very reasonable rates (as low as AU$139 at Office Works in October 2004) making it a viable solution for secure home networking, and at a reasonable speed.

2.2 The functional limitation of limited functionality

Unfortunately, like most routers targeted at the home market segment, this device - with its native firmware - is unable to be configured as;

- A NAT-less (routing only) device, while supporting packet filtering
- An IPSEC 3DES/AES tunnel termination point for WiFi and/or LAN clients

Such features are atypical requirements and would be foreign to most home users – but end users will hardly achieve best practice when the functionality does not exist to begin with.

These short-comings are not significant, however, as anyone who has embedded or even just deployed Linux will tell you; these “corporate” or “enterprise” features are not strictly hardware dependant.

Put simply, obtaining these features and functionality is as simple as replacing the Belkin firmware with a compact, yet, feature-rich Linux distribution

2.3 OpenWRT

The answer would seem to come from the OpenWRT project. OpenWRT is an open source project, targeted at deploying an embedded Linux distribution in these Broadcom-based home wireless routers.

From the project site itself;

```
OpenWRT is a Linux distribution for the Linksys WRT54G. Instead of trying to
cram every possible feature into one firmware, OpenWRT provides only a
minimal firmware with support for add-on packages. For users this means the
ability to custom tune features, removing unwanted packages to make room for
other packages and for developers this means being able to focus on packages
without having to test and release an entire firmware.
```

This is not unlike the native Belkin firmware – a Linux distribution itself, it comes with a modern kernel, and its application software is retrieved from a small CRAMFS file system stored on the 2Mbyte Flash.

However, OpenWRT goes further. Its modular construction facilitates a best-practice approach to router security – ensuring that only the required features are installed, enabled and configured;

```
The OpenWRT firmware contains two file systems, a small read-only squashfs
partition and a larger writable jffs2 partition. The squashfs file system is
the core of OpenWRT - it provides a minimal Linux environment suitable for
booting the router and providing basic functionality.

Specifically, the core provides:
   • Network initalization (ethernet and wireless)
   • Firewalling
   • DHCP client / server
   • Caching DNS server (with hooks to DHCP to lookup DHCP client
     hostnames)
   • Telnet server and busybox environment

That's it. Everything else (SSH, HTTP administration, etc) can be done in
the form of a package on the jffs2 file system; OpenWRT's goal is to provide
a minimal base which can be expanded through the use of software packages.
```

In this regard, a huge number of packages already exist for OpenWRT, and can be found at the project web site – including a couple that will be required for the feature extensions set out in the section above – i.e.;

- Openswan

  This package allows you to use your WRT54G as an IPSEC endpoint. You can get more information about how to set this up here: Using a Linksys WRT54G as IPSEC endpoint.

- ab0oo

  Packages related to commercial use of the OpenWRT firmware, including a full SSL lib, olsr meshing daemon, netsnmp, tc (traffic control), OpenVPN, etc.

- ramereth

  This is where you get dropbear, a ssh daemon. For files for creating the packages, see http://www.ramereth.net/openwrt/src

- sam

  Web interface and php related

- Yani

  UPnP (linux-igd), Shorewall (firewall) and various packages.

The OpenWRT project, when compiled, yields three binary firmware files, the third should be appropriate for "non-Linksys WRT54G" routers that are "based on the Broadcom chipset".

3 Disappointment

3.1 What went wrong

OpenWRT images do not readily work on the Belkin router. This is due to a number of differences between the Belkin wireless router and the Linksys WRT54G.

3.2 Recovery – reliably flashing firmware

The Belkin wireless router comes with the NVRAM variable *boot_wait* enabled (in the hardware versions up to those available at the time of writing – including those with the F5D7230-4-au_V4.00.03.bin firmware). The difference, however, is that the actual wait period is around 1 second, not the 3 seconds that is more widely published, based on the Linksys WRT54G.

A well timed execution of a slightly modified version of the published TFTP firmware upload technique, will still achieve the desired result. As documented;

1. Shutdown the router (unplug it from the power supply).
2. Enter the following at a Linux shell prompt;

```
tftp 192.168.2.1
tftp> binary
tftp> rexmt 1
tftp> trace
Packet tracing on.
tftp> put firmware.bin
```

3. Boot the router (plug it in to the power supply after it has been powered off for at least three (3) seconds).

This procedure will ensure that you can upload OEM or open source firmware images, and recover your router in times of great need.

3.3 An Extended Firmware

The Belkin Extended Firmware adds a “LOAD” (header) and an “NVAR” (footer) to the previously understood type “HDR0” firmware data block. The header adds very little value to the firmware image, but the footer allows for NVRAM variables to be applied along with the new kernel and file system image.

Without going into detail here, the HDR0 data block is fundamentally;

- A firmware (HDR0) header
- A Linux kernel
- A Linux CRAMFS root file system
- Null byte padding to reach a round numbered file size

While the OpenWRT project does output three binary firmware images – the third being *openwrt-linux.trx* that is constructed from the program “trx” – this firmware

image is a single HDR0 type data block and, hence, not a complete extended firmware that will be accepted by the Belkin.

4 New data structures

4.1 Belkin Extended Firmware Overview

As mentioned in the differences above, the Belkin Extended Firmware adds a "LOAD" (header) and "NVAR" (footer) to the previously understood "trx" type "HDR0" data block.

The "trx" program that comes with OpenWRT already produces the HDR0 data block correctly. Later in this paper, the program "belky" will be documented to demonstrate the valid construction of the extended firmware, from a HDR0 data block, and a user.conf file, exported from the Belkin HTTP user interface.

This table shows the overall relationship between the three data blocks. The two new data blocks are detailed further below;

Offset*	Length*	Block Magic	Description
0x0	0x1B	LOAD	Generic header that provides a file size and a CRC for the whole file
0x1C	(0x1B + Kernel Len + Padding + Root FS Len) Dynamic	HDR0	Standard HDR0 data block that contains a kernel, roofs and padding
Dynamic	(0x1B + NVAR Len) Dynamic	NVAR	NVRAM data (variables and values) to be stored with the new image

Table 1 – Extended Firmware Overview (* All values are in octets – u8)

By understanding and working with the Belkin Extended Firmware format, it is hoped that the OpenWRT project can be extended to support the Belkin hardware.

4.2 Extended Firmware Header - LOAD

The Extended Firmware Header creates the type "LOAD" data block. It adds very little value to the previously understood type "HDR0" data block. It seems to act as an identifying mechanism for the fact that the file has an extended footer.

Octet															
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Magic				File Size				CRC32				Version			
Reserved				Reserved				Reserved				Data …			

Table 2 - Extended Firmware Header Diagram

Offset*	Length*	Field Name	Description	Value
0x0	0x4	Magic	Magic identifier for this data block. This value is constant.	0x44414F4C ("LOAD")
0x4	0x4	File Size	Size, in octets, of the firmware image file (32bit, little-endian).	Dynamic
0x8	0x4	CRC32	CRC32 calculation for the data between offset 0xC and the end of the firmware image file (32bit, little-endian, bit-inverted).	Dynamic
0xC	0x4	Version	Version number for this data block. This value is constant.	0x00000800
0x10	0x4	Reserved	Unknown. This value is constant.	0x00000000
0x14	0x4	Reserved	Unknown. This value is constant.	0x00000000
0x18	0x4	Reserved	Unknown. This value is constant.	0x00000000
0x1B	…	Data	Data for this block. This value is constant.	Data block "HDR0" …

Table 3 - Extended Firmware Header Detail (* All values are in octets – u8)

Structure

```
struct extended_header {
        uint32_t        Magic
        uint32_t        Length
        uint32_t        crc32
        uint32_t        Version
        uint32_t        reserved_0
        uint32_t        reserved_1
        uint32_t        reserved_2
}
```

Table 4 - Extended Firmware Header GNU / ANSI-C Structure

4.3 Extended Firmware Footer - NVAR

The Extended Firmware Footer creates the type "NVAR" data block. It is used to apply a set of preferred or default NVRAM values to the flash space of the device, for use with the new firmware being uploaded in the type "HDR0" data block.

Octet															
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Magic				File Size				CRC32				Version			
Reserved				Reserved				Reserved				Data …			

Table 5 - Extended Firmware Footer Diagram

Offset*	Length*	Field Name	Description	Value
0x0	0x4	Magic	Magic identifier for this data block. This value is constant.	0x5241564E ("NVAR")
0x4	0x4	File Size	Size, in octets, of this data block, from offset 0x0 to the firmware image file (32bit, little-endian).	Dynamic
0x8	0x4	CRC32	CRC32 calculation for the data between offset 0xC and the end of the firmware image file (32bit, little-endian, bit-inverted).	Dynamic
0xC	0x4	Version	Version number for this data block. This value is constant.	0x00008000
0x10	0x4	Reserved	Unknown. This value is constant.	0x00000000
0x14	0x4	Reserved	Unknown. This value is constant.	0x00000000
0x18	0x4	Reserved	Unknown. This value is constant.	0x00000000
0x1B	…	Data	NVRAM records (variables and values), stored in the ASCII text format of; "variable=value", 0x0a	Dynamic

Table 6 - Extended Firmware Footer Detail (* All values are in octets – u8)

Structure

```
struct extended_footer {
        uint32_t        Magic
        uint32_t        Length
        uint32_t        crc32
        uint32_t        Version
        uint32_t        reserved_0
        uint32_t        reserved_1
        uint32_t        reserved_2
}
```

Table 7 - Extended Firmware Footer GNU / ANSI-C Structure

4.4 Format of the “user.conf” file

When you choose to export / download your wireless router configuration, the file that is sent to you is full of NVRAM variables stored in the format identified here.

It is important to break down this file format, as it makes it possible to export an existing user-defined or factory firmware configuration for import back into the device with any other firmware type or version.

Octet															
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
File Size		Data …													
MD5															

Table 8 - Configuration File Format Diagram

Offset*	Length*	Field Name	Description	Value
0x0	0x2	File Size	Size, in octets, of this file, from offset 0x0 to the end of file (16bit, little-endian).	Dynamic
0x2	(0xN) Dynamic	Data	NVRAM records (variables and values), stored in the ASCII text format of; “variable=value”, 0x0a	Dynamic (ASCII)
(0X2 + 0xN) Dynamic	0x10	MD5	MD5 calculation for the data between offset 0x2 and the end of the NVRAM data [either offset 0x2 + 0xN or offset File Size – 0x10] (32bit, little-endian).	Dynamic

Table 9 – Configuration File Format Detail (* All values are in octets – u8)

5 Towards open source

5.1 Overview

Repeated testing revealed that, for some reason, booting the wireless router with a modified CRAMFS does not corrupt the boot process. However, booting with a new kernel does (namely due to the OpenWRT kernel not supporting CRAMFS).

The following process aims to facilitate further discovery; to provide a platform for understanding the uniqueness of the Belkin firmware, versus the Linksys WRT54G.

To this end the following process does not aim to create the desired additional functionality identified in earlier chapters, instead, it aims to provide a "telnet" root shell for further exploration.

Note that this documentation has been written for the F5D7230-4 and in a factory-default configuration. Following these instructions will void your Belkin warrantee, and may render your router unusable.

No warrantee is implied or intended when you choose to follow these instructions.

Proceed at your own risk.

5.2 Acquiring native Belkin firmware online

The Belkin firmware can be found at the Belkin "54g Wireless DSL/Cable Gateway Router – F5D7230-4" Support page, here;

```
http://web.belkin.com/support/download/download.asp?download=F5D7230-4
```

The firmware used for testing this process was F5D7230-4-AU_V4.00.03.BIN;

```
http://web.belkin.com/support/download/downloaddetails.asp?file_id=1691
```

5.3 Splitting out the firmware into a kernel and root file system

Breaking the firmware down into its component files is largely an exercise for the reader. A program could be created to automate the process – but the judicious use of "hexdump –C" and "dd" will yield good results.

All of the offset data required can be found in section 4, above (though do not forget to take into account the little-endian architecture of the Mipsel chipset).

Once the firmware has been displaced, store the files as follows;

- The kernel image → ./belkin-kernel.bin
- The CRAMFS image → ./belkin-cramfs.bin

Acquire a desirable user.conf by configuring the Belkin via its native HTTP interface and then exporting the configuration;

- An exported user.conf file → ./belkin-user.conf

5.4 Creating a firmware workspace

As CRAMFS is a ROM-like file system (like an ISO image), there is no means to modify an existing image. Instead, a copy of the files from the image can be stored on an ordinary (EXT2 / EXT3) file system, to be worked upon, until a new CRAMFS image is ready to be created.

A firmware workspace will allow you to perform this and other firmware operations.

Archive a complete copy of the native firmware CRAMFS (as a collection of files);

```
mkdir mnt
mount –o loop ./belkin-cramfs.bin ./mnt
cd mnt
tar cvfzp ../cramfs.tgz *
cd ..
umount ./mnt
```

Then, make a workspace for modifying the firmware CRAMFS files;

```
mkdir fs
cd fs
tar xvfzp ../cramfs.tgz
cd ..
```

The “fs/” directory is now the firmware workspace directory.

5.5 Modifying the firmware workspace to get a root shell

The program “door” was authored to provide a very simplistic method for connecting an interactive shell to a pre-determined TCP port.

In the sample, door.c below, door has been configured to support standard in, standard out, and standard error through an instance of “/bin/sh –i”, on 2323/TCP (for all IP interfaces that the router has configured);

```
/*
 * Copyright (C) 2004
 * "Ian (Larry) Latter" <ian dot latter at midnightcode dot org>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
 * General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

/*
 * door
 *
 *   -- Generic shell door, originally written October 2004 as
 *      a dirty netcat clone to access the Belkin (broadcom) 54g
 *      wireless router.  Reduced to the most basic components to
 *      support simple cross compilation (mipsel for Belkin)
 *
 *   -- Adapted from "gibd00r v3.0" by axess of March 2000
 *
 *   -- Binds a shell to a port, by default we use 2323
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <strings.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <signal.h>
#include <unistd.h>

int main(int argc, char *argv[])
{
        struct sockaddr_in local;
        struct sockaddr_in remote;
        char pwd[256];

        int s,r,size,uid;

        signal(SIGCHLD, SIG_IGN);

        size = sizeof(struct sockaddr_in);
        memset(pwd, 0, 256);
```

```
        local.sin_family = AF_INET;
        local.sin_port = htons(2323);
        local.sin_addr.s_addr = INADDR_ANY;

        if((s=socket(AF_INET, SOCK_STREAM, 0)) == 1)
        {
                perror("Socket");
                exit(1);
        }

        if(bind(s, (struct sockaddr *)&local, sizeof(struct sockaddr))
                == -1)
        {
                perror("Port");
                exit(1);
        }

        if(listen(s, 5) == -1)
        {
                perror("Listen");
                exit(1);
        }

        uid = getuid();

        for(;;)
        {
                if((r=accept(s, (struct sockaddr *)&remote, &size)) == -1)
                {
                        perror("accept");
                        exit(1);
                }

                if(!fork())
                {
                        close(0);
                        close(1);
                        close(2);
                        dup2(r,0);
                        dup2(r,1);
                        dup2(r,2);

                        chdir("/");

                printf("Welcome to -door- from midnightcode.org\n\n");
                if(getcwd(pwd, 255) != NULL) {
                  printf("You are user id [%d] in directory [%s]\n",
                                uid, pwd);
                } else {
                  printf("You are user id [%d]\n", uid, pwd);
                }
                printf("You have stdin, stdout and stderr on this "
                        "socket\n");
                printf("_____________________________"
                        "_____________________________\n\n");

                        execl("/bin/sh","/bin/sh","-i",(char *)0);
                        close(r);
                        exit(0);
                }
                close(r);
        }
}
```

Compile door as a static binary against uClibc for Mipsel, using the OpenWRT cross-compiling environment, to produce the binary required.

Before the new binary can be added to the firmware, some space must be made.

Remove the “parent-control” binary to free up about 65kbytes of flash space;

```
cd fs/usr/sbin/
rm –f parent-control
cd ../../..
```

Now, copy the new door binary into your firmware workspace;

```
cp door fs/usr/sbin/door
chown 520:8 fs/usr/sbin/door
chmod 755 fs/usr/sbin/door
```

To get door to run in an embedded environment that contains no init scripts, an existing boot-launched process needs to be intercepted. Thus, by completing the following commands, a shell-script-based hook is created to launch the door process on each router boot.

Modify the httpd initialisation process to launch additional processes;

```
cd fs/usr/sbin/
mv httpd httpd.bin
chown 520:8 httpd.bin
chmod 755 httpd.bin
echo “#!/bin/sh” > httpd
echo “/usr/sbin/door &” >> httpd
echo “/usr/sbin/httpd.bin $*” >> httpd
chown 520:8 httpd
chmod 755 httpd
cd ../../..
```

The firmware workspace is now fully equipped with an interactive terminal interface.

5.6 Constructing the new firmware image from the workspace

The program “belky” (belky.c below) was authored to provide an easy way to construct a Belkin Extended Firmware file from a “trx” type HDR0 data block (kernel and file system), and a user.conf file.

Originally designed to run on both Linux and Windows, belky was never tested outside of the Linux OS.

Using your native Linux compiler, Make the following program;

```
/*
 * Copyright (C) 2004
 * "Ian (Larry) Latter" <ian dot latter at midnightcode dot org>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
 * General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

/*
 * October 10, 2004
 *
 * This "belky" tool is a hacked addition to OpenWRT's "trx" tool
 * (as at July 29, 2004) which in itself is a hacked replacement for
 * the 'trx' utility used to create wrt54g .trx firmware files.  It
 * isn't pretty, but ... you know ...
 *
 *   -- Adapted from "trx" by Manuel Novoa III of August 2004
 *
 */

/* linux makefile touches stdafx.h .. Visual Studio blows */
#include "stdafx.h"

#include <stdio.h>
#include <stdlib.h>
#include <stddef.h>
#include <stdint.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>
#include <endian.h>
#include <byteswap.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#define  O_BINARY 0

#if __BYTE_ORDER == __BIG_ENDIAN
#define STORE32_LE(X)             bswap_32(X)
#elif __BYTE_ORDER == __LITTLE_ENDIAN
#define STORE32_LE(X)             (X)
#else
```

```
#error unkown endianness!
#endif

uint32_t crc32buf(char *buf, size_t len);

/****************************************************************/

/* Program Info */
char    vernam[32]       = "belky";
char    vernum[6]        = "0.3";


/* home-made goodies .. */

#define BELKY_MAGIC_H    0x44414F4C       /* "LOAD" */
#define BELKY_MAGIC_F    0x5241564E       /* "NVAR" */
#define BELKY_VERS_H     0x00000800       /* Version Magic */
#define BELKY_VERS_F     0x00008000       /* Version Magic */
#define BELKY_MAX_LEN    0x3A0000

struct belky_header {
        uint32_t magic;          /* 0  ...  3 "LOAD"
        uint32_t len;            /* 4  ...  7  Length of file including
header   */
        uint32_t crc32;          /* 8  ... 11  32-bit CRC (ver to eof),
LE, !bit */
        uint32_t version;        /* 12 ... 27  0x8000 (32bit LE), 0, 0, 0
        uint32_t pad_0;
        uint32_t pad_1;
        uint32_t pad_2;
        char offset_data;        /* not a header field - sneaky data
offset only */
};

struct belky_header *header;

struct belky_footer {
        uint32_t magic;          /* 0  ...  3 "NVAR"
        uint32_t len;            /* 4  ...  7  Length of footer including
magic   */
        uint32_t crc32;          /* 8  ... 11  32-bit CRC (ver to eof), LE
        uint32_t version;        /* 12 ... 27  0x8000 (32bit LE), 0, 0, 0
        uint32_t pad_0;
        uint32_t pad_1;
        uint32_t pad_2;
        char offset_data;        /* not a footer field - sneaky data
offset only */
};

struct belky_footer *footer;



/* Transient program global variables */
uint32_t config_maxlen;
char             config_trx[256];
char             config_output[256];
char             config_userconf[256];

int              allow_dumb_defaults = 1;

/****************************************************************/


int
```

```
input_validation(void) {

        /* output file check */
        if (strlen(config_output) == 0) {
                if (allow_dumb_defaults) {
                        strncpy(config_output, "firmware.belkin", 255);
                } else {
                        printf("%s: no output file specified "
                                "(try --help).\n", vernam);
                        return -1;
                }
        }

        /* OpenWRT firmware file check */
        if (strlen(config_trx) == 0) {
                if (allow_dumb_defaults) {
                        strncpy(config_trx, "linux.trx", 255);
                } else {
                        printf("%s: no OpenWRT firmware file specified "
                                "(try --help).\n", vernam);
                        return -1;
                }
        }

        /* user.conf file check */
        if (strlen(config_userconf) == 0) {
                if (allow_dumb_defaults) {
                        strncpy(config_userconf, "user.conf", 255);
                } else {
                        printf("%s: no user.conf file specified "
                                "(try --help).\n", vernam);
                        return -1;
                }
        }

        /* maxlen check */
        if (config_maxlen > BELKY_MAX_LEN) {
                printf("%s: *warning* maxlen [%u] exceeds default maximum
"
                        " [%u]\n", vernam, config_maxlen, BELKY_MAX_LEN);
                printf("%s: Be very careful, you may overwrite NVRAM\n",
                        vernam);
        } else {
                config_maxlen = BELKY_MAX_LEN;
        }


        return 0;
}


int
read_in_data_file(char * data_file, void * buffer, int maxbytes, int type)
{
        int fd;
        int bytes;
        char filesize[3];

        bytes = 0;

        if (maxbytes < 0) {
                printf("%s: spare buffer space is negative -- "
                        "increase maxlen (?)\n",
                        vernam);
                return -1;
```

```
        }

        if ((fd = open(data_file, O_BINARY | O_RDONLY )) < 0) {
                printf("%s: unable to open file [%s] for reading.\n",
                        vernam, data_file);
                return -1;
        }

        if(lseek(fd, 0, SEEK_SET) < 0) {
                printf("%s: unable to seek to start of file [%s]\n",
                        vernam, data_file);
                return -1;
        }

        /* special work for user.conf file */
        if (type == 2) {
                memset(filesize, 0, 3);
                bytes = read(fd, filesize, 2);
                if (bytes < 0) {
                        printf("%s: unable to read from user.conf file
[%s].\n",
                                vernam, data_file);
                        return -1;
                }
                bytes = (uint8_t)filesize[1] * 256 + (uint8_t)filesize[0];
                if (bytes > 0) {
                        if ((bytes - 18) <= maxbytes) {
                                maxbytes = bytes - 18;
                        } else {
                                printf("%s: more data in file [%s] than
space "
                                        "left in buffer -- increase
maxlen?\n",
                                        vernam, data_file);
                                return -1;
                        }
                } else {
                        printf("%s: unable to read file size from header
of "
                                "file [%s].\n", vernam, data_file);
                        return -1;
                }
        }

        bytes = read(fd, buffer, maxbytes);

        close(fd);

        if(bytes < 0) {
                printf("%s: unable to read from file [%s].\n",
                        vernam, data_file);
                return -1;
        }

        return bytes;
}


int
write_firmware(char * data_file, char * buffer, int maxbytes) {
        int fd;

        if ((fd = open(data_file, O_BINARY | O_RDWR | O_CREAT | O_TRUNC,
                         S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH )) < 0) {
                printf("%s: unable to open firmware file [%s] for
```

```
writing.\n",
                        vernam, data_file);
                return -1;
        }

        if (write(fd, buffer, maxbytes) <= 0) {
                printf("%s: unable to write to firmware file [%s].\n",
                        vernam, data_file);
               return -1;
        }

        close(fd);

        return 0;
}


int
fill_headers(uint32_t filelen, uint32_t footerlen) {

        header->magic    = STORE32_LE(BELKY_MAGIC_H);
        header->len      = STORE32_LE(filelen);
        header->crc32    = crc32buf((char *) &header->version,
                                filelen - offsetof(struct belky_header,
version));
        header->crc32    = STORE32_LE(header->crc32);
        header->version  = STORE32_LE(BELKY_VERS_H);

        footer->magic    = STORE32_LE(BELKY_MAGIC_F);
        footer->len      = STORE32_LE(footerlen);
        footer->version  = STORE32_LE(BELKY_VERS_F);
        footer->crc32    = crc32buf((char *) &footer->version, footerlen);
        footer->crc32    = STORE32_LE(footer->crc32);

        return 0;
}


int
write_debug(char * data_file, char * buffer, int maxbytes) {
        int fd;

        if ((fd = open(data_file, O_BINARY | O_RDWR | O_CREAT | O_TRUNC ))
                < 0) {
                printf("%s: unable to open debug file [%s] for
writing.\n",
                        vernam, data_file);
                return -1;
        }

        if (write(fd, buffer, maxbytes) <= 0) {
                printf("%s: unable to write to debug file [%s].\n",
                        vernam, data_file);
               return -1;
        }

        close(fd);

        return 0;
}



int
usage(int errlvl, char * runbin) {
```

```
        /* how many printf's must a man print out, before he is truly a
man .. */
        printf("\n");

        printf("%s: Usage;\n\n", vernam);
        printf("  %s [-t {linux.trx}] [-u {user.conf}] [-o
{firmware.belkin}] [-m maxlen]\n",
                runbin);
        printf("\n");
        printf("%s: Parameters;\n\n", vernam);
        printf("  Long       Short Args        Description\n");
        printf("  ~~~~~~~~~~ ~~~~~ ~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n");
        printf("  --help     -h                Display this help
screen\n");
        printf("  --maxlen   -m    {num}       Over-ride / set firmware
max length to {num}\n");
        printf("  --trx      -t    {file}      Read OpenWRT TRX firmware
file {file}\n");
        printf("  --output   -o    {file}      Write Belkin firmware to
file {file}\n");
        printf("  --userconf -u    {file}      Read Belkin \"user.conf\"
file {file}\n");
        printf("  --version  -V                Display version and
exit\n");
        printf("\n");

        /* the answer, my friend, is exit number one ...   */
        /* the answer is exit number one ..                        */
        exit(errlvl);
}


void
parse_args(int argc, char** argv) {
        int argn;

        /* defaults */
        config_maxlen                   = 0;
        memset(config_trx,              0, 256);
        memset(config_output,           0, 256);
        memset(config_userconf,         0, 256);

        argn = 1;
        while (argn < argc && argv[argn][0] == '-') {
                // -h, --help
                if (strcmp(argv[argn], "-h") == 0 ||
                     strcmp(argv[argn], "--help") == 0) {
                        usage(0, argv[0]);
                }
                // -m, --maxlen
                else if ((strcmp(argv[argn], "-m") == 0 ||
                     strcmp(argv[argn], "--maxlen") == 0)
                     && argn + 1 < argc) {
                        ++argn;
                        config_maxlen = (uint32_t)atoi(argv[argn]);
                }
                // -t, --trx
                else if ((strcmp(argv[argn], "-t") == 0 ||
                     strcmp(argv[argn], "--trx") == 0)
                     && argn + 1 < argc) {
                        ++argn;
                        strncpy(config_trx, argv[argn], 255);
                }
                // -o, --output
```

```
                else if ((strcmp(argv[argn], "-o") == 0 ||
                     strcmp(argv[argn], "--output") == 0)
                     && argn + 1 < argc) {
                         ++argn;
                         strncpy(config_output, argv[argn], 255);
                }
                // -u, --userconf
                else if ((strcmp(argv[argn], "-u") == 0 ||
                     strcmp(argv[argn], "--userconf") == 0)
                     && argn + 1 < argc) {
                         ++argn;
                         strncpy(config_userconf, argv[argn], 255);
                }
                // -V, --version
                else if (strcmp(argv[argn], "-V") == 0 ||
                     strcmp(argv[argn], "--version") == 0) {
                         printf("%s v%s\n", vernam, vernum);
                         exit(0);
                }
                // doh
                else {
                         printf("%s: Unrecognised command parameter [%s],"
                                 " exiting.\n", vernam, argv[argn]);
                         usage(1, argv[0]);
                }
                ++argn;
        }
        if ( argn != argc )
                usage(1, argv[0]);
}



// Evil way to make main main
#ifdef WIN32
int _tmain(int argc, _TCHAR* argv[]) {
#else
int main(int argc, char **argv) {
#endif /* win32 */

        int maxlen;
        int bytes;
        int totlen;
        int fotlen;

#if O_BINARY
        _setmode (fileno(stdin), O_BINARY);
        _setmode (fileno(stdout), O_BINARY);
#endif

        /* obtain configuration state */
        parse_args(argc, argv);

        /* input validation check */
        if (input_validation() < 0) {
                return -1;
        }

        /* create memory region for firmware storage */
        if ((header = (struct belky_header *)malloc(config_maxlen)) ==
NULL) {
                printf("%s: unable to malloc [%d] bytes.\n",
                        vernam, config_maxlen);
                return -1;
```

```
        }
        memset(header, 0, config_maxlen);

        /* read in the linux.trx file */
        maxlen = config_maxlen - sizeof(struct belky_header) -
                 sizeof(struct belky_footer) + (sizeof(char) * 2);
        bytes = read_in_data_file(config_trx, (void *)&header-
>offset_data,
                maxlen, 1);
        if (bytes < 0) {
                return -1;
        }
        totlen = sizeof(struct belky_header) - sizeof(char) + bytes - 3;

        /* set the correct footer offset */
        footer = (struct belky_footer *)((char *)header +
(uint32_t)totlen);

        /* read in the user.conf file */
        maxlen = config_maxlen - sizeof(struct belky_footer) +
sizeof(char) -
                 totlen;
        bytes = read_in_data_file(config_userconf, (void *)&footer-
>offset_data,
                maxlen, 2);
        if (bytes < 0) {
                return -1;
        }
        totlen += sizeof(struct belky_footer) - sizeof(char) + bytes - 3;
        fotlen =  sizeof(footer->version) + sizeof(footer->pad_0) +
                        sizeof(footer->pad_1) + sizeof(footer->pad_2) +
bytes;

        /* time to colour in the picture */
        if (fill_headers((uint32_t)totlen, (uint32_t)fotlen) < 0) {
                return -1;
        }

        /* write firmware to firmware.belkin file */
        maxlen = totlen;
        bytes = write_firmware(config_output, (char *)header, maxlen);
        if (bytes < 0) {
                return -1;
        }


        return(0);
}


/*********************************************************************/
/* The following was grabbed and tweaked from the old snippets collection
 * of public domain C code. */

/*********************************************************************\
|* Demonstration program to compute the 32-bit CRC used as the frame  *|
|* check sequence in ADCCP (ANSI X3.66, also known as FIPS PUB 71     *|
|* and FED-STD-1003, the U.S. versions of CCITT's X.25 link-level     *|
|* protocol).  The 32-bit FCS was added via the Federal Register,     *|
|* 1 June 1982, p.23798.  I presume but don't know for certain that   *|
|* this polynomial is or will be included in CCITT V.41, which        *|
|* defines the 16-bit CRC (often called CRC-CCITT) polynomial.  FIPS  *|
|* PUB 78 says that the 32-bit FCS reduces otherwise undetected       *|
|* errors by a factor of 10^-5 over 16-bit FCS.                       *|
\*********************************************************************/
```

```
/* Copyright (C) 1986 Gary S. Brown.  You may use this program, or
   code or tables extracted from it, as desired without restriction.*/

/* First, the polynomial itself and its table of feedback terms.  The  */
/* polynomial is                                                       */
/* X^32+X^26+X^23+X^22+X^16+X^12+X^11+X^10+X^8+X^7+X^5+X^4+X^2+X^1+X^0 */
/* Note that we take it "backwards" and put the highest-order term in  */
/* the lowest-order bit.  The X^32 term is "implied"; the LSB is the   */
/* X^31 term, etc.  The X^0 term (usually shown as "+1") results in    */
/* the MSB being 1.                                                    */

/* Note that the usual hardware shift register implementation, which   */
/* is what we're using (we're merely optimizing it by doing eight-bit  */
/* chunks at a time) shifts bits into the lowest-order term.  In our   */
/* implementation, that means shifting towards the right.  Why do we   */
/* do it this way?  Because the calculated CRC must be transmitted in  */
/* order from highest-order term to lowest-order term.  UARTs transmit */
/* characters in order from LSB to MSB.  By storing the CRC this way,  */
/* we hand it to the UART in the order low-byte to high-byte; the UART */
/* sends each low-bit to hight-bit; and the result is transmission bit */
/* by bit from highest- to lowest-order term without requiring any bit */
/* shuffling on our part.  Reception works similarly.                  */

/* The feedback terms table consists of 256, 32-bit entries.  Notes:   */
/*                                                                     */
/*  1. The table can be generated at runtime if desired; code to do so */
/*     is shown later.  It might not be obvious, but the feedback      */
/*     terms simply represent the results of eight shift/xor opera-    */
/*     tions for all combinations of data and CRC register values.     */
/*                                                                     */
/*  2. The CRC accumulation logic is the same for all CRC polynomials, */
/*     be they sixteen or thirty-two bits wide.  You simply choose the */
/*     appropriate table.  Alternatively, because the table can be     */
/*     generated at runtime, you can start by generating the table for */
/*     the polynomial in question and use exactly the same "updcrc",   */
/*     if your application needn't simultaneously handle two CRC       */
/*     polynomials.  (Note, however, that XMODEM is strange.)          */
/*                                                                     */
/*  3. For 16-bit CRCs, the table entries need be only 16 bits wide;   */
/*     of course, 32-bit entries work OK if the high 16 bits are zero. */
/*                                                                     */
/*  4. The values must be right-shifted by eight bits by the "updcrc"  */
/*     logic; the shift must be unsigned (bring in zeroes).  On some   */
/*     hardware you could probably optimize the shift in assembler by  */
/*     using byte-swap instructions.                                   */

static const uint32_t crc_32_tab[] = { /* CRC polynomial 0xedb88320 */
0x00000000, 0x77073096, 0xee0e612c, 0x990951ba, 0x076dc419, 0x706af48f,
0xe963a535, 0x9e6495a3, 0x0edb8832, 0x79dcb8a4, 0xe0d5e91e, 0x97d2d988,
0x09b64c2b, 0x7eb17cbd, 0xe7b82d07, 0x90bf1d91, 0x1db71064, 0x6ab020f2,
0xf3b97148, 0x84be41de, 0x1adad47d, 0x6ddde4eb, 0xf4d4b551, 0x83d385c7,
0x136c9856, 0x646ba8c0, 0xfd62f97a, 0x8a65c9ec, 0x14015c4f, 0x63066cd9,
0xfa0f3d63, 0x8d080df5, 0x3b6e20c8, 0x4c69105e, 0xd56041e4, 0xa2677172,
0x3c03e4d1, 0x4b04d447, 0xd20d85fd, 0xa50ab56b, 0x35b5a8fa, 0x42b2986c,
0xdbbbc9d6, 0xacbcf940, 0x32d86ce3, 0x45df5c75, 0xdcd60dcf, 0xabd13d59,
0x26d930ac, 0x51de003a, 0xc8d75180, 0xbfd06116, 0x21b4f4b5, 0x56b3c423,
0xcfba9599, 0xb8bda50f, 0x2802b89e, 0x5f058808, 0xc60cd9b2, 0xb10be924,
0x2f6f7c87, 0x58684c11, 0xc1611dab, 0xb6662d3d, 0x76dc4190, 0x01db7106,
0x98d220bc, 0xefd5102a, 0x71b18589, 0x06b6b51f, 0x9fbfe4a5, 0xe8b8d433,
0x7807c9a2, 0x0f00f934, 0x9609a88e, 0xe10e9818, 0x7f6a0dbb, 0x086d3d2d,
0x91646c97, 0xe6635c01, 0x6b6b51f4, 0x1c6c6162, 0x856530d8, 0xf262004e,
0x6c0695ed, 0x1b01a57b, 0x8208f4c1, 0xf50fc457, 0x65b0d9c6, 0x12b7e950,
0x8bbeb8ea, 0xfcb9887c, 0x62dd1ddf, 0x15da2d49, 0x8cd37cf3, 0xfbd44c65,
0x4db26158, 0x3ab551ce, 0xa3bc0074, 0xd4bb30e2, 0x4adfa541, 0x3dd895d7,
```

```
0xa4d1c46d, 0xd3d6f4fb, 0x4369e96a, 0x346ed9fc, 0xad678846, 0xda60b8d0,
0x44042d73, 0x33031de5, 0xaa0a4c5f, 0xdd0d7cc9, 0x5005713c, 0x270241aa,
0xbe0b1010, 0xc90c2086, 0x5768b525, 0x206f85b3, 0xb966d409, 0xce61e49f,
0x5edef90e, 0x29d9c998, 0xb0d09822, 0xc7d7a8b4, 0x59b33d17, 0x2eb40d81,
0xb7bd5c3b, 0xc0ba6cad, 0xedb88320, 0x9abfb3b6, 0x03b6e20c, 0x74b1d29a,
0xead54739, 0x9dd277af, 0x04db2615, 0x73dc1683, 0xe3630b12, 0x94643b84,
0x0d6d6a3e, 0x7a6a5aa8, 0xe40ecf0b, 0x9309ff9d, 0x0a00ae27, 0x7d079eb1,
0xf00f9344, 0x8708a3d2, 0x1e01f268, 0x6906c2fe, 0xf762575d, 0x806567cb,
0x196c3671, 0x6e6b06e7, 0xfed41b76, 0x89d32be0, 0x10da7a5a, 0x67dd4acc,
0xf9b9df6f, 0x8ebeeff9, 0x17b7be43, 0x60b08ed5, 0xd6d6a3e8, 0xa1d1937e,
0x38d8c2c4, 0x4fdff252, 0xd1bb67f1, 0xa6bc5767, 0x3fb506dd, 0x48b2364b,
0xd80d2bda, 0xaf0a1b4c, 0x36034af6, 0x41047a60, 0xdf60efc3, 0xa867df55,
0x316e8eef, 0x4669be79, 0xcb61b38c, 0xbc66831a, 0x256fd2a0, 0x5268e236,
0xcc0c7795, 0xbb0b4703, 0x220216b9, 0x5505262f, 0xc5ba3bbe, 0xb2bd0b28,
0x2bb45a92, 0x5cb36a04, 0xc2d7ffa7, 0xb5d0cf31, 0x2cd99e8b, 0x5bdeae1d,
0x9b64c2b0, 0xec63f226, 0x756aa39c, 0x026d930a, 0x9c0906a9, 0xeb0e363f,
0x72076785, 0x05005713, 0x95bf4a82, 0xe2b87a14, 0x7bb12bae, 0x0cb61b38,
0x92d28e9b, 0xe5d5be0d, 0x7cdcefb7, 0x0bdbdf21, 0x86d3d2d4, 0xf1d4e242,
0x68ddb3f8, 0x1fda836e, 0x81be16cd, 0xf6b9265b, 0x6fb077e1, 0x18b74777,
0x88085ae6, 0xff0f6a70, 0x66063bca, 0x11010b5c, 0x8f659eff, 0xf862ae69,
0x616bffd3, 0x166ccf45, 0xa00ae278, 0xd70dd2ee, 0x4e048354, 0x3903b3c2,
0xa7672661, 0xd06016f7, 0x4969474d, 0x3e6e77db, 0xaed16a4a, 0xd9d65adc,
0x40df0b66, 0x37d83bf0, 0xa9bcae53, 0xdebb9ec5, 0x47b2cf7f, 0x30b5ffe9,
0xbdbdf21c, 0xcabac28a, 0x53b39330, 0x24b4a3a6, 0xbad03605, 0xcdd70693,
0x54de5729, 0x23d967bf, 0xb3667a2e, 0xc4614ab8, 0x5d681b02, 0x2a6f2b94,
0xb40bbe37, 0xc30c8ea1, 0x5a05df1b, 0x2d02ef8d
};

#define UPDC32(octet,crc) (crc_32_tab[((crc) ^ (octet)) & 0xff] ^ ((crc) >>
8))

uint32_t crc32buf(char *buf, size_t len)
{
      uint32_t crc;

      crc = 0xFFFFFFFF;

      for ( ; len; --len, ++buf)
      {
            crc = UPDC32(*buf, crc);
      }

      return crc;
}
```

Ensure that the “belky” binary, and the “trx” binary (from OpenWRT) are both in the current directory.

Then, make a new firmware image from the original firmware files, the workspace and the user.conf;

```
rm –f ./new-cramfs.bin ./new-image.bin ./new-firmware.bin
mkfs.cramfs fs/ ./new-cramfs.bin
./trx –o ./new-image.bin ./belkin-image-kernel.bin ./new-cramfs.bin
./belky –o ./new-firmware.bin –t ./new-image.bin –u ./belkin-user.conf
```

The file “./new-firmware.bin” contains the original Belkin kernel, but with the modified CRAMFS, and your preferred user.conf NVRAM settings.

5.7 Uploading the new firmware to the router

Follow this process to upload the new firmware and do not use the Windows TFTP client as it will not work;

1. Shutdown the router (unplug it from the power supply).
2. Enter the following at a Linux shell prompt;

```
tftp 192.168.2.1
tftp> binary
tftp> rexmt 1
tftp> trace
Packet tracing on.
tftp> put new-firmware.bin
```

3. Boot the router (plug it in to the power supply after it has been powered off for at least three (3) seconds).

Note; Where possible, plug the router directly into the Linux workstation, as some switches take a long time (up to half a second) to negotiate the port characteristics. On such switches it is virtually impossible to flash the router.

Once flashed, the router will pause, then reboot. Do not switch off the router or interfere with it during this process. Once the router lights settle (stop flash-cycling) then try either;

- A “telnet 192.168.2.1 2323” from the Linux prompt, or;
- A RAW PuTTY session to 192.168.2.1 at 2323/TCP (ensuring that “Implicit CR in every LF” is enabled, under Terminal)

You should obtain a root shell (you may need to wait up to three (3) seconds from the connection launch, before you see terminal output).

5.8 Debugging fault conditions

Once you've flashed the router, you may experience different error conditions;

1. If the router continues to flash-cycle, then your new firmware is not valid or is incorrect in some way. Try;
    - (Recovery) Re-flashing the router using an original (untainted) Belkin firmware, and the TFTP flashing instructions above.
    - (Repair) Confirm that you have followed the instructions for adding door, above. If you have not completed the instructions correctly, any number of error conditions may occur. Re-flash the router with your fixed firmware, and the TFTP flashing instructions above.

2. If the router appears to, either, enter an endless series of reboots or become intermittently ping-able, then your router may be overtly configured. Try;
    - (Recovery) Re-flashing the router using an original (untainted) Belkin firmware, and the TFTP flashing instructions above.
    - (Repair) Ensure that the router is not configured with a WAN interface address. This includes static addresses, ADSL modems, etc. It appears that the missing /usr/sbin/parent-control binary causes the router to corrupt its initialisation process, but only once the WAN interface is configured. Factory default NVRAM settings will yield success.

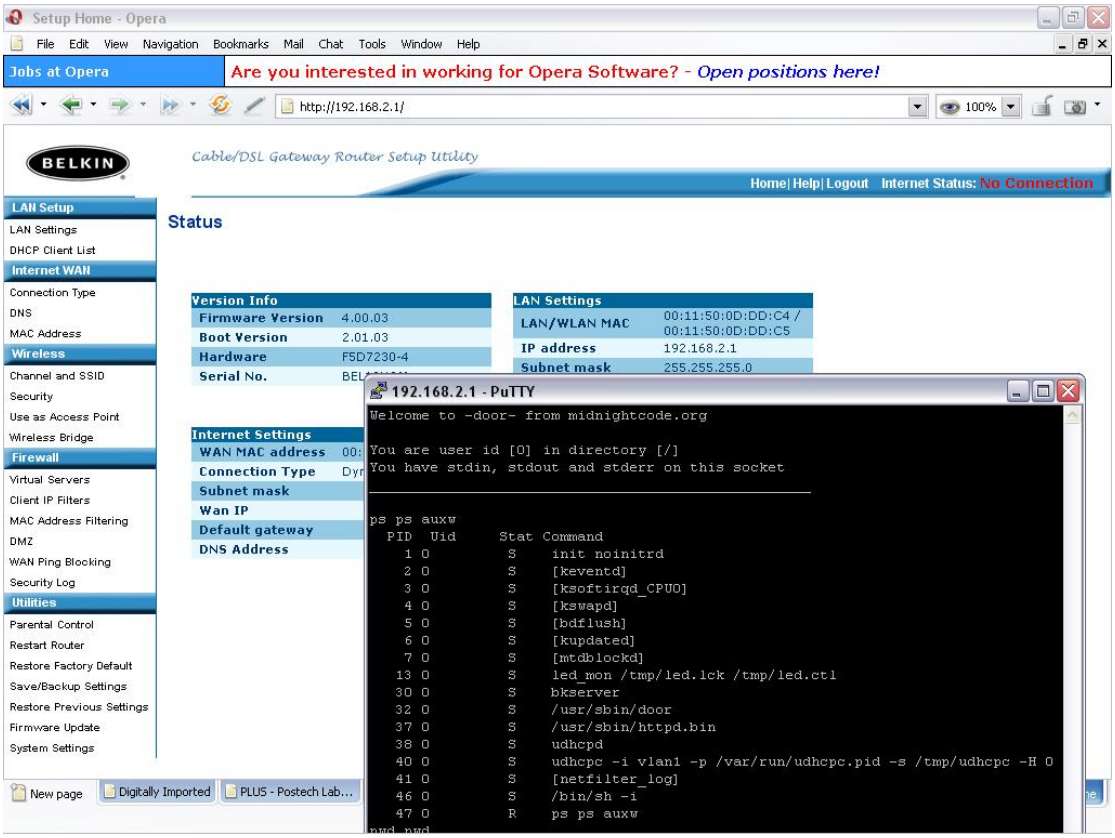
3. If the router appears to operate correctly, including rendering a valid HTTP interface, but your attempts to telnet to 2323/TCP are refused, then your firmware image may have been rejected (with the router rebooting to the previous firmware image). Try;
    - (Recovery) Re-flashing the router using an original (untainted) Belkin firmware, and the TFTP flashing instructions above.
    - (Repair) This typically occurs if the firmware is predictably corrupt. Symptoms include invalid CRC32 sums, file sizes and/or the firmware image being too large. Confirm that you have followed the instructions for adding door, above. If you have not completed the instructions correctly, any number of these error conditions may occur.

If all else fails, you should be able to recover your router by re-flashing it using an original (untainted) Belkin firmware, and the TFTP flashing instructions above.

6 Findings

6.1 Progress

The root shell becomes a simple way to readily access the router, as can be seen here;



With an exploratory platform available, it has been possible to collate a good deal of Linux-based information about the device.

6.2 /proc/kmsg

Output of “cat /proc/kmsg .”;

```
<4>CPU revision is: 00029007
<4>Primary instruction cache 8kb, linesize 16 bytes (2 ways)
<4>Primary data cache 4kb, linesize 16 bytes (2 ways)
<4>Linux version 2.4.20 (lchen@penguin.askey.com) (gcc version 3.0 20010422
(prerelease) with bcm4710a0 modifications) #8 Mon Dec 1 20:51:49 PST 2003
<4>Determined physical RAM map:
<4> memory: 00800000 @ 00000000 (usable)
<4>On node 0 totalpages: 2048
<4>zone(0): 2048 pages.
<4>zone(1): 0 pages.
<4>zone(2): 0 pages.
<4>Kernel command line: root=/dev/mtdblock2 noinitrd console=ttyS0,115200
<4>CPU: BCM4712 rev 1 at 200 MHz
<4>Calibrating delay loop... 199.47 BogoMIPS
<6>Memory: 6424k/8192k available (1255k kernel code, 1768k reserved, 108k
data, 64k init, 0k highmem)
<6>Dentry cache hash table entries: 1024 (order: 1, 8192 bytes)
<6>Inode cache hash table entries: 512 (order: 0, 4096 bytes)
<4>Mount-cache hash table entries: 512 (order: 0, 4096 bytes)
<4>Buffer-cache hash table entries: 1024 (order: 0, 4096 bytes)
<4>Page-cache hash table entries: 2048 (order: 1, 8192 bytes)
<4>Checking for 'wait' instruction...  unavailable.
<4>POSIX conformance testing by UNIFIX
<4>PCI: Fixing up bus 0
<4>PCI: Fixing up bridge
<4>PCI: Fixing up bus 1
<6>Linux NET4.0 for Linux 2.4
<6>Based upon Swansea University Computer Society NET3.039
<4>Initializing RT netlink socket
<4>Starting kswapd
<6>devfs: v1.12c (20020818) Richard Gooch (rgooch@atnf.csiro.au)
<6>devfs: boot_options: 0x1
<4>pty: 256 Unix98 ptys configured
<6>Serial driver version 5.05c (2001-07-08) with MANY_PORTS SHARE_IRQ
SERIAL_PCI enabled
<6>ttyS00 at 0xb8000300 (irq = 3) is a 16550A
<6>ttyS01 at 0xb8000400 (irq = 0) is a 16550A
<6>PPP generic driver version 2.4.2
<7>Physically mapped flash: Found an alias at 0x200000 for the chip at 0x0
<7>Physically mapped flash: Found an alias at 0x400000 for the chip at 0x0
<7>Physically mapped flash: Found an alias at 0x600000 for the chip at 0x0
<7>Physically mapped flash: Found an alias at 0x800000 for the chip at 0x0
<7>Physically mapped flash: Found an alias at 0xa00000 for the chip at 0x0
<7>Physically mapped flash: Found an alias at 0xc00000 for the chip at 0x0
<7>Physically mapped flash: Found an alias at 0xe00000 for the chip at 0x0
<7>Physically mapped flash: Found an alias at 0x1000000 for the chip at 0x0
<7>Physically mapped flash: Found an alias at 0x1200000 for the chip at 0x0
<7>Physically mapped flash: Found an alias at 0x1400000 for the chip at 0x0
<7>Physically mapped flash: Found an alias at 0x1600000 for the chip at 0x0
<7>Physically mapped flash: Found an alias at 0x1800000 for the chip at 0x0
<7>Physically mapped flash: Found an alias at 0x1a00000 for the chip at 0x0
<7>Physically mapped flash: Found an alias at 0x1c00000 for the chip at 0x0
<7>Physically mapped flash: Found an alias at 0x1e00000 for the chip at 0x0
<5> Amd/Fujitsu Extended Query Table v1.0 at 0x0040
<5>number of CFI chips: 1
<5>Flash device: 0x200000 at 0x1c000000
<5>Physically mapped flash: cramfs filesystem found at block 742
<5>Creating 5 MTD partitions on "Physically mapped flash":
<5>0x00000000-0x00020000 : "pmon"
<5>0x00020000-0x001f0000 : "linux"
<5>0x000b99d8-0x001f0000 : "rootfs"
```

```
<5>0x00004000-0x00006000 : "profile"
<5>0x001f0000-0x00200000 : "nvram"
<3>sflash: found no supported devices
<6>NET4: Linux TCP/IP 1.0 for NET4.0
<6>IP Protocols: ICMP, UDP, TCP
<6>IP: routing cache hash table of 512 buckets, 4Kbytes
<6>TCP: Hash tables configured (established 512 bind 1024)
<4>ip_conntrack version 2.1 (64 buckets, 512 max) - 344 bytes per conntrack
<4>ip_tables: (C) 2000-2002 Netfilter core team
<4>ipt_time loading
<6>NET4: Unix domain sockets 1.0/SMP for Linux NET4.0.
<6>NET4: Ethernet Bridge 008 for NET4.0
<1>802.1Q VLAN Support v1.7 Ben Greear <greearb@candelatech.com>
<1>All bugs added by David S. Miller <davem@redhat.com>
<4>VFS: Mounted root (cramfs filesystem) readonly.
<6>Mounted devfs on /dev
<6>Freeing unused kernel memory: 64k freed
<4>eth0: Broadcom BCM47xx 10/100 Mbps Ethernet Controller 3.50.21.0
<4>eth1: Broadcom BCM43XX 802.11 Wireless Controller  3.50.21.0 (Compiled
in . at 16:25:58 on Nov  5 2003)
<4>vlan1: Setting MAC address to  00 11 50 0d dd c4.
<4>VLAN (vlan1):  Underlying device (eth0) has same MAC, not checking
promiscious mode.
<6>vlan1: add 01:00:5e:00:00:01 mcast address to master interface
<4>Algorithmics/MIPS FPU Emulator v1.5
<6>vlan0: dev_set_promiscuity(master, 1)
<6>device eth0 entered promiscuous mode
<6>device vlan0 entered promiscuous mode
<4>
<4>Platform: 4712
<4><==sintInstallLEDs: VIOBA=b8000000
<6>device eth1 entered promiscuous mode
<6>br0: port 2(eth1) entering learning state
<6>br0: port 1(vlan0) entering learning state
<6>br0: port 2(eth1) entering forwarding state
<6>br0: topology change detected, propagating
<6>br0: port 1(vlan0) entering forwarding state
<6>br0: topology change detected, propagating
```


6.3 /proc/cpuinfo

Output of “cat /proc/cpuinfo .”;

```
system type             : Broadcom BCM947XX
processor               : 0
cpu model               : BCM3302 V0.7
BogoMIPS                : 199.47
wait instruction        : no
microsecond timers      : yes
tlb_entries             : 32
extra interrupt vector  : no
hardware watchpoint     : no
VCED exceptions         : not available
VCEI exceptions         : not available
dcache hits             : 4278056828
dcache misses           : 134217729
icache hits             : 4248412818
icache misses           : 329396824
instructions            : 0
```

6.4 /proc/mtd

Output of “cat /proc/mtd .”;

```
dev:    size   erasesize  name
mtd0: 00020000 00010000 "pmon"
mtd1: 001d0000 00010000 "linux"
mtd2: 00136628 00010000 "rootfs"
mtd3: 00002000 00002000 "profile"
mtd4: 00010000 00010000 "nvram"
```


6.5 /proc/pci

Output of “cat /proc/pci .”;

```
PCI devices found:
  Bus  0, device   0, function  0:
    Class 0501: PCI device 14e4:0800 (rev 1).
      IRQ 3.
      Non-prefetchable 32 bit memory at 0x18000000 [0x18000fff].
      Non-prefetchable 32 bit memory at 0x1fc00000 [0x1fffffff].
      Non-prefetchable 32 bit memory at 0x1c000000 [0x1dffffff].
      Non-prefetchable 32 bit memory at 0x1a000000 [0x1bffffff].
  Bus  0, device   1, function  0:
    Class 0280: PCI device 14e4:4320 (rev 1).
      IRQ 4.
      Non-prefetchable 32 bit memory at 0x18001000 [0x18001fff].
  Bus  0, device   2, function  0:
    Class 0200: PCI device 14e4:4713 (rev 1).
      IRQ 5.
      Non-prefetchable 32 bit memory at 0x18002000 [0x18002fff].
  Bus  0, device   3, function  0:
    Class 0c03: PCI device 14e4:4717 (rev 1).
      IRQ 6.
      Non-prefetchable 32 bit memory at 0x18003000 [0x18003fff].
  Bus  0, device   4, function  0:
    Class 0c03: PCI device 14e4:4716 (rev 1).
      IRQ 2.
      Non-prefetchable 32 bit memory at 0x18004000 [0x18004fff].
  Bus  0, device   5, function  0:
    Class 0b30: PCI device 14e4:0816 (rev 1).
      IRQ 2.
      Non-prefetchable 32 bit memory at 0x18005000 [0x18005fff].
  Bus  0, device   6, function  0:
    Class 0500: PCI device 14e4:080f (rev 1).
      IRQ 3.
      Non-prefetchable 32 bit memory at 0x18006000 [0x18006fff].
      Non-prefetchable 32 bit memory at 0x0 [0x7fffffff].
      Non-prefetchable 32 bit memory at 0x10000000 [0x17ffffff].
      Non-prefetchable 32 bit memory at 0x80000000 [0x9fffffff].
  Bus  0, device   7, function  0:
    Class 0604: PCI device 14e4:0804 (rev 1).
      IRQ 2.
      Non-prefetchable 32 bit memory at 0x18007000 [0x18007fff].
      Non-prefetchable 32 bit memory at 0x8000000 [0xffffffff].
  Bus  1, device   0, function  0:
    Class 0600: PCI device 14e4:4712 (rev 1).
      IRQ 2.
      Non-prefetchable 32 bit memory at 0x40000000 [0x40001fff].
      Prefetchable 32 bit memory at 0x0 [0x7fffffff].
  Bus  1, device   1, function  0:
    Class 0600: PCI device 14e4:4712 (rev 1).
      IRQ 2.
      Non-prefetchable 32 bit memory at 0x40002000 [0x40003fff].
      Prefetchable 32 bit memory at 0x48000000 [0x4fffffff].
  Bus  1, device   2, function  0:
    Class 0600: PCI device 14e4:4712 (rev 1).
      IRQ 2.
      Non-prefetchable 32 bit memory at 0x50000000 [0x50001fff].
      Prefetchable 32 bit memory at 0x58000000 [0x5fffffff].
  Bus  1, device   3, function  0:
    Class 0600: PCI device 14e4:4712 (rev 1).
      IRQ 2.
      Non-prefetchable 32 bit memory at 0x60000000 [0x60001fff].
      Prefetchable 32 bit memory at 0x68000000 [0x6fffffff].
  Bus  1, device   4, function  0:
    Class 0600: PCI device 14e4:4712 (rev 1).
```

```
      IRQ 2.
      Non-prefetchable 32 bit memory at 0x70000000 [0x70001fff].
      Prefetchable 32 bit memory at 0x78000000 [0x7fffffff].
  Bus  1, device   5, function  0:
    Class 0600: PCI device 14e4:4712 (rev 1).
      IRQ 2.
      Non-prefetchable 32 bit memory at 0x80000000 [0x80001fff].
      Prefetchable 32 bit memory at 0x88000000 [0x8fffffff].
  Bus  1, device   6, function  0:
    Class 0600: PCI device 14e4:4712 (rev 1).
      IRQ 2.
      Non-prefetchable 32 bit memory at 0x90000000 [0x90001fff].
      Prefetchable 32 bit memory at 0x98000000 [0x9fffffff].
  Bus  1, device   7, function  0:
    Class 0600: PCI device 14e4:4712 (rev 1).
      IRQ 2.
      Non-prefetchable 32 bit memory at 0xa0000000 [0xa0001fff].
      Prefetchable 32 bit memory at 0xa8000000 [0xafffffff].
  Bus  1, device   8, function  0:
    Class 0600: PCI device 14e4:4712 (rev 1).
      IRQ 2.
      Non-prefetchable 32 bit memory at 0xb0000000 [0xb0001fff].
      Prefetchable 32 bit memory at 0xb8000000 [0xbfffffff].
  Bus  1, device   9, function  0:
    Class 0600: PCI device 14e4:4712 (rev 1).
      IRQ 2.
      Non-prefetchable 32 bit memory at 0xc0000000 [0xc0001fff].
      Prefetchable 32 bit memory at 0xc8000000 [0xcfffffff].
  Bus  1, device  10, function  0:
    Class 0600: PCI device 14e4:4712 (rev 1).
      IRQ 2.
      Non-prefetchable 32 bit memory at 0xd0000000 [0xd0001fff].
      Prefetchable 32 bit memory at 0xd8000000 [0xdfffffff].
  Bus  1, device  11, function  0:
    Class 0600: PCI device 14e4:4712 (rev 1).
      IRQ 2.
      Non-prefetchable 32 bit memory at 0xe0000000 [0xe0001fff].
      Prefetchable 32 bit memory at 0xe8000000 [0xefffffff].
  Bus  1, device  12, function  0:
    Class 0600: PCI device 14e4:4712 (rev 1).
      IRQ 2.
      Non-prefetchable 32 bit memory at 0xf0000000 [0xf0001fff].
      Prefetchable 32 bit memory at 0xf8000000 [0xffffffff].
  Bus  1, device  13, function  0:
    Class 0600: PCI device 14e4:4712 (rev 1).
      IRQ 2.
      Non-prefetchable 32 bit memory at 0x0 [0x1fff].
      Prefetchable 32 bit memory at 0x8000000 [0xffffffff].
  Bus  1, device  14, function  0:
    Class 0600: PCI device 14e4:4712 (rev 1).
      IRQ 2.
      Non-prefetchable 32 bit memory at 0x10000000 [0x10001fff].
      Prefetchable 32 bit memory at 0x18000000 [0x1fffffff].
  Bus  1, device  15, function  0:
    Class 0600: PCI device 14e4:4712 (rev 1).
      IRQ 2.
      Non-prefetchable 32 bit memory at 0x20000000 [0x20001fff].
      Prefetchable 32 bit memory at 0x28000000 [0x2fffffff].
```

7 References

7.1 Forums / Wikis / Home Pages

7.1.1 broadband >> Forums >> Belkin >> Belkin Firmware Image Header
http://www.dslreports.com/forum/remark,10837191~mode=flat
hwa, September 2004

7.1.2 broadband >> Forums >> Belkin >> Belkin firmware size limit?
http://www.dslreports.com/forum/remark,10887965
yoyo18, August 2004

7.1.3 Belkin F5D7230-4 - SeattleWireless
http://gir.seattlewireless.net/index.cgi/Belkin_20F5D7230_2d4
Wiki, July 2004

7.1.4 I-Appliance BBS – Linksys WRT54G, etc
http://www.linux-hacker.net/cgi-bin/UltraBoard/UltraBoard.pl?Action=ShowPost&Board=RG&Post=1
Linuxguru, June 2004.

7.1.5 I-Appliance BBS – Belkin F5D7230 Wireless 802.11 b/g Router
http://www.linux-hacker.net/cgi-bin/UltraBoard/UltraBoard.pl?Action=ShowPost&Board=RG&Post=4
hardware1, February 2004.

7.1.6 Belkin F5D7230-4 Circuit Board and Info
http://www.linux-hacker.net/misc/F5D7230/
hardware1.

7.2 Distributions

7.2.1 OpenWRT
http://openwrt.org/

7.2.2 Belkin Firmware
http://belkin.com/support/download/download.asp?download=F5D7230-4

7.3 Articles

7.3.1 The Little Engine That Could
http://www.pbs.org/cringely/pulpit/pulpit20040527.html
R. X. Cringely, May 2004.

7.4 Miscellaneous

7.4.1 PuTTY: A free telnet/ssh client
http://www.chiark.greenend.org.uk/~sgtatham/putty/

7.4.2 Digitally Imported Radio
http://di.fm/

7.4.3 Know Your Enemy: Statistics
http://project.honeynet.org/papers/stats/
HoneyNet Project, July 2001.

8 Contact

8.1 Additions, Modifications and Deletions

For changes to this document, please refer to the author and revision history blocks in the control page. Please report errors or omissions to the author.

8.2 Consultation

If you would like to discuss wireless router firmware or other concepts related to this paper, then please contact the author;

Ian Latter

Late night coder …

MidnightCode.org

Email: ian dot latter at midnightcode dot org

Subject: OpenWRT on the Belkin F5D7230-4