Modification d'un firmware pour le DG834G --Troisieme partie--

publié par malphx le dimanche, novembre 11 2007 - 23:17

ATTENTION
Cette manipulation est de loin la plus dangereuse pour votre routeur.
Si vous n'avez jamais customisé/recompilé de noyau linux,
alors je vous déconseille fortement de vous lancer directement sur la recompilation du kernel pour le DG834G.
Pour les autres, dans la plupart des cas, la modification du kernel n'est pas nécessaire.
Elle n'est nécessaire que pour l'ajout d'une fonctionnalité nécessitant un support au niveau du noyau.

Préparation des sources

Renommage du repertoire sourceLors de la compilation du noyau le nom d'origine (**DG834(G).V2.10.17_src**) pose problème à cause des parenthèses.

1. On peut le renommer par exemple en V2.10.17_src, ou tout autre nom ne comportant pas de caractères spéciaux.`mv DG834(G).V2.10.17_src V2.10.17_src`

Application du patch patch-knlDepuis le repertoire source du firmware (maintenant V2.10.17) , les sources du noyau se trouvant dans le repertoire **linux-2.4.17_mvl21**

```
      [root@evectra V2.10.17_src]# lsapps DG834B_V2.10.17.img
linux-2.4.17_mvl21patch-apps README target.tar.bz2 uClibc-0.9.19
build.sh fs.bin MonFirmwareCustom.img patch-knl target tools
[root@evectra V2.10.17_src]# patch -p0 sortie
tronquée*****************patching file
linux-2.4.17_mvl21/net/Makefile.ipsecmd5patching file
linux-2.4.17_mvl21/net/Makefile.preipsecpatching file
linux-2.4.17_mvl21/net/Makefile.wipsecpatching file
linux-2.4.17_mvl21/net/atm/Makefilepatching file
linux-2.4.17_mvl21/net/atm/br2684.cpatching file
linux-2.4.17_mvl21/net/atm/common.cpatching file
linux-2.4.17_mvl21/net/atm/pppoatm.cpatching file
linux-2.4.17_mvl21/net/atm/pppoatm.c.3.5patching file
linux-2.4.17_mvl21/net/atm/pvc.cpatching file
linux-2.4.17_mvl21/net/ax25/ax25_ds_in.cpatching file
linux-2.4.17_mvl21/net/ax25/ax25_ds_subr.cpatching file
linux-2.4.17_mvl21/net/bluetooth/af_bluetooth.cpatching file
linux-2.4.17_mvl21/net/bluetooth/hci_core.c***********
```

Le patch modifie les sources du noyau original de montavista, il ajoute le support des éléments spécifiques à la plateforme du routeur.
Point important, le patch créé aussi le fichier **.config** qui contient toute la configuration pour la compilation par défaut.
En clair, on peut dire qu'après l'application du patch, le noyau serait prêt à être compilé.

Modification/personalisation

Que vous dire ici, sinon que cela se passe exactement comme pour un noyau linux :D.
Mais à moins de savoir exactement ce que vous faites, je vous déconseille de modifier le paramètrage d'origine… (retrait d'option / modification de valeur pour la flash…)

Que faire alors ?A vous de voir pourquoi vous voulez recompiler ce noyau ;)

Quelques idées:Ajout de fonction au niveau ipsec, ajout de modules iptables pour profiter de toute la puissance de ce parefeu, etc…

Compilation

Vos modifications sont faites ?
il ne reste plus qu'à compiler et générer une image intégrable dans le firmware.
Pour cette partie, vous aurez besoin de l'utilitaire dgfirmware qui va permettre de fusionner le noyau avec l'image du firmware.
De plus, j'ai utilisé une toolchain différente: mips-fp-le, issue du previewkit de montavista.
Malheureusement, je n'ai plus le lien :(, mais comptez sur moi pour le mettre en ligne dès que possible.

Les étapes de la compilation:

- make dep
- make ram_zimage_pad

Voila, on peut trouver maintenant dans le repertoire des sources du noyau un fichier nommé **ram_zimage_pad.bin**.
C'est ce fichier que nous allons fusionner dans l'image du firmware, et ce, à l'aide de **dgfirmware**
Voici la syntaxe de cet utilitaire:

```
[root@evectra DG834G.V2.10.09_src]# ./dgfirmware -husage:
dgfirmware [|opts|] |img| |img| firmware image filename |opts|
-h print this message -f fix the checksum -x |file| extract the
rootfs file to |file| -xk |file| extract the kernel to |file|
-m |file| merge in rootfs filrom |file| -k |file| merge in
kernel from |file| -w |file| write back the modified firmware
```

La commande sera donc de la forme: **./dgfirmware -f -k ram_zimage_pad.bin -w MonfirmwareCustom.img DG834G_V2.10.17.img**
Ou:

-f => fixe le checksum de la nouvelle image (calcul du CRC)
-k ram_zimage_pad.bin => image du noyau à fusionner
-w MonfirmwareCustom.img => Nom de la nouvelle image intégrant notre kernel
DG834G_V2.10.17.img => Image du firmware original.

```
[root@evectra DG834G.V2.10.09_src]# ./dgfirmware -f -k
ram_zimage_pad.bin -w MonfirmwareCustom.img DG834G_V2.10.17.img
Read firmware file
Firmware product: DG834
Firmware version: 1.42.09
Read kernel file
image checksum = 1f86
real checksum = ab3c
Bad Checksum, fix it
Write image file
```

Bien que la sortie indique un **Bad Checksum**, dgfirmware a bien corrigé le checksum.Pour en être sur:

```
[root@evectra DG834G.V2.10.09_src]# ./dgfirmware -f
MonfirmwareCustom.img
Read firmware file
Firmware product: DG834
```

```
    Firmware version: 1.42.09
image checksum = ab3c
real checksum = ab3c
Checksum is correct, good
```

Ben on dirait que ça y est :D
Il ne reste plus qu'à tester en flashant le routeur.

Mais n'oublie pas que l'utilitaire de recovery est ton ami :)

Modification d"un firmware pour le DG834G --Deuxieme partie--

publié par malphx le dimanche, novembre 11 2007 - 23:04

Pour plus d'info sur le Wake on Lan, voici un mini-howto.

On y est ! Notre machine Linux est prête, on peut commencer la modif.

Inspection rapide du filesystem 'target'le voici:

```
[root@evectra target]# ls -l
total 44
drwxr-xr-x 2 root root 4096 jui 2 2004 bin
drwxr-xr-x 2 root root 4096 mai 20 2001 dev
lrwxrwxrwx 1 root root 8 mai 25 23:45 etc -> /tmp/etc
drwxr-xr-x 3 root root 4096 jui 2 2004 lib
drwxr-xr-x 2 root root 4096 nov 13 2000 proc
drwxr-xr-x 2 root root 4096 jui 2 2004 sbin
drwxr-xr-x 2 root root 4096 jui 28 2000 tmp
drwxr-xr-x 8 root root 4096 mar 12 2004 usr
lrwxrwxrwx 1 root root 8 mai 25 23:45 var -> /tmp/var
lrwxrwxrwx 1 root root 8 mai 25 23:45 www -> /tmp/www
drwxr-xr-x 2 root root 4096 jun 3 2004 www.deu
drwxr-xr-x 2 root root 4096 jui 1 2004 www.eng
drwxr-xr-x 2 root root 4096 mai 13 2004 www.fre
drwxr-xr-x 2 root root 4096 jun 3 2004 www.ita
```

On retrouve la structure familière de ce bon vieux pingouin…
A noter tout de même, les liens symboliques **etc**,**var** et **www**.
Comme indiqué dans un précédent article: après le boot du DG, seul **/tmp** sera en lecture/ecriture puisque situé en RAM.
Le reste des repertoires seront en lecture seule.

Compilation de notre utilitaire

L'utilitaire sera donc **ether-wake v1.09** de Donald Becker (ether-wake.c: v1.09 11/12/2003 Donald Becker,http://www.scyld.com/).
Les sources sont disponibles ici.
Note: Les sources ont été légèrement modifiées pour pouvoir être linké avec ucLibc. Enf effet, le programme original faisait appel à des fonctions non implémentées dans la librairie C du DG834(G).
Les fonctions en question ont simplement été commentées. L'utilitaire final dans son utilisation sur le DG834(G) ne souffre pas de ces modifications.

La compilation proprement dite se passe très simplement:

```
[root@evectra WOL]# mipsel-uclinux-gcc -Wall -o ether-wake
ether-wake.c
[root@evectra WOL]# ls -l
total 36
-rwxr-xr-x 1 root root 21283 mai 26 13:22 ether-wake
-rw-r–r– 1 franck franck 11113 mar 29 22:00 ether-wake.c
[root@evectra WOL]#
```

Après avoir saisi la commande **mipsel-uclinux-gcc -Wall -o ether-wake ether-wake.c**, on obtient un executable MIPS:

```
[root@evectra WOL]# file ether-wake
ether-wake: ELF 32-bit LSB MIPS-I executable, MIPS, version 1
(SYSV), dynamically linked (uses shared libs), not stripped
```

On remarquera qu'il est linké dynamiquement à ucLibc.

```
[root@evectra WOL]# mipsel-uclinux-ldd ether-wake
libc.so.0 => /opt/brcm/hndtools-mipsel-uclibc-0.9.19/lib/libc.so.0
(0x00000000)
/lib/ld-uClibc.so.0 => /lib/ld-uClibc.so.0 (0x00000000)
```

Il ne nous reste plus qu'a stripper notre binaire pour gagner quelques octets:

```
[root@evectra WOL]# mipsel-uclinux-strip ether-wake
[root@evectra WOL]# ls -l ether-wake
-rwxr-xr-x 1 root root 15048 mai 26 13:26 ether-wake
```

Et voila, notre utilitaire est prêt à l'intégration dans le firmware.

Intégration au firmware

Le plus simplement du monde, on copie le binaire ether-wake dans un repertoire de notre filesystem ‘target’.
/usr/sbin par exemple.
Notre filesystem target est modifié, il ne nous reste plus qu’a construire l’image de notre firmware custom.

Reconstruction de l’image du firmware

Netgear, nous fournit tous les outils permettant cette reconstruction.
Voici comment les utiliser pour construire notre image.
Depuis la racine du repertoire créé lors de l’extraction des sources du firmware:

```
[root@evectra V2.10.17_src]# lsapps build.sh DG834B_V2.10.17.img
linux-2.4.17_mvl21 patch-apps patch-knl README target
target.tar.bz2 tools uClibc-0.9.19
```

On va utiliser le script **build.sh**, dont la syntaxe est la suivante:
build.sh [Nom du firmware source] |[Nom du FS target] [Nom du firmware destination]
Dans notre cas:

```
[root@evectra V2.10.17_src]# ./build.sh DG834B_V2.10.17.img target
MonFirmwareCustom.imgOriginal
Image: DG834B_V2.10.17.img
Your Filesystem: target
New Image: MonFirmwareCustom.img
Press ‘y’ to continue
```

On presse ‘y’, c’est parti:
Note: suivant la puissance de votre machine, cette étape peut être un peu longue.

```
Creating little endian filesystem on fs.bin,
block size 32768.
Little endian filesystem,
data block size 32768,
compressed data,
compressed metadata,
compressed fragments
Filesystem size 1697.66 Kbytes (1.66 Mbytes)
26.56% of uncompressed filesystem size (6391.35 Kbytes)
Inode table size 3147 bytes (3.07 Kbytes)
35.03% of uncompressed inode table size (8985 bytes)
Directory table size 2651 bytes (2.59 Kbytes)
55.19% of uncompressed directory table size (4803 bytes)
```

```
Number of duplicate files found 1Number of inodes 393
Number of files 267
Number of fragments 39
Number of symbolic links 67
Number of device nodes 32
Number of fifo nodes 0
Number of socket nodes 0
Number of directories 27
Number of uids 1 unknown (0)
Number of gids 0
MonFirmwareCustom.img Created!
```

Et voila, notre nouveau firmware est créé dans le repertoire courant

```
[root@evectra V2.10.17_src]# lsapps DG834B_V2.10.17.img
linux-2.4.17_mvl21 patch-apps README target.tar.bz2
uClibc-0.9.19build.sh fs.bin MonFirmwareCustom.img patch-knl target
tools
```

Il ne reste plus qu'a l'uploader sur notre routeur par la procédure habituelle. Ensuite nous pourrons utiliser notre nouvel utilitaire depuis une connexion telnet sur le routeur.

ATTENTION
Toute modification du firmware est potentiellement dangereuse.
Vous pouvez, en cas de mauvaise manipulation, perdre l'usage de votre routeur.
En cas d'utilisation d'un firmware custom, Netgear ne supportera plus votre matériel.
Vous êtes avertis.

Nous venons de voir le principe de modification du filesystem du DG834(G).
Avec cet exemple, certes simple, vous avez les bases pour l'ajout de fonctionnalité sur le routeur. Bien sur, le programme ether-wake ajouté seul, ne présente pas beaucoup d'intérrêt.
Pour finaliser, une modification des menus HTML du routeur, ainsi que le developpement d'un programme CGI sont nécéssaire pour une pleine intégration de la nouvelle fonctionnalité WoL dans l'interface d'administration du routeur.
A vous de jouer !

Un dernier conseil: si vous souhaitez vous lancer dans la création d'un firmware custom, la première chose à implémenter, c'est une busybox avec l'applet 'wget', ainsi vous pourrez tester vos nouveaux binaires en les téléchargent directement depuis le DG, dans son repertoire /tmp.
Vous évitant ainsi de reflasher à chaque essai ;).

Dans la 3eme partie de ce mini tuto, j'aborderai la recompilation du noyau linux.

@+',

Modification d"un firmware pour le DG834G --Premiere partie--

publié par malphx le dimanche, novembre 11 2007 - 22:52

Matériel requis

- Une machine Linux

Software requis

Une toolchain Linux MIPS Pour ma part, jchr(146)'utilise la toolchain fournie par Linksys pour son routeur WRT54g,que vous pourrez trouver sur le site de Linksys, partie GPL code.

Les sources GPL du firmwardisponible ici

Lchr(146)'utilitaire dgfirmware (créé par les membres du projet OpenDG834) Disponible ici en version source.

Installation de la toolchain

Pourquoi une toolchain MIPS ?

Cchr(146)'est très simple, le DG834G est architecturé autour dchr(146)'un System On Chip (SoC) de la famille AR7 de Texas Instrument.
Le processeur de ce SoC est un MIPS 4KEc cadencé à 150Mhz.Il nous faut donc une toolchain permettant de générer des binaires dans un format reconnu par ce processeur.
Nous allons grâce à cette toolchain, cross-compiler pour le DG834G. C'est à dire compiler des sources sur un type d'architecture machine (Intel x86 pour la plupart je pense), mais générer en sortie des executables pour architecture différente, en l'occurence Mips LE. (LE pour Little Endian).

Pour la toolchain, deux possibilités:

1. Utiliser une toolchain déjà construite :D
2. Eh construire la sienne

Vous trouverez plus bas une description de lchr(146)'installation dchr(146)'une toolchain prête à l'emploi, puisque cchr(146)'est le choix que j'ai fais.

IMPORTANT: Cet article ne décrit pas la construction d'une toolchain MIPS.
Si vous cherchez des informations sur la construction => Google est votre ami !

Note: la description qui suit est faite à partir de lchr(146)'archive wrt54g.2.02.7.tgz provenant du site Linksys (lien plus haut) .

En avant pour lchr(146)'install !Une fois le tarball téléchargé:

DecompressionOn obtient le repertoire **WRT54G**

Copie de la toolchain sous **/opt**Elle se trouve dans le repertoire **WRT54G/tools/brcm**il faut copier **brcm** vers /opt

Modification de la variable $PATHIl faut ajouter les chemins: **/opt/brcm/hndtools-mipsel-linux/bin** et **/opt/brcm/hndtools-mipsel-uclibc/bin** à la variable **PATH**.

Création de liens symboliquesOn va créer des liens symboliques pour que les fichiers contenus sous /opt/brcm/hndtools-mipsel-uclibc/bin soit de la forme : **mipsel-uclinux-***
Pour cela, il faut depuis le repertoire /opt/brcm/hndtools-mipsel-uclibc/bin, lancer cette commande:

• `for i in ''ls''; do ln -s $i ${i/uclibc/uclinux}; done`Le résultat final pour ce repertoire:`[root@evectra bin]# lsmipsel-uclibc-addr2line mipsel-uclibc-g++ mipsel-uclibc-objcopy mipsel-uclinux-addr2line mipsel-uclinux-g++ mipsel-uclinux-objcopymipsel-uclibc-ar mipsel-uclibc-gasp mipsel-uclibc-objdump mipsel-uclinux-ar mipsel-uclinux-gasp mipsel-uclinux-objdumpmipsel-uclibc-as mipsel-uclibc-gcc mipsel-uclibc-ranlib mipsel-uclinux-as mipsel-uclinux-gcc mipsel-uclinux-ranlibmipsel-uclibc-c++ mipsel-uclibc-ld mipsel-uclibc-size mipsel-uclinux-c++ mipsel-uclinux-ld mipsel-uclinux-sizemipsel-uclibc-cc mipsel-uclibc-ldd mipsel-uclibc-strings mipsel-uclinux-cc mipsel-uclinux-ldd mipsel-uclinux-stringsmipsel-uclibc-cpp mipsel-uclibc-nm mipsel-uclibc-strip mipsel-uclinux-cpp`
Ainsi la toolchain est prête à l'emploi .
Les sources du firmware Une fois téléchargée et décompressée, lchr(146)'archive nous livre ses secrets:

```
[root@evectra V2.10.17_src]# lsapps build.sh
DG834B_V2.10.17.img linux-2.4.17_mvl21 patch-apps patch-knl
README target.tar.bz2 tools uClibc-0.9.19
```

Petite description de lchr(146)'ensemble:
apps: repertoire contenant les diverses applications opensource utilisée dans le firmware du DG.
build.sh: script permettant la reconstruction du firmware.
DG834B_V2.10.17.img: Image binaire du firmware (identique à celle que vous telecharger sur le routeur pour faire une mise à jour).
linux-2.4.17_mvl21: repertoire contenant les sources du noyau Linux.
patch-apps et **patch-knl**: respectivement, un patch à appliquer sur les sources des applications et un patch à appliquer sur les sources du noyau.
target.tar.bz2: Dans cette archive se trouve le **root fs**, une fois decompressé un repertoire **target**.
tools: repertoire contenant les outils permettant la création de l'image binaire:

- 7zip: Utilitaire de compression, utilisé au moment de la compilation du kernel (compression lzma).
- lzma: Utilitaire de compression, utilisé lors de la compression du filesystem SquashFS.
- mksquashfs: Utilitaire permettant la création de l'image binaire, remplace dans l'image du firmware la partie rootfs et y ajoutant le nouveau filesystem préalablement transformé en SquashFS compressé.
- sqsh: Utilitaire permettant la transformation du filesystem target en SquashFS 2.

uClibc-0.9.19: repertoire contenant la librairie C (version source).

Note importante
A ma connaissance, seule les sources du firmware 2.10.17 fournissent tous les utilitaires permettant de modifier nchr(146)'importe quelle partie du firmware (Kernel et rootfs).
Dans les versions précédentes, lchr(146)'utilitaire 7zip permettant la compression du kernel nchr(146)'était pas fournit, et il semble que ce soit une version exotique de ce dernier.
En effet, le 7zip officiel ne permet pas dchr(146)'obtenir un kernel valide.

Pour en terminer avec la préparation de lchr(146)'environnement de travail, il ne reste plus qu'à decompresser lchr(146)'archive **target.tar.bz2**

```
tar jxvf target.tar.bz2
```

On obtient ainsi le repertoire target, qui va nous permettre d'ajouter/retirer ce que nous voulons au rootfs.
Voila, cchr(146)'est la fin de ce premier article.
Notre machine est prête pour commencer le travail.
Dans la seconde partie, je traiterai un exemple de modification puis de reconstruction de l'image du firmware en vue de son téléchargement sur ce bon vieux routeur. ;)
@+