EIN BLICK IN DAS INNERE DER FRITZ!BOX

Boxenstop

Linux-Rechner erobern die Haushalte: In Form von Firmware kleiner so genannter Appliances nistet sich der Pinguin in staubige Ecken ein. Ein solches Linux-Gerät ist AVMs FRITZ!Box. An ihr zeigen wir, wie sich das eingebettete Linux vom Desktop-System unterscheidet und wo der Bastler für Modifikationen ansetzen kann.

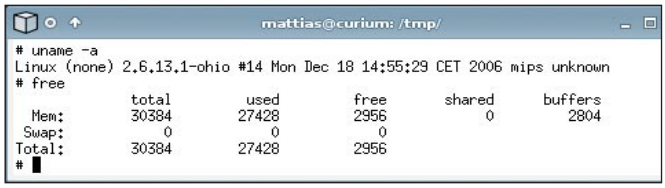
VON **MATTIAS SCHLENKER**

Oft ist nicht einmal bekannt, dass es sich bei den vielen dedizierten Netzwerkgeräten im Technikerhaushalt um einen Linux-Rechner handeln könnte. Viele Hardware-Hersteller handeln zwar GPL-konform und zeigen im Handbuch oder auf ihrer Webseite Möglichkeiten zum Download der GPL-Komponenten, doch kaum einer hängt den Aufabu an die große Glocke. Dass AVMs FRITZ!Boxen nicht nur auf Linux basieren, sondern auch einen Systemaufbau verwenden, der kleinere Erweiterungen oder die Installation einer eigenen Firmware zulässt, kann als Glücksfall für die Community gedeutet werden.

Bevor Sie sich jetzt ans Tuning Ihrer FRITZ!Box machen, **noch eine kleine Warnung**: Einige der vorgestellten Modifikationen bergen das Risiko, Ihre FRITZ!Box unbrauchbar zu machen. Zwar ist in den meisten Fällen das Zurücksetzen auf die Werkseinstellungen möglich, doch dafür ist ein Internet-Zugang sinnvoll. Sollten Sie Modifikationen planen, die jenseits der Möglichkeiten der *debug.cfg* liegen, ist es sinnvoll, einen fertig konfigurierten zweiten DSL-Router bereitzuhalten.

Erstkontakt

Kaum eine Appliance bietet bereits ab Werk einen aktivierten Telnet- oder SSH-Daemon für die Anmeldung an den Boxen an. Meist ist zwar einer installiert, dessen Start aber nur über Umwege möglich. Alle uns bekannten Firmware-Revisionen der FRITZ!Box 7170 und die meisten der 7050 verfügen über einen Telnet-Daemon, der sich mit der magischen Telefonnummer
#96*7*
aktivieren lässt. Anschließend sollten Sie sich via Telnet ohne Passwort an der Box einloggen können. Klappt das nicht, müssen schwere Geschütze aufgefahren werden. Der Autor dieser Zeilen hat ein Pseudo-Firmware-Update erstellt, das einen statisch kompilierten SSH-Daemon enthält und ein kleines Script, das zuerst nach dem Telnet-Daemon sucht und diesen startet - mehr Infos dazu weiter unten. Fehlt der Telnet-Daemon, startet das Skript stattdessen den SSH-Daemon. In diesem Fall ist das SSH-Login auf Port *2222* nötig, Nutzername und Passwort lauten



Nach dem Log-in findet man sich in einer normalen Linux-Shell wieder. Diese FRITZ!Box hat einen MIPS-Prozessor und 32 MByte RAM.

Weblinks

www.ip-phone-forum.de/forumdisplay.php?f=525
Viel für die Verbreitung der FRITZ!Box hat *danisahne* getan, dessen Scriptsammlung eine modifizierte Firmware erzeugt, aber auch dazu genutzt werden kann, selbst Programme für die Box zu erstellen.

www.rahul.net/dholmes/ctorrent/#download
Ein kompakter Bittorrent-Client für die Kommandozeile ist *cttorrent-dnh*

www.busybox.net
Das Multicall-Binary *Busybox* kommt auch auf der FRITZ!Box zum Einsatz. Hier erhalten Sie die aktuellen Quellcodes.

www.uclibc.org
Die FRITZ!Box verwendet die C-Bibliothek *uClibc*. Fertige Scripte für den Bau eines Chroot-Images hält deren Projektseite bereit.

http://blog.rootserverexperiment.de/2007/01/27/fritzbox-rootserver
Auch der Autor setzt sich mit der Fritzbox auseinander, seine Erkenntnisse legt er in seinem Blog offen.

toro. Weil das Pseudo-Firmware-Update im flüchtigen Speicher entpackt wird und keine dauerhaften Änderungen an der FRITZ!Box vornimmt, ist das Risiko gering. Nichtsdestotrotz registriert die FRITZ!Box, dass ein nicht von AVM signiertes Image eingespielt wurde, und weist fortan im Webinterface darauf hin. Sollte sich nach dem Pseudo-Firmware-Update gar nichts tun, hilft es, die Box einige Sekunden von ihrer Stromversorgung zu trennen, um sie zu resetten. Während der mit dem Pseudo-Firmware-Update gestartete SSH- oder Telnet-Daemon aktiv ist, ist übrigens die Verbindung zur Außenwelt getrennt, möglicherweise für die FRITZ!Box benötigte Dateien sollten Sie vorher herunterladen.

Nun können Sie in der Shell einen Rundgang durch die Box antreten: Auf unserer 7170 mit Firmware *29.04.29* zeigte uns *uname -a* einen 2.6.13er-Kernel und bestätigte die MIPS-Plattform, *ls /lib* wies auf die kompakte C-Bibliothek *uClibc* in Version *0.9.28* hin. Die exakte Belegung der Dateisysteme war mangels *df* nicht zu ermitteln, dafür zeigte *cat /proc/mounts*, was wohin gemountet war und wies mit dem Typ *romfs* gleich darauf hin, dass einzig */var* schreibbar ist. Der Befehl *free* zeigte etwa 24 Mbyte belegten Speicher (bei laufendem SSH-Daemon) von insgesamt 30 MByte - nicht viel Platz für eigene Erweiterungen. Mangels *top* ist keine aktuelle Liste von Speicher- und Prozessorbedarf zu bekommen, immerhin zeigt *ps w* oder *ps waux* die aktiven Prozesse an. Wer noch einen Blick nach */bin* wirft, erkennt schnell, dass fast alle Befehle Softlinks auf die Busybox sind: Bei ihr handelt es sich um ein so genanntes Multicall-Binary, eine einzige ausführbare Datei, die viele Linux-Tools in einem Binary vereint. Einige wenige Programme von AVM für die Verwaltung der Box sind hier zu finden. Lieber nicht ausprobieren wollten wir *prepare_fwupgrade*, ganz lustig ist hingegen *update_led_on* das einfach die Update-LED blinken lässt (das Gegenstück zum Ausschalten heisst *update_led_off*).

Dauerhaft geändert

Eine ganz nette Hintertüre für kleine Modifikationen hat sich AVM mit der Datei */var/flash/debug.cfg* offen gelassen. Es ist die einzige aus dem Dateisystem heraus schreibbare Datei und wird beim Hochfahren der Box als Shellscript interpretiert.

Leider ist die Datei nicht direkt schreibfähig, sie muss am Stück mit „cat“ geschrieben werden, beispielsweise indem zunächst die Datei

Kritische Punkte an AVM und der FRITZ!Box

AVMs FRITZ!Boxen sind nicht aus Bastlerfreundlichkeit mit freier Software ausgestattet. Stattdessen verfolgt AVM handfeste wirtschaftliche Interessen, wozu vor allem günstige Verkaufspreise zählen. Die Detailwünsche einzelner Kundengruppen sind zweitrangig, wenn dafür die große Masse der Nutzer ein gut funktionierendes, günstiges Gerät bekommt. Auch sollten Sie sich vor Augen halten, dass die FRITZ!Box nicht so große Verbreitung erlangt hat, weil sie von den Bastlern geliebt wird, sondern umgekehrt erst die große Verbreitung der Box als kostenloser Bonus zum DSL-Vertrag viele Nutzer dazu animiert hat nachzuschauen, was drin steckt. Bevor Sie beginnen, Ihre gesamte Kommunikationsinfrastruktur um eine FRITZ!Box herum aufzubauen, beachten Sie bitte:

Modifikationen der Firmware: Firmware-Updates der FRITZ!Box können sicher geglaubte Funktionen deaktivieren oder gravierende Änderungen an der Verzeichnisstruktur zur Folge haben. Mühsam selbst installierte Software funktioniert dann gar nicht mehr oder nicht mehr zuverlässig. Zwar veröffentlich AVM mittlerweile Beta-Versionen von Firmware-Updates mit neuen Features, dennoch ist ungewiss, ob die finalen Versionen deren Aufbau beibehalten.

Eine maßgeschneiderte Architektur: AVM setzt in den von uns getesteten Versionen 3070, 7050 und 7170 einen MIPS-Prozessor ein, um den eine sehr spezielle Hardware-Umgebung aufgebaut wurde. Das ist bei großen Stückzahlen sinnvoll, weil es die Hardware-Kosten senkt, erfordert aber starke Anpassungen an der Software. Auf „Evaluation Boards“ von Prozessorherstellern zugeschnittene Software-Entwicklungsumgebungen wie die uClibc-Buildroot-Scripte funktionieren deshalb oft nicht.

USB 1.1: Die FRITZ!Box 3070 und 7170 verfügen lediglich über einen langsamen USB-1.1-Anschluss, erst die FRITZ!Box 7270 bringt USB 2.0 mit. USB 1.1 genügt für das Streaming von MP3-Dateien zu ein paar wenigen Clients, für einfache Datei- und Druckdienste oder den nächtlichen Download eines Linux-ISOs via Bittorrent. Viel mehr sollten Sie dem Port nicht zumuten: Die vielerorts zu findenden USB-Kraken, bei denen eine Box via USB-Hub zwei externe Festplatten und einen Drucker versorgt, lassen zudem durch den Einsatz vieler ineffizienter Netzteile Zweifel an der Energiebilanz einer solchen Lösung aufkommen.

Die Versionsvielfalt: FRITZ!Boxen gibt es mit und ohne USB, mit oder ohne WLAN, mit oder ohne VoIP-Funktionalität - in praktisch jeder Kombination dieser Merkmale. Jede dieser Boxen stellt eigene Anforderungen an den Kernel und die Systemprogramme, was in eigenen Firmware-Versionen und Detail-Unterschieden resultiert. Ein auf einer Box erstelltes Tutorial muss auf einer zweiten aus einer anderen Serie nicht unbedingt funktionieren.

Knapper Speicher: Nach unserem Kenntnisstand verfügen die FRITZ!Boxen derzeit zwischen 16 und 64 Megabyte RAM. Das war schon vor zehn Jahren für einen Linux-Server recht knapp. Der Arbeitsspeicher ist so bemessen, dass den standardmäßig installierten Anwendungen genug Luft bleibt. Jede zusätzlich gestartete Anwendung zehrt jedoch am Arbeitsspeicher. Zu viele zusätzliche Anwendungen sollten also nicht benutzt werden. Auch die Vergrößerung des virtuellen Speichers durch die Nutzung von Swap-Space auf einer USB-Festplatte ist angesichts der langsamen USB-1.1-Schnittstelle vieler Boxen nur bedingt sinnvoll.

Schwacher Prozessor: Die größten Rechenlasten der FRITZ!Box werden von maßgeschneiderten Signalprozessoren oder dem WLAN-Chip übernommen. Der Hauptprozessor kann sich um die Verwaltung des Systems kümmern. Mutet man diesem prozessorintensive Aufgaben wie die Prüfsummenerstellung bei der Datensicherung mit *rsync* oder der Codierung/Decodierung verschlüsselter Verbindungen (SSH) zu, schlägt sich dies schnell in der gesamten Geschwindigkeit und damit in der Effizienz des Routing nieder.

Echte Alternativen zur FRITZ!Box gibt es keine - sieht man von der deutlich teureren Horstbox von DLink ab - wohl aber sinnvolle Ergänzungen. So kann die FRITZ!Box beispielsweise als Drucker-Server dienen und ein SSH-Login zum Aufwecken Wake-on-LAN-kompatibler Rechner bereithalten, während statt zwei USB-Festplatten und USB-Hub (und deren drei Netzteile) besser ein kleines NAS oder ein selbstgezimmerter Server auf EPIA-Basis zum Einsatz kommt.

```
/var/tmp/debug.cfg
```

angepasst und dann mit ihr die */var/flash/debug.cfg* überschrieben wird:

```
cat /var/tmp/debug.cfg >
➥ /var/flash/debug.cfg
```

Bei Boxen, die einen Telnet-Daemon mitbringen, aktivieren Sie diesen beim Start mit:

```
if [ -f /usr/sbin/telnetd ]
then
        /usr/sbin/telnetd -l
➥ /sbin/ar7login
fi
```

Viele Programme sind lediglich Softlinks auf das Multicall Binary *bin/busybox*.

Cross-Compiling für die FRITZ!Box

Unter *cross-compiling* versteht man das Erstellen von Binärdateien für Zielsysteme, die über eine andere Prozessorarchitektur verfügen als das System, auf dem gebaut wird. Was zunächst nach dem einfachen Einsatz eines anderen Compilers klingt, zieht einen Rattenschwanz an Abhängigkeiten nach sich. So müssen neben einem Compiler, der für die andere Zielarchitektur übersetzt, passende *binutils* vorhanden sein. Sie erstellen aus einzelnen Objektdateien die funktionierenden Programme. Und auch die benötigten Bibliotheken dürfen nicht fehlen. Die Erstellung einer Cross-Compile-Toolchain ist meist eine relativ aufwändige Sache. Einfacher geht es, wenn man das *ds-mod*-Paket aus dem *IP Phone Forum* verwendet: Es erstellt mit dem Kommando *make toolchain* automatisch die von AVM verwendete Toolchain. Mit dieser können nun Programme für die FRITZ!Box kompiliert werden. Ausprobiert haben wir es am Beispiel des Bittorrent-Clients *ctorrent*:

Zunächst müssen die Tools für *mipsel* in den Suchpfad aufgenommen werden. Vorne im Pfad ist besser, unsere lagen unter */data/ds26-14.4*:

```
export PATH=/data2/ds26-
➥ 14.4/ toolchain/target/bin:$PATH
```

Der Aufruf von *./configure* benötigt weit mehr Parameter als bei übereinstimmendem Ziel- und Buildsystem, hier deaktivieren wir zudem SSL:

```
./configure –with-ssl=no
➥–target=mipsel-linux –host=mipsel-
➥linux –build=i386 CFLAGS="-g -O2
➥-march=4kc"
```

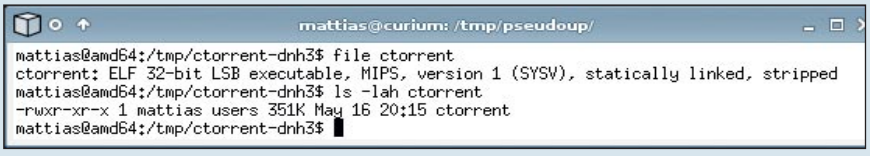
Das eigentliche Bauen soll in einem statischen Binary resultieren, dafür sind dem Linker Flags mitzugeben:

```
make LDFLAGS="-static"
```

Nun wird das resultierende Binary noch von Debug-Informationen befreit, was die Größe etwa halbiert:

```
mipsel-linux-strip ctorrent
```

Das fertige *ctorrent* konnten wir zur FRITZ!Box kopieren und dort starten. Beim Bauen mit einer Cross-Compile-Toolchain sollten Sie darauf achten, dass nur Bibliotheken verwendet werden, die in der Toolchain von *ds-mod* vorhanden sind, ansonsten kann die recht aufwändige Nachinstallation von Bibliotheken und Angabe zusätzlicher Include-Pfade erforderlich sein.



Mit der Toolchain *ds-mod* sind kleine Tools schnell für die FRITZ!Box kompiliert.

Fehlt der Telnet-Daemon, muss erst ein SSH-Daemon heruntergeladen und dieser dann gestartet werden. Das resultierende Script ist etwas aufwändiger, weil Berechtigungen gesetzt und Schlüssel erzeugt werden müssen. Anstatt den User *root* zu modifizieren, fügen wir einen zweiten User *toro* mit User-ID *0* an. Der hat die gleichen Rechte, ist aber schneller nachgerüstet. Den Passwort-Hash in der letzten Zeile haben wir übrigens mit dem Befehl *openssl passwd* erzeugt.

```
DROPBEAR="http://fbh.mattiasschlenker.
➥de/7170-29.04.29/dropbear-static"
sleep 45
wget -O /var/tmp/dropbear $DROPBEAR
chmod 0755 /var/tmp/dropbear
ln -sf /var/tmp/dropbear
➥/var/tmp/dropbearkey
ln -sf /var/tmp/dropbear
➥/var/tmp/dropbearclient
ln -sf /var/tmp/dropbear
➥/var/tmp/dbclient
ln -sf /var/tmp/dropbear /var/tmp/ssh
/var/tmp/dropbearkey -t rsa -f
/var/tmp/dbkey.rsa -s 1024
echo 'toro:x:0:0:toro:/:/bin/sh' >>
➥/etc/passwd
echo
'toro:bDAd.KLmrm3gk:12332:0:99999:7:::'
➥>> /etc/shadow
/var/tmp/dropbear -p 22 -r
➥/var/tmp/dbkey.rsa
```

Sollten Sie über eine der Boxen mit USB-Anschluss verfügen, können Sie den SSH-Daemon und die Schlüsseldatei auf einem USB-Stick oder der USB-Festplatte unterbringen, mit

```
if [ -f /pfad/zum/dbkey.rsa ]
then
   ...
```

das Vorhandensein überprüfen und die benötigten Dateien ins */var/tmp* der FRITZ!Box kopieren. Der Start geht dann schneller und Sie müssen nicht nach jedem Neustart eine neue Schlüsseldatei bestätigen.
Ein laufender SSH-Daemon genügt noch nicht für ein Login via Internet, da die Firewall der Box den Zugriff sperrt. Ein kleiner Trick hilft. Zunächst wird eine Alias-Ethernet-Adresse im gleichen Netz vergeben:

```
ifconfig lan:1 inet 192.168.1.251
➥netmask 255.255.255.0 broadcast
➥192.168.1.255
```

Anschließend müssen Sie im Webfrontend der Box per Portforwarding einen von außen erreichbaren Port auf den SSH-Port der

FRITZ!Box und die „neue“ IP-Adresse weiterleiten. Bei Firmware 29.04.29 war das Tool ether-wake mit an Bord, sodass es jetzt von außerhalb möglich ist, Rechner im Heimnetz aufzuwecken. Der geneigte Leser kann anschließend mit plink.exe und einem mit Puttygen erzeugten Schlüssel eine Batch-Datei auf dem Windows-Desktop des Arbeitsrechners ablegen, die auf Doppelklick den PC zu Hause hochfährt.

Transportfragen

Um Dateien auf die Box zu übertragen, stehen prinzipiell zwei Möglichkeiten zur Verfügung: Von der Box aus können Sie per *wget* oder *ssh* Dateien herunterladen. Insbesondere *wget* ist ideal, um während des Startups Dateien aus dem Netz auf die Box zu kopieren. Wenn Sie während Ihrer FRITZ!Box-Experimente hin und wieder einzelne Dateien zur Box kopieren oder von dieser holen müssen, genügt eine geschickte Pipeline. Eine auf dem Desktop erstellte *test.txt* landet mit dem folgenden Befehl im */var/tmp* der FRITZ!Box:

```
cat test.txt | ssh toro@192.168.1.252
➥ „cat > /var/tmp/test.txt“
```

Das Holen von Dateien geht ähnlich:

```
( ssh toro@192.168.1.252 „cat
➥ /var/tmp/test.txt“ ) > /tmp/test.txt
```

Sollten Sie häufiger Dateien zur FRITZ!Box kopieren oder von ihr holen, raten wir dazu, diese Befehle in ein Shellscript auszulagern.

Ein kompletter PC

Die seit etwa einem Jahr verfügbaren Boxen mit USB-Host laden dazu ein, große Datenmengen auf der Box zu speichern: MedienServer, Bittorrent-Client, sogar Webserver, alles schien möglich. Zunächst wollten wir ohne Rücksicht auf den Platz ausprobieren, wo die Grenzen liegen. Wir verwendeten ein Root-Dateisystem-Image von **www.uclibc.org**, erstellten eine statisch gelinkte Busybox für den MIPS-Prozessor und ein Ext2-Kernel-Modul für den damals aktuellen 2.4er-Kernel der FRITZ!Box. Mit dem Kernel-Modul konnten wir eine Ext2-Partition auf der USB-Festplatte mounten und eine Swap-Partition aktivieren, um schließlich mit dem *chroot* der statisch gelinkten Busybox in das auf der Ext2-Partition entpackte uClibc-Rootfile-System zu wechseln.

In diesem steht eine komplette Entwicklungsumgebung mit Compiler, Linker etc. zur Verfügung, so dass es uns gelang, *ctorrent*, den Samba-Server und einige weitere Tools zu kompilieren. *ctorrent* arbeitete mit einer Limitierung auf einige wenige parallele Upstreams dann auch zufriedenstellend. Stellte man den Befehl *nohup* voran, war es auch möglich, einen Download im Hintergrund fortzusetzen.

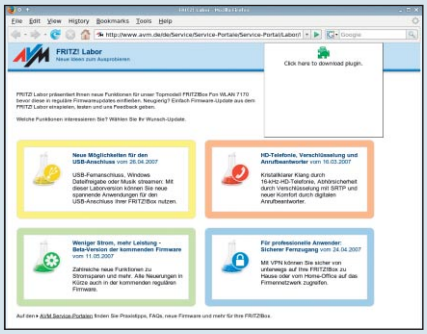
Leider bringen die auf Kernel 2.6 basierenden Firmware-Images zwar ein Kernel-Modul für Ext2- und Ext3-Partitionen mit, doch auf Grund einiger Inkompatibilitäten lassen sich einige Programme in der Chroot-Umgebung nicht mehr starten. Sollten Sie noch über eine FRITZ!Box 7170 mit 2.4er-Kernel basierter Firmware verfügen, können Sie das Chroot-Image auf der Webseite des Autors herunterladen.

Einen anderen Weg geht *Danisahne* mit seiner Scriptsammlung zur Erzeugung eines angepassten Firmware-Images: Auf Basis der Original-AVM-Toolchain gibt er dem Nutzer die Möglichkeit, en Détail zu bestimmen, welche Module seine Busybox verwenden soll und welche zusätzlichen Tools installiert werden sollen. Das Script kompiliert dann die benötigten Werkzeuge und entnimmt nicht frei verfügbare Werkzeuge einem Original-AVM-Firmware-Image. Aus den Komponenten setzt es ein neues Firmware-Image zusammen, mit dem Sie die FRITZ!Box nun flashen können.

Eine derartige Modifikation ist natürlich riskanter als ein paar in */var/flash/debug.cfg* nachgetragene Zeilen. **jkn**

Frisch aus der Hexenküche

Mit dem Aufkommen freier FRITZ!Box-Modifikationen musste auch AVM lernen, dass eine Pflege der vorhandenen Firmware nicht immer reicht, sondern viele Nutzer nach mehr gieren. AVM lässt deshalb den Endverbraucher Einblicke in den Entwicklungsprozess nehmen und bietet Beta- und Testversionen seiner Firmware zum Download an. Viele der angebotenen Features wie der Samba-Server für die Freigabe von angestöpselten USB-Laufwerken als Netzwerk-Shares oder der UPnP-Streaming-Server dienen aber eher zum Ausloten des Möglichen und müssen nicht ihren Weg in die endgültigen Firmware-Updates finden. Auch ist es nicht garantiert, dass die Funktionen einer Labor-Version auch in der nächsten zu finden sind.



AVMs Labor-Firmware verspricht einen relativ risikoarmen Einblick in die Firmware-Entwicklung.

Der Test ist relativ gefahrlos und richtet sich an Nutzer der „vollausgestatteten“ FRITZ!Box 7170. Falls Sie am Test interessiert und bereit sind, AVM Ihre Erfahrungen mitzuteilen, können Sie die Labor-Versionen unter dieser Adresse herunterladen:
www.avm.de/de/Service/Service-Portale/Service-Portal/Labor/labor.php

Kaputt gemacht?

Alleine durch die Veränderung der */var/flash/debug.cfg* können Sie wenig kaputtmachen. Das Risiko steigt jedoch, wenn Sie modifizierte Firmware-Images wie das *dsmod* einspielen. Haben Sie „nur“ irgendwelche Einstellungen zerschossen, kann ein Reset auf die Werkseinstellungen bereits genügen. Den erreichen Sie, indem Sie an einem angeschlossenen Analogtelefon die Tastenkombination
#99**
eingegeben. Schlägt das fehl, hilft nur das Aufspielen einer originalen Firmware. Wie viele andere dedizierte „Devices“ erlaubt die FRITZ!Box beim Start einige Sekunden lang den ungeschützten Zugriff auf ihr Innerstes, hier via FTP-Zugang. Den nutzen spezielle Recovery-Tools aus, die Kernel und Root-Filesystem zurückspielen. Eine detaillierte Anleitung (die leider so nur unter Windows umzusetzen ist) hält das „Router FAQ“ bereit: www.router-faq.de/index.php?id=fb&fb=recover

Das „IP Phone Forum“ gibt allgemeine Tipps:
www.ip-phone-forum.de/showthread.php?t=66924