

Firmware Analysis & Filesystem Extraction

 redfoxsec.com/blog/analyzing-firmware-and-extracting-filesystem

Kunal Kumar

June 4, 2024



Analyzing Firmware and Extracting Filesystem

- June 4, 2024
- Active Directory
- Kunal Kumar

Every modern device—whether it's a Wi-Fi router, smart TV, medical equipment, or industrial controller—relies on firmware. Firmware is the invisible layer of code that powers hardware, managing communication between components, processing data, and enabling functionality.

Unlike regular software, firmware runs deep inside devices, yet it's often overlooked during security assessments. This makes it a prime target for attackers. Hidden inside firmware images, you might find hardcoded passwords, outdated libraries, misconfigurations, or even backdoors. A successful firmware compromise can give adversaries persistent, stealthy access—often invisible to traditional security tools.

That's why learning to analyze and extract firmware is more than just reverse engineering—it's an essential skill for cybersecurity professionals. By inspecting firmware, you can uncover vulnerabilities before they're exploited, harden devices against attacks, and gain insights into how embedded systems operate.

This guide walks through the end-to-end process: acquiring firmware, extracting its filesystem, performing analysis, and interpreting findings. Whether you're a pentester, security researcher, or simply curious about how your devices work, this breakdown will help you explore the hidden world of firmware.

But here's the catch: firmware is often overlooked in security assessments, even though it's a prime target for attackers. Hidden within firmware images could be **hardcoded passwords, outdated libraries, misconfigurations, or even backdoors** that jeopardize an entire system. Once compromised, firmware can allow attackers to gain persistent, stealthy access—undetectable by traditional security measures.

That's why analyzing and extracting firmware isn't just an exercise in reverse engineering—it's a **critical skill for anyone serious about cybersecurity**. By peering inside, we can uncover vulnerabilities before adversaries exploit them, strengthen device security, and gain valuable insights into how embedded systems operate.

This guide walks you through the **end-to-end process of firmware analysis**, from acquisition to filesystem exploration, static and dynamic analysis, and finally, reporting findings. Whether you're a security researcher, pentester, or simply curious about what powers your devices, this step-by-step breakdown will show you how to crack open the black box of firmware and uncover its secrets.

Why Firmware Analysis Matters

Firmware analysis provides clear benefits for both individuals and organizations:

- **Discover vulnerabilities** like hardcoded credentials and outdated libraries.
- **Identify sensitive data exposures** that could compromise devices or user privacy.
- **Strengthen defenses** by detecting weaknesses before attackers exploit them.
- **Enable testing without hardware**, simulating penetration testing environments.
- **Gain operational insights** into device functionality at the lowest level.

In short, firmware analysis improves security posture and offers a cost-effective way to manage risks.

The Process of Firmware Extraction And Analysis

Step 1: Firmware Acquisition

The first step is obtaining a firmware image:

- Download directly from the device or the manufacturer's official site.
- Verify authenticity to ensure the image hasn't been tampered with.

Example:

For this walkthrough, we'll use `FW_WRT1900ACSV2_2.0.3.201002_prod`. Extracting it with Linux's `7z` tool yields raw firmware files for analysis.

```
$ git clone https://github.com/prokunal/Dumping-Router-Firmware-Image
```

Extracting the firmware image from the archive.

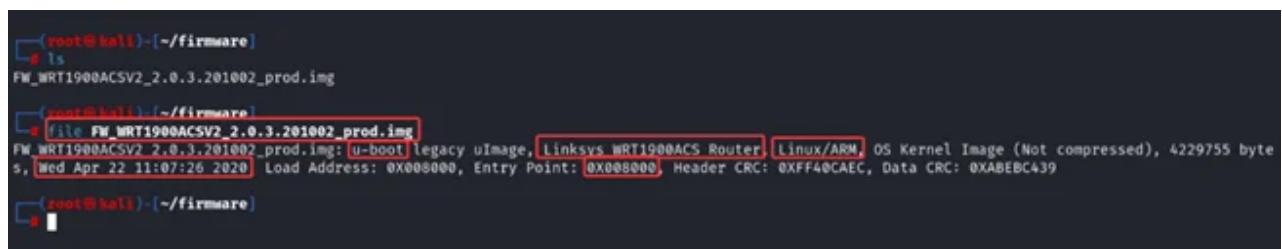
```
$ 7z x FW_WRT1900ACSV2_2.0.3.201002_prod.zip
```

Step 2: Firmware Extraction

With the image in hand, several tools help extract and analyze its content:

```
$ file FW_WRT1900ACSV2_2.0.3.201002_prod.img
```

file command – Provides metadata (architecture, compression type, creation date, entry point).



The terminal window shows the following session:

```
root@kali:~/firmware$ ls
FW_WRT1900ACSV2_2.0.3.201002_prod.img
root@kali:~/firmware$ file FW_WRT1900ACSV2_2.0.3.201002_prod.img
FW_WRT1900ACSV2_2.0.3.201002_prod.img: u-boot legacy image, Linksys WRT1900ACS Router, Linux/ARM OS Kernel Image (Not compressed), 4229755 bytes, Wed Apr 22 11:07:26 2020 Load Address: 0X008000, Entry Point: 0X008000, Header CRC: 0xFF40CAEC, Data CRC: 0xABEBC439
root@kali:~/firmware$
```

strings command – Reveals readable text within the image, often containing useful details.

```
$ strings FW_WRT1900ACSV2_2.0.3.201002_prod.img
```

From the above command, we can analyse the strings to find more details about the firmware.

```

root@kali:[~/firmware]
# strings FW_WRT1900ACSV2_2.0.3.201002_prod.img
Linksys WRT1900ACS Router
@ #
!1C "
-- System halted
Attempting division by 0!
Uncompressing Linux ...
decompressor returned an error
done, booting the kernel.
invalid distance too far back
invalid distance code
invalid literal/length code
incorrect header check
unknown compression method
invalid window size
invalid block type
invalid stored block lengths
too many length or distance symbols
invalid code lengths set
invalid bit length repeat
invalid literal/lengths set
invalid distances set
incorrect data check
Out of memory while allocating output buffer
Out of memory while allocating input buffer
Out of memory while allocating z_stream
Out of memory while allocating workspace
Not a gzip file
header error
read error
uncompression error
phandle
linux,phandle

```

binwalk – Identifies file signatures and extracts embedded data, such as compressed filesystems or hidden resources.

Binwalk is a versatile tool for identifying well-known file signatures within a given file. While its primary function is to analyse file structures, it can also be utilized in areas like Steganography. For instance, it can uncover hidden files within images.

In the context of router analysis, Binwalk proves invaluable for extracting the filesystem, allowing for deeper inspection and potential modification.

```
$ binwalk -e FW_WRT1900ACSV2_2.0.3.201002_prod.img --run-as=root
```

```

root@kali:[~/firmware]
# binwalk -e FW_WRT1900ACSV2_2.0.3.201002_prod.img --run-as=root
DECIMAL      HEXADECIMAL      DESCRIPTION
-----      -----      -----
0          0x0      uImage header, header size: 64 bytes, header CRC: 0xFF40CAEC, created: 2020-04-22 11:07:26, image size: 4229755 bytes, Data Address: 0x8000, Entry Point: 0x8000, data CRC: 0xABEBC439, OS: Linux, CPU: ARM, image type: OS Kernel Image, compression type: none, image name: "Linksys WRT1900ACS Router"
64          0x40      Linux Kernel ARM boot executable zImage (little-endian)
26736        0x6870      gzip compressed data, maximum compression, from Unix, last modified: 1970-01-01 00:00:00 (null date)
4214256      0x404DF0      Flattened device tree, size: 15563 bytes, version: 17
6291456      0x600000      JFFS2 filesystem, little endian

root@kali:[~/firmware]
# ls
FW_WRT1900ACSV2_2.0.3.201002_prod.img  FW_WRT1900ACSV2_2.0.3.201002_prod.img.extracted
[  ]
```

The above command output also gives basic information about the firmware and filesystem, such as image size, OS, CPU, product name, and compression type. The filesystem is JFFS2 in little-endian format.

Analysing the extracted file, found two files one is a jffs2 filesystem, and another one looks like a compressed file.

```

└─(root㉿kali)-[~/firmware]
└─# ls
FW_WRT1900ACSV2_2.0.3.201002_prod.img _FW_WRT1900ACSV2_2.0.3.201002_prod.img.extracted
└─(root㉿kali)-[~/firmware]
└─# cd _FW_WRT1900ACSV2_2.0.3.201002_prod.img.extracted
└─(root㉿kali)-[~/firmware/_FW_WRT1900ACSV2_2.0.3.201002_prod.img.extracted]
└─# ls
600000.jffs2 6870
└─(root㉿kali)-[~/firmware/_FW_WRT1900ACSV2_2.0.3.201002_prod.img.extracted]
└─# du -sh *
8.0M 6870
27M 600000.jffs2
└─(root㉿kali)-[~/firmware/_FW_WRT1900ACSV2_2.0.3.201002_prod.img.extracted]
└─# file 6870
6870: data
└─(root㉿kali)-[~/firmware/_FW_WRT1900ACSV2_2.0.3.201002_prod.img.extracted]
└─# 

```

Analysing the compressed file 6870 using strings to find more information about this file.

```
$ strings 6870
```

```

net/ipv6/ip6_offload.c
net/ipv6/inet6_hashtables.c
sk_RefCnt Type Proto Iface R Rmem User Inode
%pK %-6d %-4d %04x %-5d %1d %-6u %-6u %-6lu
&net->packet.sklist_lock
net/packet/af_packet.c
TPACKET version not supported.
3packet size is too long (%d > %d)
3packet size is too short (%d < %d)
3Packet exceed the number of skb frags(%lu)
&po->pg_vec_lock
Tx-ring is not supported.
3packet_mmap: vma is busy: %d
3Attempt to release alive packet socket: %p
TPACKET version not supported
3bridge: can't register sap for STP
net/bridge/br_fdb.c
4%: adding interface %s with same address as a received packet
bridge_fdb_cache
4%: received packet on %s with own address as source address
6bridge: RTM_NEIGH with invalid state %#
6bridge: RTM_NEIGH with invalid vlan
6bridge: RTM_NEIGH with invalid vlan id %d
6bridge: RTM_NEIGH %s not a bridge port
6bridge: RTM_NEIGH with unconfigured vlan %d on port %
6bridge: RTM_NEIGH %s not a bridge port
6bridge: RTM_DELNEIGH with unconfigured vlan %d on port %
plip
net/bridge/br_if.c
brport
failed insert local address bridge forwarding table
6%: port %u(%s) entered %s state
5%: root port %u not found for topology notice
6%: topology change detected, %
propagating
sending tcn bdu

```

From the above command, it didn't give much information about the file.

Extracting the content of 6870 file using binwalk.

```
$ binwalk -e 6870 -run-as=root
```

DECIMAL	HEXADECIMAL	DESCRIPTION
1904228	0x1D0E64	SHA256 hash constants, little endian
4112676	0x3EC124	SHA256 hash constants, little endian
5877920	0x59B0A0	Linux kernel version 3.10.3
6120324	0x5D6384	AES S-Box
6120580	0x5D6484	AES Inverse S-Box
6176102	0x5E3D66	Unix path: /var/run/rpcbind.sock
6261498	0x5F8AFA	MPEG transport stream data
6261758	0x5F8BFE	MPEG transport stream data
6902132	0x695174	Unix path: /dev/vc/0
6993884	0x6A87DC	xz compressed data
7027944	0x6B3CE8	Unix path: /lib/firmware/updates/3.10.39
7048952	0x6B8EF8	Ubiquiti firmware additional data, name: UTE DEVICE DIAGNOSTIC, size: 1313820672 bytes, size2: 1179407699 bytes, CR C32: 0
7194599	0x6DC7E7	Copyright string: "Copyright 2005-2007 Rodolfo Giometti <giometti@linux.it>"
7218153	0x6E23E9	Copyright string: "Copyright(c) Pierre Ossman"
7301128	0x6F6808	Unix path: /etc/init.d/ipv6_react_to_ra.sh
7304520	0x6F7548	Neighborly text, "NeighborSolicits6InDatagrams"
7304540	0x6F755C	Neighborly text, "NeighborAdvertisementsorts"
7309086	0x6F871E	Neighborly text, "neighbor %2x.%2x.%M lost rename link %s to %s"
7946423	0x7940B7	LZMA compressed data, properties: 0xC0, dictionary size: 0 bytes, uncompressed size: 64 bytes
7970360	0x799E38	ASCII cpio archive (SVR4 with no CRC), file name: "dev", file name length: "0x00000004", file size: "0x00000000"
7970476	0x799EAC	ASCII cpio archive (SVR4 with no CRC), file name: "dev/console", file name length: "0x0000000C", file size: "0x00000000"
7970600	0x799F28	ASCII cpio archive (SVR4 with no CRC), file name: "root", file name length: "0x00000005", file size: "0x00000000"
7970716	0x799FC0	ASCII cpio archive (SVR4 with no CRC), file name: "TRAILER!!!", file name length: "0x0000000B", file size: "0x00000000"
7996352	0x7A03C0	CRC32 polynomial table, little endian
8075823	0x7B3A2F	LZMA compressed data, properties: 0xC0, dictionary size: 0 bytes, uncompressed size: 32 bytes
8227376	0x7D8A30	Copyright string: "Copyright 2000-2013, Marvell International Ltd."
8275455	0x7E45FF	LZMA compressed data, properties: 0xC0, dictionary size: 0 bytes, uncompressed size: 192 bytes

From the above output, we can assume that this is an updated file of the firmware for version 3.10.39.

Analysing the extracted 6870 files, we found that they don't contain much informational data on it, except 799E38.cpio file which might contain interesting information.

```
(root㉿kali)-[~/firmware/_FW_WRT1900ACSV2_2.0.3.201002_prod.img.extracted]
# ls
600000.jffs2 6870 _6870.extracted

(root㉿kali)-[~/firmware/_FW_WRT1900ACSV2_2.0.3.201002_prod.img.extracted]
# cd _6870.extracted

(root㉿kali)-[~/firmware/_FW_WRT1900ACSV2_2.0.3.201002_prod.img.extracted/_6870.extracted]
# ls
6AB7DC.xz 7940B7 7940B7.7z 799E38.cpio 7B3A2F 7B3A2F.7z 7E45FF 7E45FF.7z console cpio-root dev root

(root㉿kali)-[~/firmware/_FW_WRT1900ACSV2_2.0.3.201002_prod.img.extracted/_6870.extracted]
#
```

Analysing 799E38.cpio file using strings command.

Cpio stands for “copy in, copy out”. It is used for processing archive files such as *.cpio or *.tar and can transfer files in both directions from or into them. Additionally, this command allows accessing archived folders by copying files directly.

```
$ strings 799E38.cpio
```

```
mark
Linux
Shelby
3.10.39
#1 SMP Wed Apr 22 04:07:09 PDT 2020
(none)
swapper
atst
earlycon
marvell,orion-system-controller
marvell,armada-370-xp-system-controller
marvell,armada-375-system-controller
marvell,armada-380-system-controller
arm,cortex-a9-scu
arm,cortex-a9-scu
marvell,coherency-fabric
marvell,armada-370-coherency-fabric
marvell,armada-375-coherency-fabric
marvell,armada-380-coherency-fabric
marvell,armada-370-pmsu
marvell,armada-370-xp-pmsu
marvell,armada-380-pmsu
marvell,armada-370-cpu-reset
marvell,armada-xp-cpu-reset
marvell,armada-375-cpu-reset
marvell,armada-380-cpu-reset
marvell,armada-375-usb-cluster
marvell,armada-380-usb-utmi
marvell,armada-375-common-phy-configuration
marvell,armada-375-serdes-pipe-configuration
marvell,armada-375-ip-configuration
/sbin/poweroff
/sbin/modprobe
atst
tasks
cgroup.procs
```

Analysing 600000.jffs2 file and examining its content.

Using file command to analyse file type.

```
$ file 600000.jffs2
```

```
└─(root㉿kali)-[~/firmware/_FW_WRT1900ACSV2_2.0.3.201002_prod.img.extracted]
  └─# ls
    600000.jffs2  6870 _6870.extracted

  └─(root㉿kali)-[~/firmware/_FW_WRT1900ACSV2_2.0.3.201002_prod.img.extracted]
    # file 600000.jffs2
    600000.jffs2: Linux jffs2 filesystem data little endian

  └─(root㉿kali)-[~/firmware/_FW_WRT1900ACSV2_2.0.3.201002_prod.img.extracted]
    #
```

From the above output, it shows the filesystem is jffs2 in little endian format.

Journalling Flash File System version 2 or JFFS2 is a log-structured file system for use with flash memory devices.

Mounting the JFFS2 filesystem.

1. Create the block device.

```
$ mknod /dev/mtdblock0 b 31 0
```

2. Create a directory for the **JFFS2 filesystem** in /mnt directory.

```
$ mkdir /mnt/jffs2_file
```

3. Load the required **kernel modules**. These modules provide the necessary functionality to interact with JFFS2 filesystems and memory technology devices.

```
$ modprobe jffs2  
$ modprobe mtdram  
$ modprobe mtldbck
```

4. Write the image to **/dev/mtldbck0** using **dd** too.

```
$ dd if=600000.jffs2 of=/dev/mtldbck0
```

5. Mount the filesystem to **/mnt/jffs2_file**

```
$ mount -f jffs2 /dev/mtldbck0 /mnt/jffs2_file
```

6. Jump into the mounted filesystem.

```
$ cd /mnt/jffs2_file
```

This command sequence facilitates the extraction and mounting of a JFFS2 filesystem from a router firmware image.

1. It initializes a block device **mtldbck** for accessing flash memory.
2. Set up a directory to mount the **JFFS2 filesystem**.
3. Necessary kernel modules (jffs2, mtdram, mtldbck) are loaded to enable interaction with the JFFS2 filesystem.
4. The firmware image is written onto the block size.
5. The JFFS2 filesystem is mounted onto the designated directory.

```
[root@kali]~/.firmware/_FW_WRT1900ACSV2_2.0.3.201002_prod.img.extracted]  
# mknod /dev/mtldbck0 b 31 0  
  
[root@kali]~/.firmware/_FW_WRT1900ACSV2_2.0.3.201002_prod.img.extracted]  
# mkdir /mnt/jffs2_file  
  
[root@kali]~/.firmware/_FW_WRT1900ACSV2_2.0.3.201002_prod.img.extracted]  
# modprobe jffs2  
  
[root@kali]~/.firmware/_FW_WRT1900ACSV2_2.0.3.201002_prod.img.extracted]  
# modprobe mtdram  
  
[root@kali]~/.firmware/_FW_WRT1900ACSV2_2.0.3.201002_prod.img.extracted]  
# modprobe mtldbck  
  
[root@kali]~/.firmware/_FW_WRT1900ACSV2_2.0.3.201002_prod.img.extracted]
```

```
[root@kali]~/.firmware/_FW_WRT1900ACSV2_2.0.3.201002_prod.img.extracted]  
# dd if=600000.jffs2 of=/dev/mtldbck0  
dd: writing to '/dev/mtldbck0': No space left on device  
8193+0 records in  
8192+0 records out  
4194304 bytes (4.2 MB, 4.0 MiB) copied, 0.0147081 s, 285 MB/s  
  
[root@kali]~/.firmware/_FW_WRT1900ACSV2_2.0.3.201002_prod.img.extracted]  
# mount -t jffs2 /dev/mtldbck0 /mnt/jffs2_file  
  
[root@kali]~/.firmware/_FW_WRT1900ACSV2_2.0.3.201002_prod.img.extracted]
```

Step 3: File System Analysis

Moving to **/mnt/jffs2_file**, we can list our mounted file.

```
[root@kali]-(~/firmware/_FW_WRT1900ACSV2_2.0.3.201002_prod.img.extracted]
# cd /mnt/jffs2_file

[root@kali]-(/mnt/jffs2_file]
# ls
bin cgroup dev etc home JNAP lib linuxrc mnt opt proc root sbin sys tmp usr var www
[root@kali]-(/mnt/jffs2_file]
#
```

Analysing the **bin** folder, found out it is using busy box. In this firmware, most of the binary is linked to a busy box which is explained below; looking for other binaries, we can assume that it must be using services like SQLite3, smb, and http.

Understanding Busy Box

BusyBox is often referred to as the “*Swiss Army Knife of Embedded Linux*.” It combines lightweight, stripped-down versions of many common UNIX utilities into a single executable file. Instead of having dozens of separate binaries for commands like **ls**, **cat**, **cp**, **mv**, **grep**, or **tar**, BusyBox provides them all within one compact program.

This design makes it ideal for **embedded systems** such as routers, IoT devices, and consumer electronics, where storage and memory are limited. While each tool included in BusyBox offers fewer features than its full-sized counterpart in standard Linux distributions, they retain enough functionality to handle most essential tasks.

Key benefits of BusyBox include:

- **Efficiency** – Reduces storage and memory footprint by consolidating utilities into one binary.
- **Flexibility** – Can be configured at build time to include only the commands needed for a specific device.
- **Portability** – Widely adopted across embedded Linux systems, ensuring consistency and reliability.
- **Simplicity** – Provides a minimal yet functional command-line environment, perfect for troubleshooting and scripting.

Common uses in firmware analysis:

When examining an extracted firmware’s filesystem, finding that many binaries link to BusyBox is a strong indicator the device is running a Linux-based embedded system. Analysts can then infer:

- The system likely uses BusyBox for core shell operations.
- Configuration files may reveal which services (like **httpd**, **telnetd**, or **udhcpc**) are being managed by BusyBox.
- BusyBox’s built-in tools can reveal much about how the device is initialized and controlled.

In short, BusyBox is a cornerstone of embedded Linux environments: small, versatile, and highly optimized for constrained hardware.

```
(root@kali)-[/mnt/jffs2_file]
# ls -la bin
total 1357
drwxr-xr-x 2 root root 0 Apr 22 2020 .
drwxr-xr-x 17 root root 0 Dec 31 1969 ..
lrwxrwxrwx 1 root root 7 Apr 22 2020 addgroup → busybox ↗
lrwxrwxrwx 1 root root 7 Apr 22 2020 adduser → busybox
lrwxrwxrwx 1 root root 7 Apr 22 2020 ash → busybox
-rwrxr-xr-x 1 root root 7112 Apr 22 2020 attr
-rwrxr-xr-x 1 root root 593280 Apr 22 2020 busybox
lrwxrwxrwx 1 root root 7 Apr 22 2020 cat → busybox
lrwxrwxrwx 1 root root 7 Apr 22 2020 catv → busybox
-rwrxr-xr-x 1 root root 8644 Apr 22 2020 chacl
lrwxrwxrwx 1 root root 7 Apr 22 2020 chgrp → busybox ↗
lrwxrwxrwx 1 root root 7 Apr 22 2020 chmod → busybox
lrwxrwxrwx 1 root root 7 Apr 22 2020 chown → busybox
lrwxrwxrwx 1 root root 7 Apr 22 2020 cp → busybox
lrwxrwxrwx 1 root root 7 Apr 22 2020 cpio → busybox
-rwrxr-xr-x 1 root root 88232 Apr 22 2020 curl
-rwrxr-xr-x 1 root root 6039 Apr 22 2020 curl-config
lrwxrwxrwx 1 root root 7 Apr 22 2020 date → busybox
lrwxrwxrwx 1 root root 7 Apr 22 2020 dd → busybox
lrwxrwxrwx 1 root root 7 Apr 22 2020 delgroup → busybox
lrwxrwxrwx 1 root root 7 Apr 22 2020 deluser → busybox
lrwxrwxrwx 1 root root 7 Apr 22 2020 df → busybox
lrwxrwxrwx 1 root root 7 Apr 22 2020 dmesg → busybox
lrwxrwxrwx 1 root root 7 Apr 22 2020 dnsdomainname → busybox ↗
lrwxrwxrwx 1 root root 7 Apr 22 2020 dumpkmap → busybox
lrwxrwxrwx 1 root root 7 Apr 22 2020 echo → busybox
lrwxrwxrwx 1 root root 7 Apr 22 2020 ed → busybox
lrwxrwxrwx 1 root root 7 Apr 22 2020 egrep → busybox
lrwxrwxrwx 1 root root 7 Apr 22 2020 false → busybox
```

Step 4: Static Analysis

Examine **etc** folder for configuration files.

```
(root@kali)-[/mnt/jffs2_file]
# cd etc
[root@kali-OptiPlex-5090 etc]# ls
24G_power_table_AP certs ethertypes jcpd.conf product.type syseventp.conf
24G_power_table_AU cloud_dns_names files-to-keep.conf l2tp profile syseventrelay.conf
24G_power_table_CA cron group l7-protocols protocols system
24G_power_table_CE ddns_update.conf guardian ld.so.conf radvd.conf system_defaults
24G_power_table_FCC ddns_update.out hostname led regccode timesettings
24G_power_table_PH devregex.json hosts lighttpd.conf registration.d udev
5G_power_table_AP dhclient.conf hotplug2-common.rules mediaserver.ini resolv.conf version
5G_power_table_AU dhcp6s.conf hotplug2-init.rules mini_httpd.conf ripd.conf VLANTagging_ISP_Profiles.json
5G_power_table_CA dhcp_options hotplug2.rules modprobe.d scsi_id.config vsftpd.conf
5G_power_table_CE dhcp_static_hosts hotplug.d nvram.cleanup.lst security wifi_power_table
5G_power_table_FCC dnsmasq.conf leases otherservices services zebra.conf
5G_power_table_PH dnsmasq.leases IGD passwd
builddate dropbear_dss_host_key igmpproxy.conf init.d persistence_settings ssmtp
builddate.timet dropbear_rsa_host_key ebttables inittab ppp ssmtp-sample
builddetails environment iproute2 product sysconfig
buildrev
```

Analysing the **etc** folder, we can find many interesting configuration files related to the router. Analyzing the **system_defaults** file contains juicy information and reveals the password for **http_admin_password** and many others.

```
$ cat system_defaults | grep password
```

```

└─[root@kali]─[/mnt/jffs2_file/etc]
# cat system_defaults | grep password
# wan_proto_password
# wan_proto_password - the password
$wan_proto_password=
# wan_proto_password - the password
# wan_proto_password - the password
# ddns_username/ddns_password - the credentials for the ddns service
$ddns_password=none
# default admin username & password ("admin" in encrypted form)
# http_is_admin_default indicates whether the default password is being used
$http_admin_password=TSLIIHauhEfge
# Default user / passwords for file sharing use
# $aiccu_enable = 0|1 1 is enabled, 0 is disabled even if $aiccu_user/password are configured
#$aiccu_password=T@#dk
# $he_enable = 0|1 1 is enabled, 0 is disabled even if $he_user/password are configured
#$he_password=T@#dk
# default access password
$guest_password=BeMyGuest
$wl1_guest_password=BeMyGuest
# Parental Control access password.
$parental_control_password=

└─[root@kali]─[/mnt/jffs2_file/etc]

```

We can analyse all configuration files to find more information about the product. Some of them which store interesting information about the product are **buildate**, **version**, **dropbear_rsa_host_key**.

TL; DR

Analyzing firmware and extracting filesystems is a powerful way to understand and secure embedded systems. Through this process, we can:

- Expose vulnerabilities.
- Identify misconfigurations.
- Gain architectural insights.

By systematically dissecting firmware, organizations can build stronger defenses, meet compliance standards, and stay ahead of evolving threats.

At [Redfox Security](#), our global team of security consultants helps organizations identify vulnerabilities and strengthen their defenses. If you're looking to improve your security posture, [contact us](#) to discuss your testing needs.

We also offer [comprehensive courses](#) to help you build expertise in areas like firmware analysis and embedded security. Join us and start your journey toward mastering the hidden layers of technology.

[Previous Cybersecurity Strategies For Small And Mid-Sized Businesses \(SMBs\)](#)
[Next Security Advisory – Multiple Vulnerabilities in Netgear WNR614 Router](#)

Recent Blog

September 09, 2025

[Is APK Decompilation Legal? What You Need To Know](#)

September 06, 2025

[When Hackers Hit the Road: The Jaguar Land Rover Cyberattack](#)

September 05, 2025

[This Is the Hacker's Swiss Army Knife. Have You Heard About It?](#)