Better Report

Team BetterTeam

COSC 6840 Ethical Hacking Theory
Midterm Project Firmware Vulnerability Research N Analysis
Leyton Meadows, Spencer Christensen, John Freeborn
October 12, 2025

Table of Contents

1. Introduction

ThanosTech LLC requested a static firmware security assessment of three firmware images used in communication relay devices: a TP-Link TL-WR841N router image, a D-Link DCS-8000LH IP-camera image, and a bare-metal firmware ELF designed to simulate an stm microcontroller. The scope of engagement was limited to offline static analysis and reproducible automation, meaning there was no live exploitation or network testing performed.

1.1 Objective

The objective of testing was to identify risks such as hardcoded credentials, insecure configurations, outdated components, and embedded secrets. Additionally, a custom automation script was developed to accelerate triage and ensure repeatable results. Based on the findings, mitigation strategies were proposed to strengthen the overall security posture of the firmware.

2. Methodologies

All analysis was performed on a clean Ubuntu 22.04 LTS virtual machine with a few additional packages installed to support firmware extraction and inspection. The VM provided a controlled and isolated environment suitable for static analysis and inspection.

2.1 Core Tools

binwalk – primary extractor and analyzer for embedded firmware images.

file – identified binary types, architectures, and compression formats.

ripgrep – recursively searched extracted directories for sensitive keywords such as “password”, “telnet”, or “key”.

strings – extracted human-readable text, version information, and configuration data from binaries.

readelf – inspected ELF metadata, including architecture. ABI version, and section offsets, to guide binary dosassembly and section-level analysis

objdump – produced human-readable assembly listings from ELF binaries, allowing inspection of ARM opcodes, vector tables, and instruction flow during static analysis

2.2 Analysis Workflow

The firmware assessment followed a structured methodology adapted for static firmware analysis. Each phase built on the previous one, ensuring consistent and reproducible results across all firmware samples.

2.2.1 OSINT Reconnaissance

Before touching the binaries we collected open-source contect for each device family. Using device identifiers and build metadata (model strings, BusyBox/Dropbear/OpenSSL versions, Jenkins/job names) extracted from initial strings passes, we performed passive OSINT checks to search NVD/Exploit-DB for known CVEs for the discovered component versions and checked manufacturer and user manuals to reveal any other insecure configurations such as automatically set default passwords.

2.2.2 Initial Reconnaissance

All firmware images were transferred into an isolated Ubuntu 22.04 LTS virtual machine configured with Python 3.10 and the Binwalk suite. The file command and binwalk were first used to determine each image’s type, compression format, and potential filesystem layout. For the bare-metal ELF, readelf identified the architecture and entry-point metadata to guide disassembly.

2.2.3 Extraction and Exploration

Linux-based firmware images (TP-Link and D-Link) were extracted using binwalk -eM, revealing SquashFS root filesystems with standard Linux directory structures. For the ELF binary, no filesystem was present; instead, code sections were isolated and objcoopy was used for deeper inspection. Directory listings were recorded, and key files such as /etc/init.d scripts and configuration files were identified for further review.

2.2.4 Enumeration of Components

Within each extracted filesystem, service startup scripts, configuration files, and user authentication artifacts were located. The team cataloged kernel and root filesystem components, noted BusyBox versions, and documented the presence of network-related services such as HTTP, Telnet, and SSH. For the ELF, section headers from readelf -S defined logical boundaries for code (.text) and data (.data) regions.

2.2.5 Vulnerability Discovery

Using ripgrep and strings, we performed targeted keyword sweeps for hardcoded credentials, private keys and API tokens. During this phase, both D-Link firmware and TP-Link firmware had hard coded root password hashes that were connected to a startup script that copied them into /etc/shadow and /var/password respectively at boot. Meanwhile the ELF contained embedded RSA, PKCS#8, Github, and AWS private keys. Both of these indicate insecure credential storage within compiled code.

2.2.6 Disassembly and Code Inspection

Objdump -D was used to disassemble the ELF's .text section and examine instruction flow. The first 64bytes displayed LDR and B branch opcodes characteristic of ARM interrupt vvector tables, confirming bare-metal behavior. Instruction patterns and function references helped determine how initialization and cryptographic routines were implemented.

2.2.7 Automation and Reporting

A Python script (fw_triage.sh) was developed to replicate extraction and scanning tasks across multiple images. The script automatically runs several commands (binwalk, strings, file, readelf, etc.), searched for credential-related patterns, and produced structured markdown output summarizing potential findings. Screenshots, logs, and Markdown results were incorporated into the final technical report and presentation.

2.3 Firmware Structural Identification

Each firmware image was analyzed to identify its kernel image type, filesystem structure, and processor architecture. The following summarizes key structural characteristics from binwalk output and supporting tool analysis.

2.3.1 D-Link DCS-8000LH Firmware

Kernel Image Type

Binwalk detected multiple JBOOT SCH2 kernel headers and uImage entries, indicating use of a legacy U-Boot-compatible kernel loader. One section references a Linux 3.10 kernel:

```
lmeadows16@leytonvm:~/EthicalHacking/team-betterteam-fw/bins$ binwalk dcs-8000lh.bin

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
134684        0x20E1C         CRC32 polynomial table, little endian
264356        0x408A4         PEM RSA private key
393216        0x60000         Squashfs filesystem, little endian, version 4.0, compression:xz, size: 276064 bytes, 14 inodes, blocksize: 131072
 bytes, created: 2017-06-07 10:12:50
1441792       0x160000        gzip compressed data, maximum compression, from Unix, last modified: 2017-06-07 00:00:47
1703936       0x1A0000        gzip compressed data, maximum compression, from Unix, last modified: 2017-06-07 00:00:47
1966080       0x1E0000        uImage header, header size: 64 bytes, header CRC: 0x8237D0E6, created: 2017-06-07 09:44:00, image size: 1661677 b
ytes, Data Address: 0x804D4940, Entry Point: 0x804D4940, data CRC: 0x5C8D17AC, OS: Linux, CPU: MIPS, image type: OS Kernel Image, compression t
ype: none, image name: "linux_3.10"
1970496       0x1E1140        LZMA compressed data, properties: 0x5D, dictionary size: 67108864 bytes, uncompressed size: -1 bytes
5111808       0x4E0000        Squashfs filesystem, little endian, version 4.0, compression:xz, size: 7657880 bytes, 2125 inodes, blocksize: 131
072 bytes, created: 2017-06-07 10:13:14
```

Figure 2.3.1.1: Binwalk over the DLink DCS-8000LH Firmware showing the Linux kernel

This confirmes a MIPS-based Linux kernel, typical for consumer IoT devices of that generation.

Root Filesystem Format

Two SquashFS filesystems were discovered:

```
lmeadows16@leytonvm:~/EthicalHacking/team-betterteam-fw/bins$ binwalk dcs-8000lh.bin

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
134684        0x20E1C         CRC32 polynomial table, little endian
264356        0x408A4         PEM RSA private key
393216        0x60000         Squashfs filesystem, little endian, version 4.0, compression:xz, size: 276064 bytes, 14 inodes, blocksize: 131072
 bytes, created: 2017-06-07 10:12:50
1441792       0x160000        gzip compressed data, maximum compression, from Unix, last modified: 2017-06-07 00:00:47
1703936       0x1A0000        gzip compressed data, maximum compression, from Unix, last modified: 2017-06-07 00:00:47
1966080       0x1E0000        uImage header, header size: 64 bytes, header CRC: 0x8237D0E6, created: 2017-06-07 09:44:00, image size: 1661677 b
ytes, Data Address: 0x804D4940, Entry Point: 0x804D4940, data CRC: 0x5C8D17AC, OS: Linux, CPU: MIPS, image type: OS Kernel Image, compression t
ype: none, image name: "linux_3.10"
1970496       0x1E1140        LZMA compressed data, properties: 0x5D, dictionary size: 67108864 bytes, uncompressed size: -1 bytes
5111808       0x4E0000        Squashfs filesystem, little endian, version 4.0, compression:xz, size: 7657880 bytes, 2125 inodes, blocksize: 131
072 bytes, created: 2017-06-07 10:13:14
```

Figure 2.3.1.2: Binwalk over the DLink DCS-8000LH Firmware showing SquashFS

Both employ XZ compression, with timestamps from 2017, suggesting the firmware build is several years out of date

Startup Scripts and Configuration Files

Within the extracted root filesystem, initialization logic was discovered under _dcs-8000lh.bin.extracted/squashfs-root-0/startkit/**boot.sh**

Vector table, Interrupt Handlers, Magic Constants

As the entry point for the linux kernel image is listed as 0x804D4940 interrupt stubs may be close to this location.

Directory Structure

/bin – BusyBox utilities and networking tools

/etc – Init scripts, service configs, and keys

/lib – Shared object libraries

/startkit - Boot logic scripts

2.3.2 TP-Link WR-841N Firmware

Kernel Image Type

binwalk analysis revealed a U-Boot 1.1.3 image header compiled in August 2022:

```
DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
53536         0xD120          U-Boot version string, "U-Boot 1.1.3 (Aug 16 2022 - 12:01:12)"
66048         0x10200         LZMA compressed data, properties: 0x5D, dictionary size: 8388608 bytes, uncompressed size: 2986732 bytes
1048576       0x100000        Squashfs filesystem, little endian, version 4.0, compression:xz, size: 3001844 bytes, 552 inodes, blocksize: 2621
44 bytes, created: 2022-08-16 04:14:58
```

Figure 2.3.2.1: Binwalk over WR-841N showing U-Boot version

This indicates that the TP-Link firmware uses a modernized U-Boot loader for initializing and booting its embedded Linux system. An internal uImage header was also detected, specifying:

```
lmeadows16@leytonvm:~/EthicalHacking/team-betterteam-fw/bins$ binwalk -Me wr-841n.bin 2>/dev/null

Scan Time:     2025-10-11 06:07:54
Target File:   /home/lmeadows16/EthicalHacking/team-betterteam-fw/bins/wr-841n.bin
MD5 Checksum:  97710bf8ae216d4665c1c89a43eca626
Signatures:    411

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
53536         0xD120          U-Boot version string, "U-Boot 1.1.3 (Aug 16 2022 - 12:01:12)"
66048         0x10200         LZMA compressed data, properties: 0x5D, dictionary size: 8388608 bytes, uncompressed size: 2986732 bytes
1048576       0x100000        Squashfs filesystem, little endian, version 4.0, compression:xz, size: 3001844 bytes, 552 inodes, blocksize: 2621
44 bytes, created: 2022-08-16 04:14:58


Scan Time:     2025-10-11 06:07:54
Target File:   /home/lmeadows16/EthicalHacking/team-betterteam-fw/bins/_wr-841n.bin-0.extracted/10200
MD5 Checksum:  9e838c993a40353344273 0b887a432c3
Signatures:    411

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
2240608       0x223060        Linux kernel version 2.6.36
2240772       0x223104        CRC32 polynomial table, little endian
2280128       0x22CAC0        CRC32 polynomial table, little endian
2509588       0x264B14        xz compressed data
2556368       0x2701D0        Neighborly text, "NeighborSolicits6InDatagrams"
2556388       0x2701E4        Neighborly text, "NeighborAdvertisementsorts"
2559855       0x270F6F        Neighborly text, "neighbor %.2x%.2x.%pM lostrename link %s to %s"
2981888       0x2D8000        ASCII cpio archive (SVR4 with no CRC), file name: "/dev", file name length: "0x00000005", file size: "0x00000000"
2982004       0x2D8074        ASCII cpio archive (SVR4 with no CRC), file name: "/dev/console", file name length: "0x0000000D", file size: "0x0
0000000"
2982128       0x2D80F0        ASCII cpio archive (SVR4 with no CRC), file name: "/root", file name length: "0x00000006", file size: "0x00000000
"
2982244       0x2D8164        ASCII cpio archive (SVR4 with no CRC), file name: "TRAILER!!!", file name length: "0x0000000B", file size: "0x000
00000"
```

Figure 2.3.2.2: Binwalk over TPLink WR-841N showing compression, OS, and CPU

confirming a MIPS based Linux kernel packaged via U-Boot.

Root Filesystem Format

binwalk identified the filesystem as SquashFS v4.0 using XZ compression, optimized for small flash memory footprints:


```
DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
53536         0xD120          U-Boot version string, "U-Boot 1.1.3 (Aug 16 2022 - 12:01:12)"
66048         0x10200         LZMA compressed data, properties: 0x5D, dictionary size: 8388608 bytes, uncompressed size: 2986732 bytes
1048576       0x100000        Squashfs filesystem, little endian, version 4.0, compression:xz, size: 3001844 bytes, 552 inodes, blocksize: 2621
44 bytes, created: 2022-08-16 04:14:58
```

Figure 2.3.2.3: Binwalk showing SquashFS system

The presence of a single SquashFS root image at offset 0x100000 suggests a standard, monolithic filesystem structure rather than a modular multi-image layout.

Start Scripts

Following extraction with binwalk -Me wr-841n.bin, standard linux directories were observed like /etc/init.d which contained rcS which is the base init script and /etc/rc.d. Startup logic includes service configuration scripts for network interfaces, DHCP, and firewall setup, consistent with TP-Link's open-source-derived embedded Linux build chain.

```
#!/bin/sh

mount -a
# added by yangcaiyong for sysfs
mount -t sysfs /sys /sys
# ended add

/bin/mkdir -m 0777 -p /var/https
/bin/mkdir -m 0777 -p /var/lock
/bin/mkdir -m 0777 -p /var/log
/bin/mkdir -m 0777 -p /var/run
/bin/mkdir -m 0777 -p /var/tmp
/bin/mkdir -m 0777 -p /var/Wireless/RT2860AP
/bin/mkdir -m 0777 -p /var/tmp/wsc_upnp
cp -p /etc/SingleSKU_FCC.dat /var/Wireless/RT2860AP/SingleSKU.dat

/bin/mkdir -m 0777 -p /var/tmp/dropbear

/bin/mkdir -m 0777 -p /var/dev
cp -p /etc/passwd.bak /var/passwd
/bin/mkdir -m 0777 -p /var/l2tp

echo 1 > /proc/sys/net/ipv4/ip_forward
#echo 1 > /proc/sys/net/ipv4/tcp_syncookies
echo 1 > /proc/sys/net/ipv6/conf/all/forwarding

echo 30 > /proc/sys/net/unix/max_dgram_qlen

#krammer add for LAN can't continuous ping to WAN when exchenging the routing mode
#bug1126
echo 3 > /proc/sys/net/netfilter/nf_conntrack_icmp_timeout

echo 0 > /proc/sys/net/ipv4/conf/default/accept_source_route
```

Figure 2.3.2.4: The start script from TPLink's file system

Type of Architecture Code is Compiled For

The earlier binwalk also revealed that the code is for MIPS 32-bit Little Endian.

Directory Structure

/bin – Core utilities and BusyBox applets

/etc – System configuration files and initialization scripts

/lib – Shared system libraries

/sbin – Administrative binaries

/usr – Application binaries and support tools

/web – Embedded web management interface

/var – Runtime data storage

2.3.3 Bare-Metal Firmware STM Microcontroller

Kernel Image Type / Executable Structure

Unlike the Linux-based firmwares, this binary was a stand-alone ELF32 executable, not a compressed filesystem image. Readelf -h identified it as:

```
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           ARM
  Version:                           0x1
  Entry point address:               0x29
  Start of program headers:          52 (bytes into file)
  Start of section headers:          6564 (bytes into file)
  Flags:                             0x5000200, Version5 EABI, soft-float ABI
  Size of this header:               52 (bytes)
  Size of program headers:           32 (bytes)
  Number of program headers:         1
  Size of section headers:           40 (bytes)
  Number of section headers:         7
  Section header string table index: 6
```

Figure 2.3.3.1: readelf output on top of bare-metal-takehome.elf

indicating a 32-bit ARM executable compiled for the EABI soft-float ABI and designed to run directly on hardware without an operating system.

Section Layout

Using readelf -S, the image contained seven sections:

```
There are 7 section headers, starting at offset 0x19a4:

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .text             PROGBITS        00000000 001000 0008fd 00  AX  0   0  4
  [ 2] .data             PROGBITS        000008fd 0018fd 000000 00  WA  0   0  1
  [ 3] .bss              NOBITS          00000900 0018fd 000018 00  WA  0   0  4
  [ 4] .comment          PROGBITS        00000000 0018fd 000045 01  MS  0   0  1
  [ 5] .ARM.attributes   ARM_ATTRIBUTES  00000000 001942 00002d 00      0   0  1
  [ 6] .shstrtab         STRTAB          00000000 00196f 000035 00      0   0  1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings), I (info),
  L (link order), O (extra OS processing required), G (group), T (TLS),
  C (compressed), x (unknown), o (OS specific), E (exclude),
  D (mbind), y (purecode), p (processor specific)
```

Figure 2.3.3.2: readelf -S over bare-metal-takehome.elf

No .rodata or dynamic linking sections were present, confirming a statically linked bare-metal application.

Root Filesystem / Boot Data

As expected for a bare-metal target, no filesystem was embedded. However, binwalk revealed additional encoded content within the data region, including PEM -formatted private keys and AWS credential strings at offsets 0x1221 and 0x18D1.

Startup / Vector Table Analysis

objdump -D showed the initial instructions beginning at 0x0:

```
00000000 <.text>:
   0:   47704800        ldrbmi  r4, [r0, -r0, lsl #16]!
   4:   00000221        andeq   r0, r0, r1, lsr #4
   8:   47704800        ldrbmi  r4, [r0, -r0, lsl #16]!
   c:   0000020c        andeq   r0, r0, ip, lsl #4
  10:   47704800        ldrbmi  r4, [r0, -r0, lsl #16]!
  14:   000001e3        andeq   r0, r0, r3, ror #3
  18:   47704800        ldrbmi  r4, [r0, -r0, lsl #16]!
  1c:   000000d9        ldrdeq  r0, [r0], -r9
  20:   47704800        ldrbmi  r4, [r0, -r0, lsl #16]!
  24:   000000b0        strheq  r0, [r0], -r0   @ <UNPREDICTABLE>
```

Figure 2.3.3.3: objdump -D bare-metal-takehome.elf (first 11 lines)

These correspond to the standard ARM interrupt vector table, defining reset and exception handlers executed at boot. No memory-protection or signature checks were observed, implying the image would execute unsigned code if flashed directly to the MCU.

Magic Constants and Identifiers

The .text section contained additional plaintext constants including build identifiers and the environment variable:

Figure 2.3.3.4: Redacted GitHub tokens and AWS access key revealing hard-coded cloud credentials used by the application.

Type of Architecture Code is Compiled For

All analysis confirmed ARMv7 compilation using GCC for embedded targets.

Directory Structure

As there was no filesystem, analysis centered on logical sections within the ELF binary:

.text – contains main logic and secrets

3. Findings

Our assessment uncovered multiple security issues across both linux-based and bare-metal firmware images. Each finding is supported by direct evidence from extracted files, disassembly, or OSINT correlation. All sensitive material such as private keys and credentials has been redacted for ethical disclosure.

3.1 Hardcoded Root Password (TP-Link / D-Link Firmware)

Evidence

Within the extracted /etc/passwd.bak for TP-Link /etc/rc.d/rcK_mfg.d/K01local for D-Link and scripts, a command writes a static root password hash to /etc/shadow and /var/passwd respectively during startup.

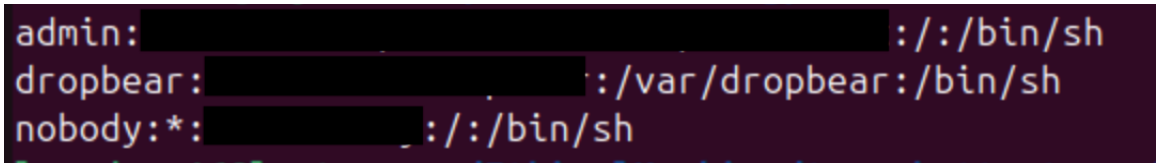
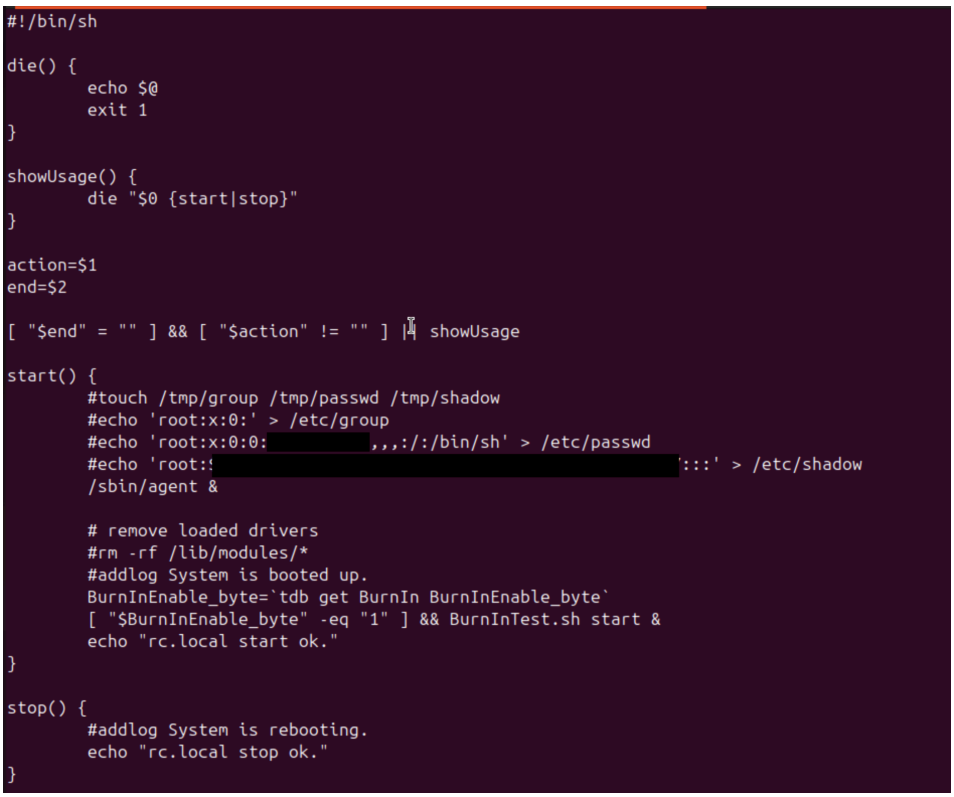


Figure 3.1.1: redacted hardcoded passwords for TPLink



```
#!/bin/sh

die() {
        echo $@
        exit 1
}

showUsage() {
        die "$0 {start|stop}"
}

action=$1
end=$2

[ "$end" = "" ] && [ "$action" != "" ] || showUsage

start() {
        #touch /tmp/group /tmp/passwd /tmp/shadow
        #echo 'root:x:0:' > /etc/group
        #echo 'root:x:0:0:[redacted],,,:/:/bin/sh' > /etc/passwd
        #echo 'root:[redacted]:::' > /etc/shadow
        /sbin/agent &

        # remove loaded drivers
        #rm -rf /lib/modules/*
        #addlog System is booted up.
        BurnInEnable_byte=`tdb get BurnIn BurnInEnable_byte`
        [ "$BurnInEnable_byte" -eq "1" ] && BurnInTest.sh start &
        echo "rc.local start ok."
}

stop() {
        #addlog System is rebooting.
        echo "rc.local stop ok."
}
```

Figure 3.1.2: Redacted password script for DLink

Impact

This creates a predictable credential across all deployed devices. An attacker with shell or telnet access could gain administrative privileges without brute-forcing credentials, violating CWE-259(Use of Hard-Coded Password).

Mitigation

Implement randomized password generation during first-boot initialization or require password setup through a secure provisioning interface. Disable legacy Telnet service in production builds and enforce SSH only.

3.2 Embedded Private Keys and AWS Credentials (Bare-Metal ELF)

Evidence

binwalk identified PEM and PKCS#8 blocks directly within the ELF binary:

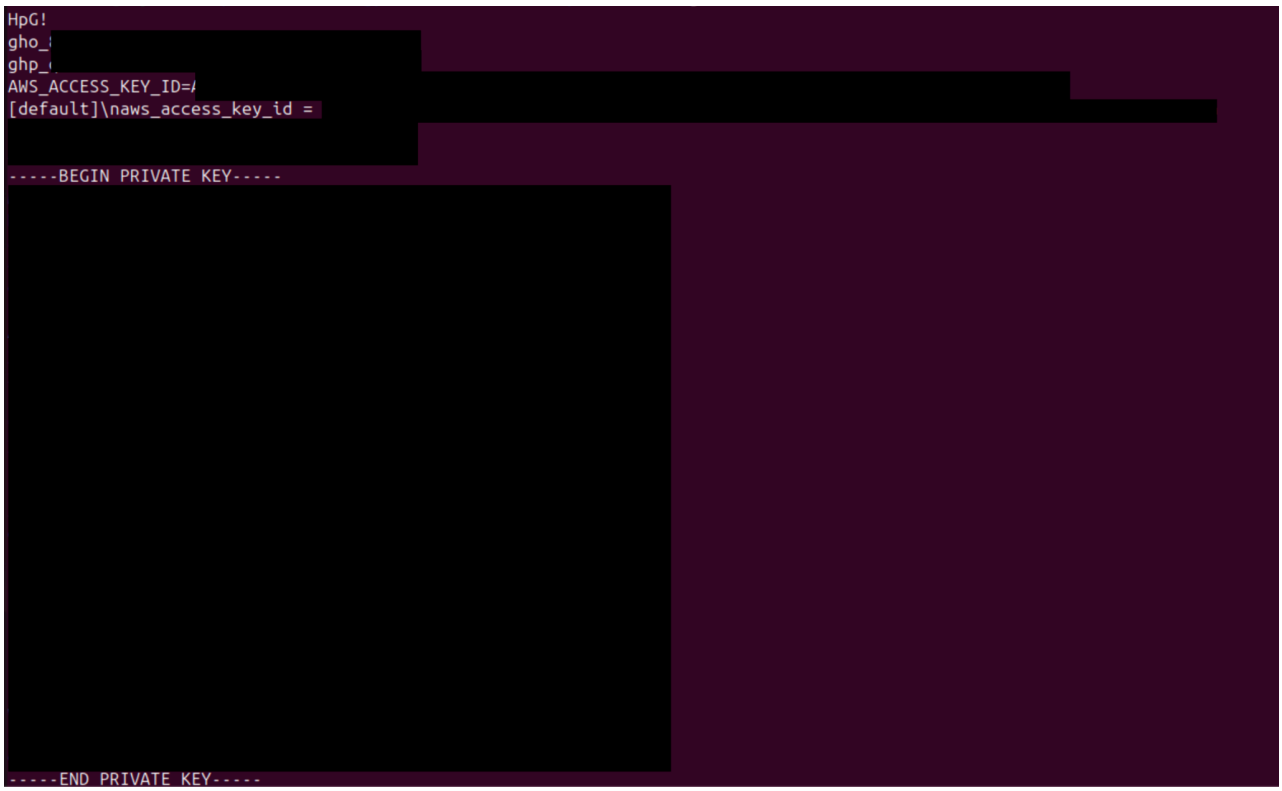


Figure 3.2.1: Redacted GitHub authorization tokens, AWS access keys, and a private key

String analysis confirmed that these keys coexist with plaintext AWS API credentials and GitHub keys.

Impact

Embedded long-term cryptographic material within firmware allows adversaries with binary access to impersonate the device or exfiltrate cloud-stored data. The compromise extends to the backend infrastructure, not just the endpoint. This aligns with CWE-321 (Use of Hard-coded Cryptographic Key) and CWE-798 (Use of Hard-coded Credentials).

Mitigation

Remove all plaintext keys from source code. Store secrets in a hardware secure element (TPM, ATECC608, etc.) or provision device-unique keys at manufacture. Use short-lived, scoped IAM tokens instead of static AWS credentials.

3.3 Exposed Public and Private Keys (D-Link Firmware)

Evidence

During inspection of the extracted D-Link firmware, several key files were found within nested extraction directories:

Figure 3.3.1: Redacted Private Key found at squashfs-root-0/etc/db/db.xml

Figure 3.3.2: Redacted public key found at squashfs-root-0/etc/db/verify.xml

verify.key contains a PEM-formatted public and private key.

Impact

Co-locating unprotected public and private keys introduces a major trust-chain risk. Attackers who obtain these keys could sign malicious firmware updates or intercept encrypted communications. This issue aligns with CWE-320 – Key Management Errors and CWE-522 – Insufficiently Protected Credentials.

Mitigation

Remove any private keys from distributed firmware images

Store signed keys on a secure build server, not within production firmware

Implement signature verification using only public keys and validate firmware authenticity through a hardware-based trust anchor or secure bootloader

3.4 BusyBox v1.37.0 Vulnerabilities (D-Link and TP-Link)

Evidence

strings /bin/busybox | grep BusyBox showed:

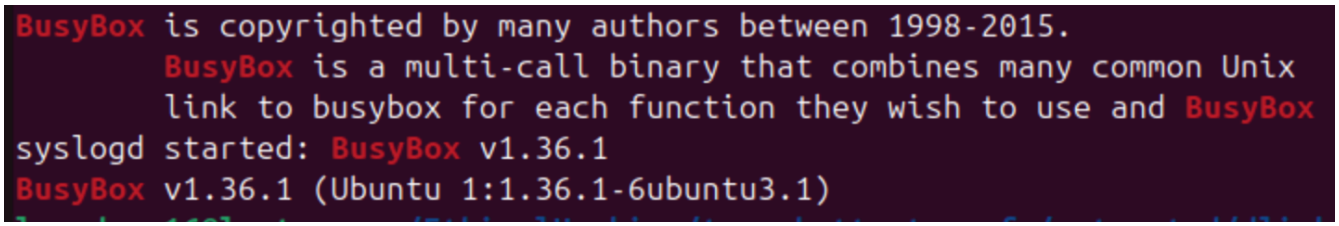


Figure 3.4.1: BusyBox version number

Cross-referencing this version via NVD and Exploit-DB revealed two disclosed CVEs:

CVE-2024-58251 – Allows a local user to perform a denial of service attack and exists due to insufficient validation of user-supplied input in netstat when launched with a specific argument

CVE-2025-46394 – Allows a remote attacker to perform a spoofing attack and exists due to insufficient validation of filenames inside a tar archive.

Impact

While these flaws require user interaction or crafted inputs, they still pose a risk on embedded systems that invoke BusyBox utilities automatically. A malicious firmware update or remote file injection could exploit these vulnerabilities for code execution or data exposure.

Mitigation

Upgrade BuyBox to a later version once patches are integrated upstream (currently unpatched). Until then, disable unused BusyBox applets (notably tar and awk) and apply runtime integrity checks for system scripts.

3.5 Insecure Bootloader and Network Boot Configuration (D-Link Firmware

Evidence

Using binwalk -Me, multiple bootloader and network components were identified, including Flattened Image Tree structures and boot configurations referencing TFTP and NFS network boot protocols. Documentation review and OSINT correlation confirmed that U-Boot versions between 2013.07-rc1 and 2014.07-rc2 are susceptible to several known issues:

Verified Boot Bypass – Improper enforcement of FIT image signatures allows a local attacker to supply a legacy image and execute unsigned kernel code.

Network Boot Vulnerabilities – Versions supporting network protocols such as NFS and TFTP are affected by CVE-2019-14192, which allows code execution via crafted network packets from a malicious NFS server.

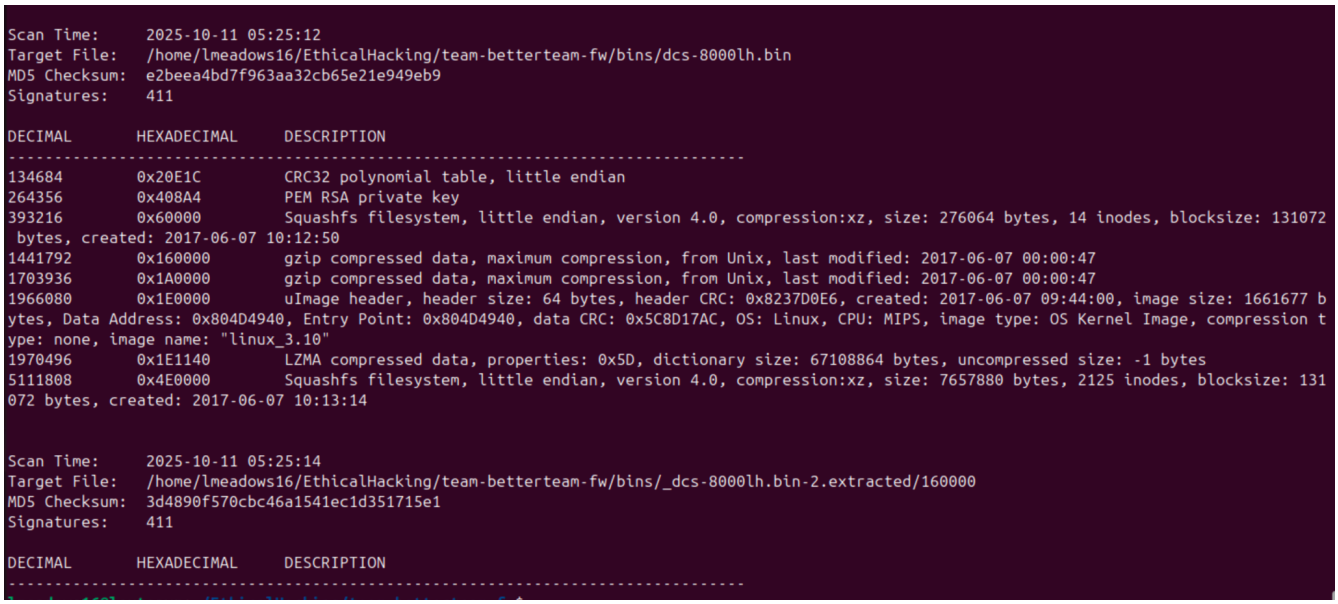


Figure 3.5.1: DLink preliminary binwalk showing Boots and TFTP

Impact

If the firmware's U-Boot version falls within the affected range, attackers with local or physical access could bypass verified boot protections and execute unsigned kernels or inject modified root filesystems. Network booting features further expand the attack surface, allowing remote exploitation in environments where the camera or router connects to untrusted networks.

Mitigation

Upgrade to a modern U-Boot release with enforced signature verification for all FIT images

Disable Network boot protocols in production builds

Require digital signatures on all kernel and initram fs images, validated by a trusted public key stored in read-only memory

Apply firmware-level integrity checks at each boot stage

4. Automation Script

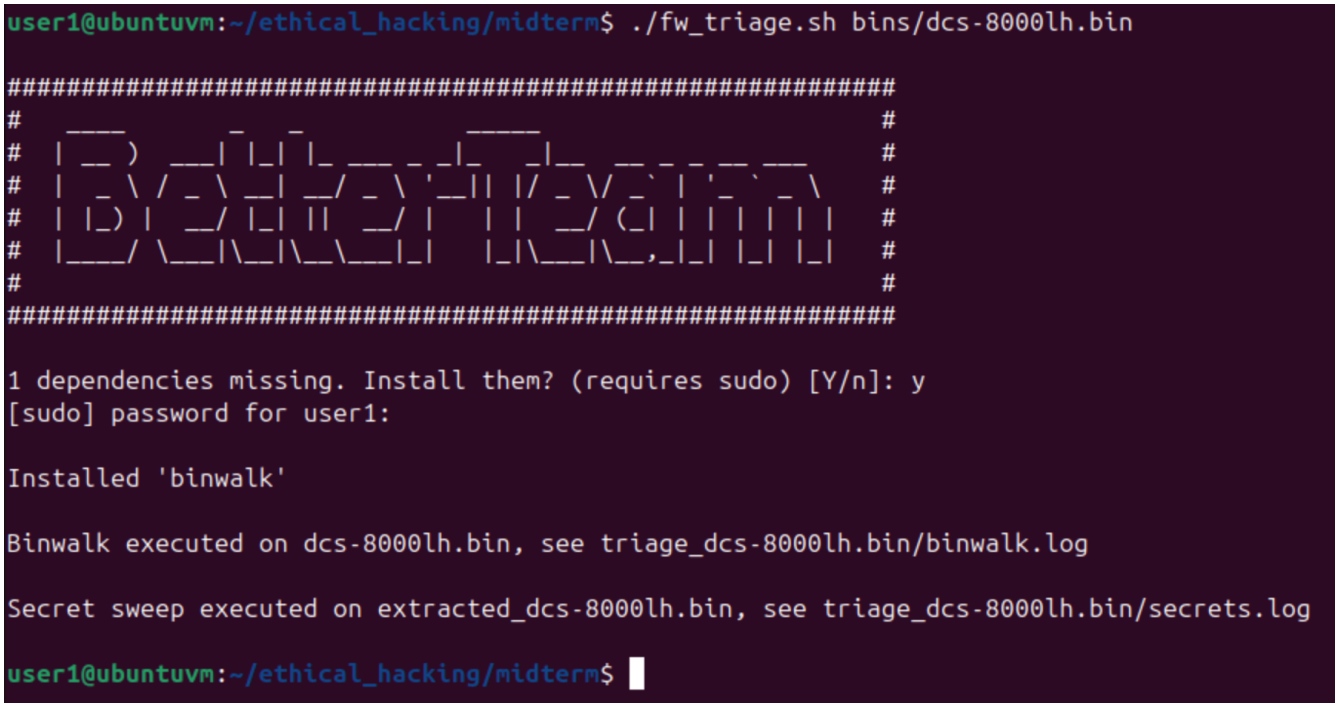


Figure 4.1.1: Script execution, including dependency installation

The purpose of this script is to save time and mental load when investigating binary firmwares. The script is highly extensible and modularized. At present the script has modules for automatic extraction, secret sweeping, and dependency installation; but increasing functionality is trivial. To add a module simply write a shell script that generates your findings and sends them to stdout, then call that script with the 'run_script' function. This will execute your module while passing any arguments, and place the output in a log file under the triage output directory. The script determines binary types and executes modules based on this determination. Additionally, any dependencies should be added to the provided dependency checker, which will check for their existence in the path and prompt for a batch installation as needed.

5. Recommendations

The following recommendations are prioritized according to impact and exploitability observed during the assessment. Each item corresponds to the vulnerabilities documented in Section 3.

5.1 Credential Management and Authentication

1. Eliminate Hard-Coded Passwords
    - Remove static credentials from initialization scripts.
    - Enforce unique password generation at first boot or require secure provisioning through a setup interface.
    - Deactivate legacy Telnet services and enforce SSH with public-key authentication.
2. Implement Secure Password Storage
    - Use salted SHA-512 or bcrypt hashes stored in /etc/shadow.
    - Prevent runtime scripts from rewriting or exposing credential files

5.2 Key and Secret Handling

1. Remove Embedded Keys and Cloud Credentials

- Strip all PEM, PKCS#8, and API key material from firmware images.
- Store device-unique keys in hardware-backed secure elements (TPM 2.0, ATECC608, etc.) or inject them during manufacturing.

2. Separate Build and Runtime Secrets
   - Maintain signing keys exclusively on protected build servers.
   - Use environment variables or encrypted configuration partitions instead of hard-coding values in source.
3. Use Short-Lived Cloud Tokens
   - Replace static AWS credentials with scoped IAM roles and automatically rotated tokens through STS or IoT Core.

5.3 Boot and Firmware Integrity

1. Enable Secure Boot Enforcement
   - Update bootloaders to verified-boot versions (e.g., U-Boot ≥ 2020.01).
   - Validate all kernel and initramfs images using RSA/ECDSA signatures verified by a trusted public key in read-only memory.
2. Protect Interrupt Vector Regions
   - Configure memory-protection units (MPUs) to mark vector tables read-only after initialization.
   - Employ CRC or digital-signature checks on boot code to detect tampering.
3. Disable Network boot Protocols in Production
   - Remove or disable TFTP/NFS boot modes unless explicitly required for manufacturing.

5.4 Component Hardening and Maintenance

1. Update BusyBox and Dependent Packages
   - Upgrade to BusyBox v1.37.2 or later to patch CVE-2024-58251 and CVE-2025-46394.
   - Restrict compiled applets to essential functions to minimize the attack surface.
2. Implement Routine Patch Management
   - Establish an over-the-air update mechanism or signed manual-update process.
   - Regularly monitor NVD and vendor advisories for new CVEs affecting kernel and userspace components.

5.5 Filesystem and Build Hygiene

1. Remove Debug Artifacts and Developer Paths
   - Purge source-path references such as /home/lmeadows16/EthicalHacking/... before release builds.
   - Exclude testing scripts, unused binaries, and commented configuration directives.
2. Restrict Permissions on Sensitive Directories
   - Limit access to /db/, /etc/ssl/, and /etc/init.d/ to root-only.
   - Enforce mount options such as nosuid and noexec on non-system partitions where applicable.

5.6 Operational and Organizational Controls

1. Integrate Static-Analysis and Secret-Scanning in CI/CD
   - Use tools such as *truffleHog* or *git-secrets* during firmware build pipelines to prevent accidental inclusion of credentials.

2. Adopt a Formal Vulnerability-Managment Process
    - Track findings, CVE status, and patch timelines under a change-control policy.
    - Provide customers with clear firmware-update documentation and end-of-support policies.

6. Conclusion

This engagement assessed three firmware images supplied by ThanosTech LLC, two Linux-based embedded firmwares (TP-Link Router and D-Link) and one bare-metal ARM ELF binary, to identify security weaknesses and propose mitigations. All analysis was performed in a controlled Ubuntu 22.04 LTS environment using open-source tooling such as binwalk, readelf, objdump, ripgrep, and a custom bash automation script (fw_triage.sh) to ensure reproducibility.

The investigation demonstrated that even current embedded builds can contain legacy or insecure design choices. Key discoveries included:

- Hard-coded root credentials in initialization scripts, allowing privilege escalation.
- Embedded RSA and AWS cloud keys within the bare-metal ELF image, exposing sensitive backend infrastructure.
- BusyBox v1.37.0 vulnerabilities present in both Linux firmwares.
- Unprotected interrupt vector tables and insecure bootloader configurations, reducing resistance to firmware tampering.
- Residual developer artifacts and debug paths, indicating insufficient release hardening.

Despite these issues, the TP-Link firmware exhibited more recent maintenance practices, including modern compression methods and the absence of plaintext credentials in its filesystem. The D-Link and bare-metal images, however, revealed critical lapses in cryptographic hygiene and boot-integrity enforcement.

Remedial efforts should prioritize:

1. Removal of embedded credentials and private keys.
2. Enforcement of secure-boot validation and hardware-based key storage.
3. Upgrading vulnerable components such as BusyBox.

This project underscores the importance of incorporating firmware-security assessment into the product-development lifecycle. By integrating automated scanning, OSINT-driven CVE correlation, and secret-detection pipelines early in development, ThanosTech LLC can prevent similar issues in future releases and improve both consumer trust and long-term device resilience.