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The objective of testing was to identify risks such as hardcoded credentials, insecure configurations, outdated components, and embedded secrets.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

2.1 Environment and Core Tools

All testing occurred in an isolated Ubuntu 22.04 LTS virtual machine. Open-source utilities were selected for transparency and reproducibility:

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

These tools allowed consistent, evidence-based analysis across Linux and bare-metal firmware samples.

2.2 Analysis Workflow

Firmware analysis followed a repeatable five-phase ethical-hacking workflow:

2.2.1 OSINT Reconnaissance

Each firmware file was initially profiled to determine vendor, architecture, and potential component versions.

- Tools used: file, strings, and binwalk to identify file headers, compression formats, and embedded metadata.
- Purpose: Establish a baseline understanding of image composition before extraction.
  Public sources such as the National Vulnerability Database (NVD) and Exploit-DB were queried for known CVEs associated with any discovered version strings (BusyBox, U-Boot, Dropbear, OpenSSL).
  Findings from this phase guided which binaries and directories would be prioritized for deeper inspection.

2.2.2 Extraction and Filesystem Reconstruction

Firmware binaries were unpacked using:

binwalk -eM <firmware>.bin

This process recursively extracted compressed sections, revealing internal components such as SquashFS images or bootloader regions. When no filesystem existed (as with the ELF image), binary sections were dumped using:

```
objcopy -O binary --only-section=.text bare-metal.elf text.bin
```

All extracted directories were archived in the project repository (extracted/<firmware>.extracted/) for reproducibility.

2.2.3 Enumeration of Components

Once extracted, directory trees were reviewed to locate high-value targets:

- /etc/init.d/ – startup and service initialization scripts
- /db/ – certificates and key material
- /bin/ and /sbin/ – core utilities and BusyBox binaries
- /www/ – web management interfaces (if present)
  Version strings, default configuration files, and service startup orders were logged to map attack surfaces.

Enumeration also included architecture validation using:

```
readelf -h <binary>   # confirm CPU type and endianness
```

and cross-checking compiled libraries for outdated dependencies.

2.2.4 Vulnerability Discovery and Verification

Targeted keyword sweeps were executed to locate sensitive material:

```
rg -n -i "password|passwd|BEGIN RSA|PRIVATE KEY|aws_access_key" extracted/
```

Any positive matches were manually reviewed to verify context.
Binary code inspection with objdump -D highlighted the presence of ARM and MIPS vector tables, confirming bare-metal startup logic.
For each plausible vulnerability, evidence was captured (file path, snippet, or offset) and categorized by type:

- Credential exposure (plaintext passwords, API keys)
- Cryptographic weakness (embedded PEM/PKCS#8 keys)
- Component versioning (outdated binaries with known CVEs)
  All artifacts were redacted and stored as part of the findings record.

2.2.5 Automation, Documentation, and Validation

A custom Bash script (fw_triage.sh) automated repetitive tasks—running Binwalk, searching for credential patterns, and generating log summaries.
Each run produced a binwalk.log which contained a recursive series of binwalks through the files and a secrets.log which contains any found secrets from the binary.

Automation ensured consistency across firmware samples and enabled third-party verification. Final validation involved manual re-inspection of top findings to confirm accuracy before inclusion in the report.

2.3 Firmware Structural Identification

Each firmware image was analyzed to identify its kernel image type, filesystem structure, startup/configuration locations, interupt/vector characteristics, and CPU architecture. The following summarizes key structural characteristics from binwalk output and supporting tool analysis.

2.3.1 D-Link DCS-8000LH Firmware

Using:

Binwalk dcs-8000lh.bin

Revealed uImage headers and nested SquashFS extracts

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

Findings

- **Kernel image type**: U-Boot / JBOOT packaged Linux kernel (kernel string indicates linux_3.10).
- **Root filesystem**: SquashFS v4.0 (XZ compression), multiple squashfs images present.
- **Startup/config**: startkit/boot.sh, mydlink-watch-dog.sh (e.g., VERSION=2.1.0-b83) and /etc/init.d scripts (one writes a root hash at boot).
- **Keys / secrets**: PEM files and key material found under extracted paths such as _.../db/verify.key and adjacent private-key files.
- **Architecture**: MIPS32, little-endian (confirmed by readelf/kernel header).

For further discussion of findings see section 3.

2.3.2 TP-Link WR-841N Firmware

Using:

binwalk -eM wr-841n.bin

Revealed a U-Boot header and a single SquashFS rootfs

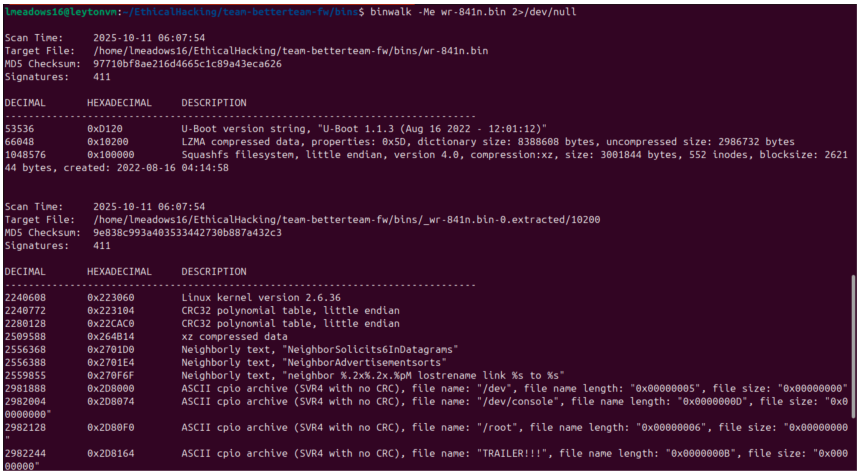


Figure 2.3.2.1: Binwalk over TPLink WR-841N showing compression, OS, and CPU

Findings

- **Kernel image type**: U-Boot packaged Linux kernel (uImage; MIPS CPU).
- **Root filesystem**: SquashFS v4.0 (XZ).
- **Startup/config**: Standard /etc/init.d scripts and web interface files in /www. No plaintext credentials discovered in extracted configs.
- **Architecture**: MIPS32, little-endian.

```
#!/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 2.3.2.2: The start script from TPLink's file system

2.3.3 Bare-Metal Firmware STM Microcontroller

Using:

readelf -h bare-metal-takehome.elf

readelf -S

Objdump -D

Confirmed an Elf32 executable for ARM AEBI v5

```
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

```
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

```
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)

Findings

- **Type**: Stand-alone ELF32 executable (no filesystem).
- **Sections**: .text, .data, .bss, .comment, .ARM.attributes, .shstrtab — data and string constants live in .text/.data.
- **Startup / vectors**: ARM reset/exception vector pattern at address 0x0 (e.g., LDR pc, [pc, #0x18] then a branch), indicating bare-metal bootstrap in code.
- **Keys / secrets**: PEM-formatted key blocks and AWS credential-like strings were discovered inside binary sections at specific offsets.
- **Architecture**: ARMv7 (ELF32, little-endian, EABI v5 soft-float).

3. Findings

Static analysis of the three firmware images revealed multiple weaknesses in credential hardening, key management, and boot integrity. Each issue below is supported by evidence captured during analysis which has been redacted for ethical disclosure.

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

Evidence

/etc/passwd.bak for TP-Link and /etc/rc.d/rcK_mfg.d/K01local for D-Link write static passwords to /var/passwd and /etc/shadow respectively during boot:

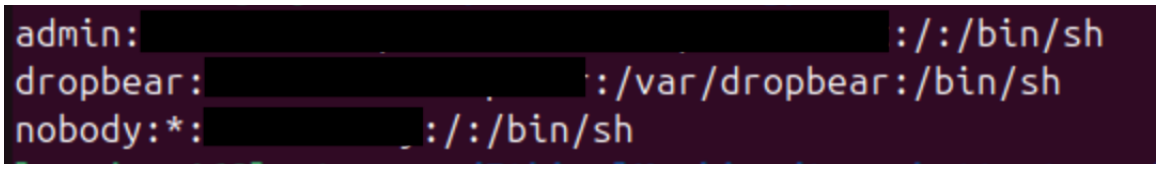
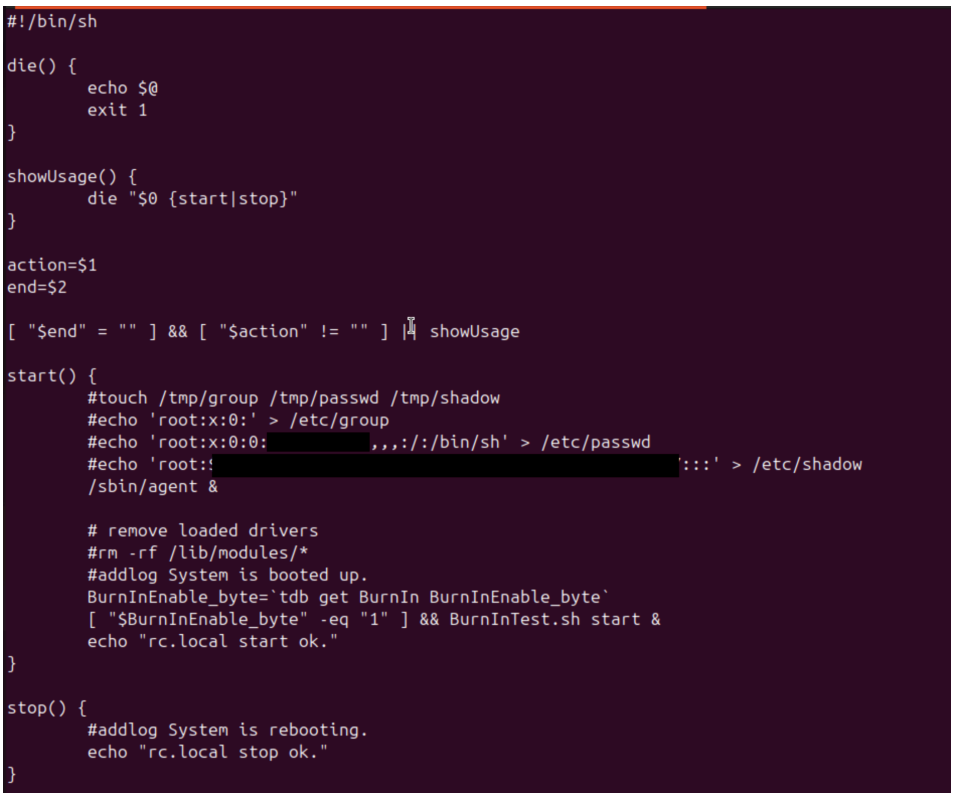


Figure 3.1.1: redacted hardcoded passwords for TPLink



```
#!/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

All devices share identical credentials, enabling trivial privilege escalation through Telnet or local console (CWE-259).

Mitigation

Remove static credentials; generate unique passwords on first boot and enforce SSH key-based login.

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

Evidence

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

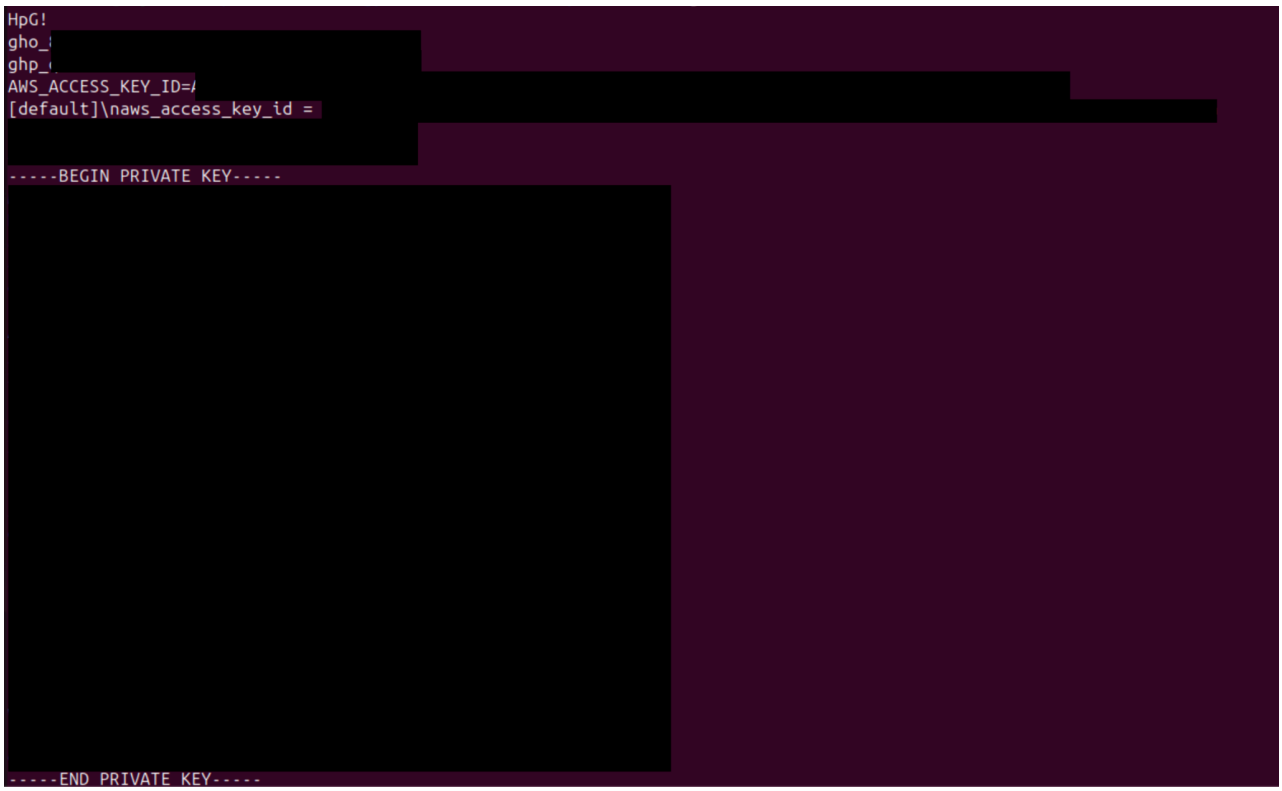


Figure 3.2.1: Redacted GitHub authorization tokens, AWS access keys, and a private key

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

Impact

Compromise of these keys permits device impersonation and backend access (CWE-321 / CWE-798).

Mitigation

Remove secrets from binaries; store per-device keys in secure elements (TPM / ATECC) and use short-lived IAM tokens.

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

Evidence

PEM files located under _dcs-8000lh.bin.extracted/_160000.extracted/db/verify.key and adjacent private key files.

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

Impact

Presence of signing or encryption keys in firmware allows attackers to sign rogue updates or decrypt traffic (CWE-320 / CWE-522).

Mitigation

Exclude private keys from production builds; store signing keys securely on build servers and verify updates with only public keys.

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

Evidence

strings /bin/busybox | grep BusyBox showed:

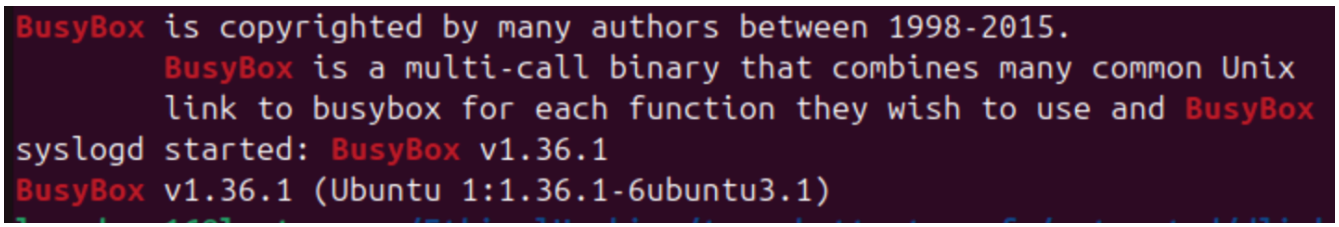


Figure 3.4.1: BusyBox version number

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

Malicious input to vulnerable applets could leak or overwrite data in maintenance scripts.

Mitigation

Upgrade to BusyBox ≥ 1.37.2 or disable affected applets until patched.

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

Evidence

Boot segments reference U-Boot 2013–2014 and network boot options (TFTP/NFS).

Linked CVE: CVE-2019-14192 (NFS overflow in U-Boot).

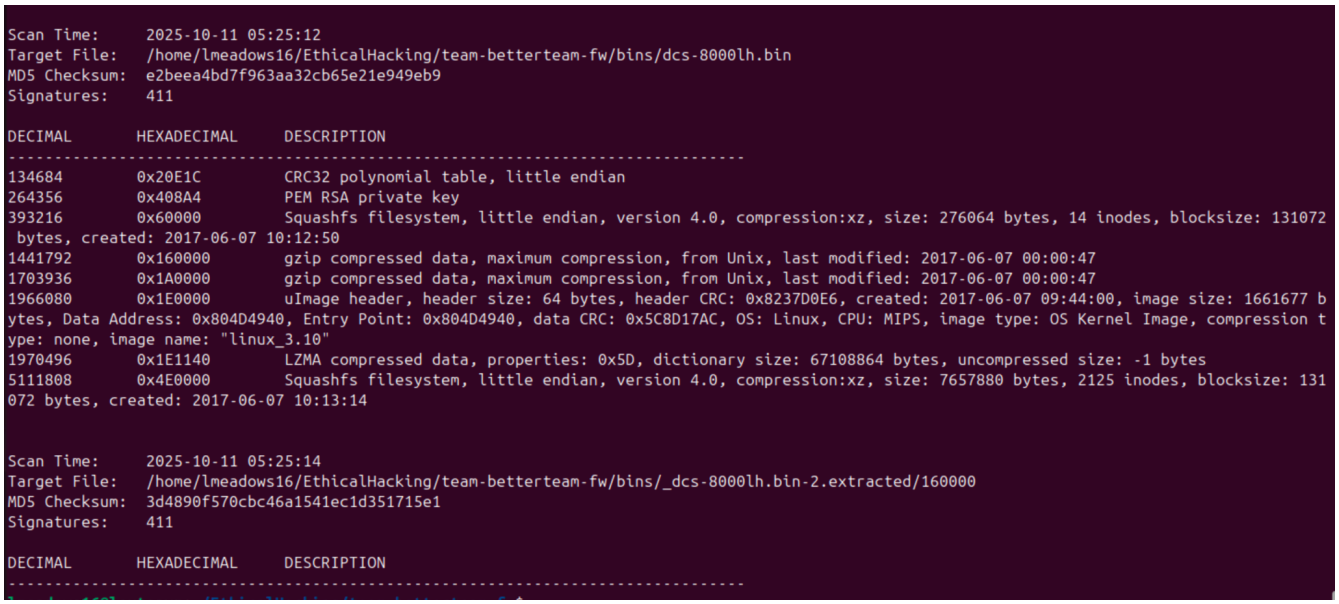


Figure 3.5.1: DLink preliminary binwalk showing Boots and TFTP

Impact

Unsigned or network-booted kernels can bypass verified-boot, enabling persistent compromise.

Mitigation

Update to U-Boot ≥ 2020.01; enforce FIT-image signature verification and disable TFTP/NFS boot in production.

4. Automation Script

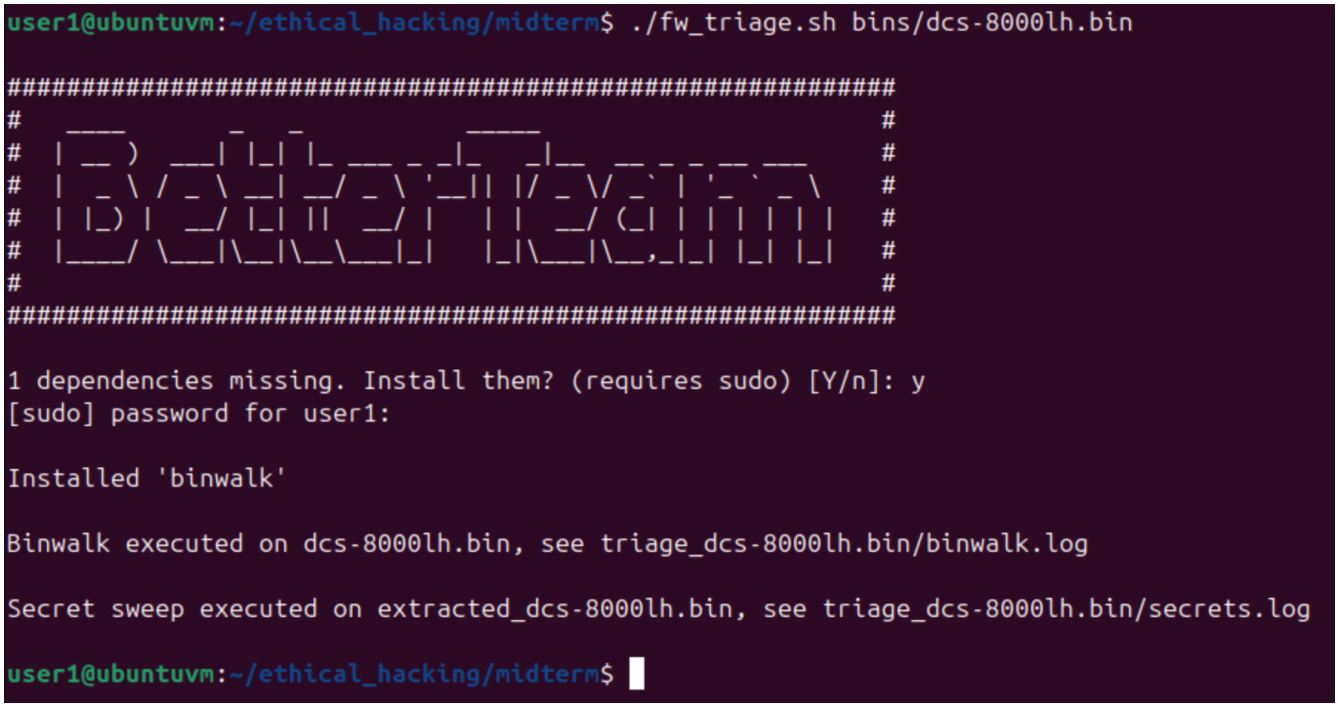


Figure 4.1.1: Script execution, including dependency installation

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 address the vulnerabilities identified in Section 3 and are prioritized by impact and feasibility. Each action directly mitigates a corresponding risk category observed in the TP-Link, D-Link, and bare-metal ELF firmware images.

5.1 Credential and Access Control

1. **Eliminate Hard-Coded Passwords** - Remove default credentials from boot or init scripts. Require per-device password initialization or SSH key-based authentication.
2. **Harden Authentication Services** -Disable Telnet and legacy web logins; restrict administrative access to SSH with key-pair login and rate-limiting (e.g., Dropbear or OpenSSH).
3. **Protect Password Storage** - Use salted SHA-512/bcrypt hashes in /etc/shadow; verify no plaintext passwords are echoed in startup logs.

5.2 Key and Secret Handling

1. **Remove Embedded Keys** - Strip PEM, PKCS#8, and AWS credentials from firmware binaries before release.

2. **Implement Secure Key Storage** - Store per-device keys in hardware security modules (TPM 2.0 / ATECC608) or inject them during manufacturing.
3. **Rotate Cloud Tokens Automatically** - Use short-lived IAM roles or IoT certificates instead of static access keys.

5.3 Bootloader and Firmware Integrity

1. **Enforce Verified Boot** - Adopt a modern U-Boot (≥ 2020.01) with FIT-image signature checks.
2. **Disable Network Boot in Production** - Remove TFTP / NFS boot options to prevent rogue kernel loading.
3. **Validate Firmware Updates** - Require signed update packages and checksum verification at runtime.

5.4 Component Hardening and Patch Management

1. **Update BusyBox and Libraries** - Upgrade to BusyBox v1.37.2 or later to patch CVE-2024-58251 and CVE-2025-46394; recompile with minimal enabled applets.
2. **Institute Routine Patch Cycles** - Integrate CVE monitoring and automated rebuilds into the vendor's CI/CD pipeline.

5.5 Filesystem and Build Hygiene

1. **Enforce File Permissions** - Set restrictive ownership for /db/, /etc/init.d/, and /etc/ssl/; mount secondary partitions with nosuid,noexec.
2. **Automate Secret Scanning** - Add tools such as truffleHog or git-secrets to pre-commit hooks.

5.6 Organizational Practices

1. **Formalize Vulnerability Management** - Track discovered issues, patches, and firmware revisions within a change-control system.
2. **Provide Customer Update Guidance** - Document update procedures and publish end-of-support timelines to improve transparency.

6. Conclusion

This assessment examined three firmware images provided by ThanosTech LLC, the TP-Link WR-841N, D-Link DCS-8000LH, and a bare-metal ARM ELF, to evaluate their security posture through static ethical-hacking techniques. Analysis in a controlled Ubuntu 22.04 LTS environment revealed recurring weaknesses in credential management, cryptographic key handling, and boot integrity, alongside component-level vulnerabilities in BusyBox v1.37.0.

Key findings included:

- Hard-coded credentials in startup scripts.
- Embedded private and cloud keys in compiled binaries.
- Legacy bootloaders lacking verified-boot enforcement.
- Two known BusyBox CVEs (2024-58251, 2025-46394).

Recommended mitigations focus on removing embedded secrets, upgrading vulnerable components, and enforcing signed, verified boot paths. Long-term improvements should embed secure-build automation, secret-scanning within CI/CD pipelines, and hardware-anchored key storage. Through this engagement, ThanosTech gains a reproducible framework for firmware analysis and a roadmap for secure product evolution—transforming reactive patching into proactive firmware security assurance.