Cyber Apocalypse 2026 — Hardware — A Fault in the Cinder

Flag: `HTB{a_f4ls3_n0t3_0p3ns_th3_tru3_s34l}`

Category: Hardware (fault injection / voltage glitching)

Bench: `154.57.164.81` — rig `:31274`, logic-analyzer `:32324`, recovery console `:32606` (ports rotate per instance)

---

1. Challenge

> *Keir's routes cut a Vaultrune confiscation convoy and lifted a Cinderbound custody coffer... Breaking the shell burns the record, so it must be recovered without forcing it: learn the ritual of authority the coffer expects of a Vaultrune auditor, forge a single instant of it, and listen as the old ward speaks the sealed word to a bearer it was never meant to trust.*

Translated out of the flavour text:

- A "coffer" (an ARM Cortex-M3 device) holds a sealed record. Physically forcing it wipes the record.
- To read it you must complete the **recovery-auditor authorization ritual** — but you don't have a valid signet.
- "**forge a single instant of it**" = a single-instruction **voltage glitch** during the authorization check.
- "**listen as the old ward speaks the sealed word**" = capture the secure-element (SE) bus with the logic analyzer after the glitch and read out the released record.

2. Getting the files

The handout `….zip` is **ZipCrypto**-encrypted (legacy, not AES). Because every file is genuinely Deflate-compressed there is no clean known-plaintext for `bkcrack`, so I extracted the hash with `zip2john` and cracked it — it turned out to be the standard HTB archive password:

```
$ zip2john handout.zip > zip.hash          # $pkzip2$...
$ john --wordlist=... --rules zip.hash
hackthebox        (handout.zip)
$ unzip -P hackthebox handout.zip
```

Contents:

File	What it is
`cinder_boot.elf`	The **boot ROM** (ARM Cortex-M3, Thumb-2, stripped). *This is the glitch target.*
`TARGET.pdf`	"Warden's Field Reference" datasheet — the full operating spec, including the timing model.
`rig.py`	Reference client for the three bench services (transport, capture download, signet console).
`factory_spi.sr`	Sigrok capture of the **external flash** bus (CS_FLASH/SCK/MOSI/MISO, 2 MHz).
`factory_service.sr`	Sigrok capture of the **secure-element** bus (CS_SE/SCK/MOSI/MISO/SE_IRQ, 8 MHz) — a full *authorized* reference session.

3. The target, from the datasheet

- **Core:** ARM Cortex-M3, 24 MHz. Internal SPI bus (flash + secure element), mode 0, MSB-first.
- **Boot sequence** on a recovery strap:
- read strap; 2. read & verify image manifests from flash;
- wake the ward (secure element) — `STARTUP`, `GET_STATUS`, `READ_NV_COUNTER`;
- **low-counter recovery:** `GET_RECOVERY_CHALLENGE`, then `VERIFY_RECOVERY_TICKET`. Without a valid signet this returns **`AUTH_DENIED`** and the boot halts.
- on authorization: `EXTEND_BOOT_PCR`, `LOCK_MEASUREMENT`;

- launch: `OPEN_FACTORY_SESSION`, `UNSEAL`, `READ_FIFO` — the sealed record is released.
- **Signet (recovery ticket):** 144 bytes, magic `RCVT`, version 1, ECDSA-P256 signed. The spine only forwards a *well-formed* envelope (right magic/version/length) to the ward; the ward then checks UID / image digest / counter / challenge / signature. Any failure is the same opaque `AUTH_DENIED`.
- **Fault rig (the digital trigger):** watches the ward's response bytes for a masked pattern and, **on the `CS_SE` deassertion edge that ends the matched response**, fires a single pulse that omits exactly one instruction after a programmable delay.
- **Timing model (this is the crux — given as a working spec):**

> *delay is measured in CPU cycles from the triggering `CS_SE` rising edge... From the `AUTH_DENIED` `CS_SE` edge the spine returns from ticket verification into the boot flow in a fixed **`D_drv = 221`** cycles. The delay is `D_drv` plus the cycles you count, under the cost table, from the first instruction after ticket verification returns up to — but not including — the fetch of the instruction you intend to disturb.*

Cost table: data-processing / `it` / `ite` = 1, load/store = 2, taken branch = 3, `bl` = 4, not-taken conditional branch = 1. Width calibration band = 6..16 ticks.

4. Finding the instruction to skip (disassembly)

Disassembling `cinder_boot.elf` (`.text` at `0x8`, Thumb) and tracing `main`:

- Ticket verification is `bl #0x2c2`, which wraps SE opcode `0x13` `VERIFY_RECOVERY_TICKET` and returns **`r0 = 0` (authorized) / `1` (denied)**.
- Verification returns to `0x00c6`. The auth result `r0` survives untouched until it is folded into the final decision:

```
0x010e: cmp   r0, #0          ; r0 = ticket result (1 = denied)
0x0110: it    eq
0x0112: orreq r2, r2, #1      ; only sets the "pass" bit if r0 == 0 (authorized)
0x0116: cmp   r2, #0
0x0118: beq   #0x30           ; <-- if r2 == 0 -> fail handler (halt).  SKIP THIS.
0x011a: ...                   ; success path: EXTEND_BOOT_PCR / UNSEAL / launch
```

For a denied ticket, `r2` stays `0` and `0x0118 beq` jumps to the fail handler at `0x30`. **Glitching out the `beq` at `0x0118`** makes execution fall straight through into the authorized path.

Cycle count `K` (0x00c6 → just before 0x0118)

The only data-dependent part is a spin loop at `0xd4` / `0xe4`:

```
0xc6 ldrb 2 | 0xca ldrb 2 | 0xce eors 1 | 0xd0 and 1              = 6
0xd4 cbnz + loop body (nop+subs+b) ...                          = 8*n + 1
0xd6..0x116 (subs/ldr/it/cmp/beq/b/ite legs, denied path)       = 26
```

where the loop runs `n = (challenge[0] ^ challenge[15]) & 0x1f` times (the challenge buffer sits at `sp+0xc..sp+0x1b`). Total:

```
K = 6 + (8n + 1) + 26 = 33 + 8n
delay = D_drv + K = 221 + 33 + 8n = 254 + 8n   (CPU cycles)
```

The recovery **challenge is known ahead of the boot** — the console offers it (`CHALLENGE …`) and the ward commits to it for the next power cycle — so `n`, and therefore the exact delay, is computable per attempt.

5. The attack

The bench protocol (from `rig.py`):

- **Recovery console (`:32606`):** connect, drain the offer (`DEVICE/IMAGE/COUNTER/CHALLENGE/TICKET?`), present the signet with `TICKET <base64>`.
- **Rig (`:31274`):** `STRAP RECOVERY`, `RESET RIG`, `TRIGGER BUS SE PATTERN <hex> MASK <hex> EDGE CS_RISE`, `GLITCH DELAY <d> WIDTH <w>`, `CAPTURE ARM INTERNAL_SPI`, then `POWER CYCLE` (rate-limited).
- **Logic analyzer (`:32324`):** `GET cap-<id> VCD.GZ` → a `CINDCAP1`/gzip envelope containing the VCD.

Steps per attempt:

1. Present a **well-formed but invalid** signet: 144 bytes, `RCVT` + version `01` + flags `00`, rest zeros. The boot ROM accepts the envelope and forwards it; the ward denies it →

`AUTH_DENIED` (our trigger).

2. **Trigger** on the `VERIFY_RECOVERY_TICKET` response: `PATTERN 13 MASK ff` (opcode `0x13` uniquely identifies that frame; the unpredictable sequence byte is masked out). The rig fires on the `CS_SE` deassertion ending it.
3. **Glitch** at `DELAY = 254 + 8n`, `WIDTH 10`.
4. Power-cycle, download the capture, decode the SE bus.

```
n     = (chal[0] ^ chal[15]) & 0x1f
delay = 254 + 8*n
```

This bypassed `AUTH_DENIED` **on the first attempt**. The decoded capture showed the full authorized flow — `STARTUP, GET_STATUS, READ_NV_COUNTER, GET_STATUS, GET_RECOVERY_CHALLENGE, VERIFY_RECOVERY_TICKET(status 0x01=denied!), EXTEND_BOOT_PCR, LOCK_MEASUREMENT, OPEN_FACTORY_SESSION, UNSEAL, GET_STATUS, READ_FIFO×19` — proving the ward denied the ticket yet the boot proceeded anyway. The 19 `READ_FIFO` responses concatenate to a **582-byte encrypted custody record**.

The full attack driver is `glitch_attack.py`.

6. Reading the "sealed word" (record decryption)

`READ_FIFO` returns *protected* bytes — the record is encrypted under the factory session, not plaintext on the wire. Recovering it required reversing the **recovery-ward firmware**, which I pulled straight out of `factory_spi.sr` (flash image 2 @ `0x040000`, a `CNDR` manifest + Thumb code that loads to `0x20002000`).

Reversing the SHA-256 / HMAC routines in that image (and confirming every step against the on-bus HMAC tags):

- **Baked secret** (HMAC key), 32 bytes @ `0x20002a8e` in the firmware, just after the `recovery-ward` string: `81a68ed89ebce0bbb7d5f4f08da9d598c3d0ffb0e7e583bdfe8cd980e6ab878b`
- **Boot measurement:** `M = SHA256("CINDER-BOOT-v1" || slot_kind || counter_le32 || payload_size_le32 || image_digest)`, then `PCR = SHA256(0x00*32 || M)`.
- **Session key:** `HMAC_SHA256(secret, MIX8 || token(16) || swap32_words(PCR) || host_nonce(16))`, where `MIX8` is derived from two baked constants (`0x7f4a7c15`, `0x9e3779b9`).
- **Per chunk `i` (≤32 B):**

- tag `= HMAC(session_key, session_id || i || ciphertext)[:16]` (matches the on-bus tag),
- keystream `= HMAC(session_key, "RESPONSE" || session_id || i || 0x00000000)[:len]`,
- `plaintext = ciphertext XOR keystream`.

Decrypting the live record (`decrypt_record.py`) yields the custody-succession record for *House Suncourt* — a "Writ of the Black Heir, under Cassian's own hand" ordering the house struck from the registry — with the flag embedded:

```
HTB{a_f4ls3_n0t3_0p3ns_th3_tru3_s34l}
```

("a false note opens the true seal" — the forged instant of authority that lets the ward speak.)

7. Artifacts

File	Purpose
`glitch_attack.py`	End-to-end: presents signet, computes `delay = 254 + 8n`, arms the rig, power-cycles, downloads + decodes the capture.
`vcd_decode.py`	VCD → SE-bus SPI transaction decoder.
`decrypt_record.py`	Firmware-derived session-key KDF + per-chunk HMAC keystream record decryptor.
`success.vcd`	The winning glitched-boot capture (full authorized flow + 19 `READ_FIFO` chunks).

8. Takeaways

- A single opaque `AUTH_DENIED` and an ECDSA-signed signet are worthless if the **one branch** that acts on the verdict can be skipped by a fault. The decision must be redundant / fault-hardened, not a lone `beq`.
- The datasheet handed out an exact cycle-cost model; the intended solve is to **derive** the delay (`254 + 8n`) from the disassembly rather than blindly sweep — the challenge title's "single instant."

- Encrypting the released record on the bus is good defence-in-depth, but the session KDF's shared secret lived in readable firmware, so the "sealed word" was recoverable once the record was dumped.