

Vulnerability Assessment Report

Dark-Dev-Proof/darkproof

November 23, 2026 — Whitebox Scan

1	8	4
MEDIUM	LOW	INFO

CONTENTS

Table of Contents

Executive Summary 3

Recommendations 4

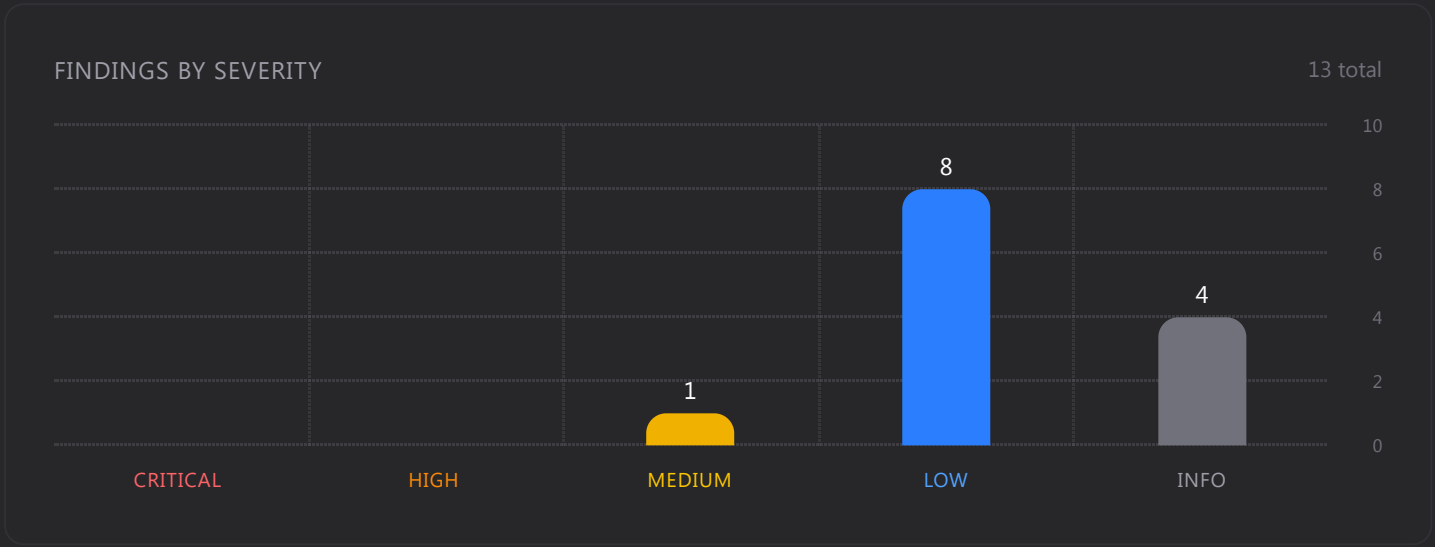
Detailed Findings 6

- Authentication 6
- Other 10
- Authorization 18

SECTION 01

Executive Summary

This assessment of upload/darkproof-whitebox-uploads/cmszzidtl0000428x7pch7etp/bbdfa780-7f09-48ff-8b1c-6d229be86035.zip identified 13 findings across 3 categories. No critical-severity issues were found.



SEVERITY DEFINITIONS

- CRITICAL — Trivially or remotely exploitable with severe impact — data breach, account or fund takeover, remote code execution. Fix immediately.
- HIGH — A serious, exploitable weakness with significant impact that a motivated attacker would likely abuse. Fix urgently.
- MEDIUM — A real weakness with limited impact or one that needs specific conditions or authentication to exploit. Fix soon.
- LOW — A minor issue or missing hardening with little direct impact on its own. Address opportunistically.
- INFO — An informational observation or best-practice note with no direct security impact.

METHODOLOGY

Source-driven (white-box) assessment with full read access to the application's code. The scanner clones the repository at HEAD and builds a static call-graph with taint analysis to trace untrusted input into security-sensitive sinks (queries, auth checks, access-control rules, external calls). An LLM triages each reachable path to separate real vulnerabilities from noise; backend-specific runtime probes then confirm configuration and access-control issues against the live deployment, and exploitable paths are proven end-to-end with real requests. Every confirmed finding is de-duplicated to a single root cause and scored with CVSS. This examines logic and data-flow a black-box scan cannot see from the outside — the reason white-box surfaces materially more, and more deeply proven, issues.

FINAL SECTION

Recommendations

Prioritized by severity — address critical and high items first.

MEDIUM

Phishing chain: a look-alike group name, a join prompt that hides the pinned admins, and free display names talk a user into the attacker's room

Show the code and the pinned admin count in the prompt on both apps ("3 admins named by this link"), so an invite is verifiable at decision time. Normalise invite names with `Vouch.key` before storing and before the twin comparison. Add the look-alike warning to iOS. Ignore a repeat link for the code already pending.

LOW

Dark-room holder check is free to pass and cheap to starve: any throwaway wallet buys an RPC call, and ~40 requests take the gate down

Put the challenge mint behind the same (or its own) bucket. Cache the verdict per address for ~60 s, `not-holder` included. Single-flight the RPC so N callers cost one call. Return `Retry-After` and keep upstream-429 handling explicit instead of folding it into a generic failure.

LOW

The repository is behind production: six live files are absent from the tree, and three audited findings were already fixed live

Deploy from the audited commit, or push the deployed files back to the repository. Add the site to the CI so the tree cannot drift from what ships.

LOW

Eight connections from one address take the whole Wi-Fi board down, silently - the refusal closes the socket with no status line

On a refusal write `HTTP/1.1 503` with `Retry-After: 5` and the one-line JSON the board already uses, then close, so the page can say "the board is busy" instead of failing like a dropped network.

LOW

Board header read copies the whole buffer once per byte - 16x CPU, ~33 MB of churn per 8 KiB header

Only rebuild the buffer when a newline arrives:

if (c != '\n'.code) continue val a = b.toByteArray(); val n = a.size

Better still, scan the internal array for the terminator and avoid the copy entirely.

LOW

`/messages` is unmetered and recompiles two regexes per word - ~20 000 Pattern compiles per request

Hoist the three patterns to `private val` object-level constants. Cache the rendered JSON briefly. Add `?since=<ts>` so a poll only pays for what it has not already seen.



LOW

A malformed percent escape in a board post aborts the response - the socket closes with nothing sent

Wrap the decode (`runCatching` → raw string) or catch in `handle` and `respond(s, 400, …)` . The response is already JSON everywhere else, so `{"ok":false,"error":"Malformed message."}` fits.

LOW

iOS peer-to-peer accepts every invitation and never evicts, so neighbouring ids can occupy all of MCSession's peers

Mirror `LinkLimits` : cap the session's peers, evict an un-HELLOed or random link on overflow, and key `found` by `MCPeerID` rather than display name.

LOW

Chain: anyone on the Wi-Fi can write into Public through the board, signed by the host's phone, and a block does not stop it

Strip the `web-` prefix before saving and before comparing a block, or store both keys. State on the board page and in the docs that a board post is the host's signature over a stranger's words.

INFO

Relay operators learn more than the blurred date implies: exact send times, a hop count, and offline confirmation of a short code

Documentation only: say so in "Who sees what" and prefer the generated code form. Optionally round the inner ts to the same 1024 s window on the relay path.

SECTION 03

Authentication 3 findings

MEDIUM

Phishing chain: a look-alike group name, a join prompt that hides the pinned admins, and free display names talk a user into the attacker's room

90% Confidence

5.4 CVSS

`app/src/main/java/it/blackout/mesh/App.kt:84 · Protocol.kt:642`

CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:L/I:L/A:N

Three informational weaknesses compose into a confidentiality break in a product whose only door is the group code.

1. Group names arriving in an invite are stored without normalisation (no NFKC, no bidi strip), while *person* names get exactly that treatment (`Vouch.key`). A name carrying a Cyrillic homoglyph, a zero-width space or U+202E renders as a twin of the user's real group while the comparison calls it different.
2. `MainActivity` is exported and has a BROWSABLE `darkproof://join` filter, so any installed app or web page can raise the join prompt at any moment, and the prompt shows only the decorative name - never the code, never the pinned admins. The pins are the trust anchor the whole design leans on (a first epoch is accepted only from a pinned admin), and nothing shows them to the person being asked; they appear only in the inviter's own sheet.
3. The Public channel lets anyone sign as any display name, so the attacker's room looks populated by the right people.

The two apps fail in opposite directions: Android warns about the look-alike but hides the code; iOS shows the code but has no look-alike warning at all and asks only "Join <name>".

End to end: attacker builds a group with a twin name, posts the link, the user taps Join, the look-alike becomes the current group, the real group's messages stop appearing, and everything written afterwards goes to the attacker.

EVIDENCE

Verified by reading both apps and comparing them against each other:

- `Groups.swift:34` builds the name with `String((name.isEmpty ? "Group" : name).prefix(maxGroupName))` - no normalisation, same as Kotlin `Protocol.kt:642` .
- `Protocol.kt:170–191` contains the normaliser that *could* be used (`Vouch.key` : NFKC, case fold, Default_Ignorable strip) and is applied to person names only.
- `App.kt:84–88` is the Android twin warning; `Views.swift:80–82` is the iOS first-run prompt, which shows nothing but `Join <name>` .
- `MainActivity.kt:136–139` sets `state.invite` from any VIEW intent, last-write-wins, no debounce.
- Verified on an emulator pair that a `darkproof://join?name=…&code=…` link raises the prompt and joins in one tap.

REPRODUCTION

1. Attacker: create a group, name it with a homoglyph twin of the target's group, e.g. "Family" (Cyrillic i).
2. Attacker: invite by link and hand it over, or fire it directly: adb shell am start -a android.intent.action.VIEW -d "darkproof://join?name=Fam%C3%ACly&code=<attacker-code>&admins=<attacker-pin>"
3. Victim taps Join. The room appears in the list with a familiar name; the real group stops being current.
4. Victim writes. The attacker reads it and can answer under any colleague's name.



MEDIUM

Phishing chain: a look-alike group name, a join prompt that hides the pinned admins, and free display names talk a user into the attacker's room

continued

CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:L/I:L/A:N

REMEDIATION

Show the code and the pinned admin count in the prompt on both apps ("3 admins named by this link"), so an invite is verifiable at decision time. Normalise invite names with `Vouch.key` before storing and before the twin comparison. Add the look-alike warning to iOS. Ignore a repeat link for the code already pending.



INFO

The join prompt can be pushed at any moment and never shows the pinned admins

90% Confidence
0.0 CVSS

`app/src/main/java/it/blackout/mesh/MainActivity.kt:136 · App.kt:82`

`MainActivity` is exported with a BROWSABLE `darkproof://join` filter, so any app or page can raise an invite whenever it likes, including while the user is reading a chat; `state.invite` is last-write-wins with no debounce. The prompt then shows the decorative name only.

Those pins are the trust anchor the design leans on - a first epoch is accepted only from a pinned admin - but nothing shows them to the person joining. They appear only in the inviter's own sheet, as the input the link is built from. Mitigations that do exist: the code lives in the URL fragment so it never reaches a server, a same-name/different-code group raises a warning, and an invite for the group in use is a no-op.

EVIDENCE

`MainActivity.kt:136–139` sets `state.invite` from any VIEW intent. `App.kt:684–688` builds the link from the admins; the Join sheet renders only `g.name`. Verified on an emulator: an `am start –a VIEW –d darkproof://join?…` raises the prompt.

REPRODUCTION

adb shell am start -a android.intent.action.VIEW -d "darkproof://join?name=X&code=Y" while the app is open.

REMEDIATION

Show the code and the pinned admin count in the prompt; ignore a repeat link for the code already pending.



INFO

Group names from invites are not normalised on either app, while person names are

85% Confidence
0.0 CVSS

`app/src/main/java/it/blackout/mesh/Protocol.kt:642` · `Groups.swift:34`

Person names get NFKC, case folding and Default_Ignorable stripping precisely so a zero-width space cannot make one name pass for another (`Vouch.key`). Invite group names get none of it: they are trimmed, defaulted and truncated.

A name carrying a Cyrillic homoglyph, a zero-width space or U+202E therefore renders as a twin while comparing as different. On Android that makes the look-alike warning fire with the same benign wording a legitimate second group gets; on iOS there is no warning at all.

EVIDENCE

`Protocol.kt:170–191` is the normaliser, applied to person names only. `Protocol.kt:642` and `Groups.swift:34` store the invite name raw. `App.kt:84` compares with `equals(ignoreCase = true)` and nothing else.

REPRODUCTION

Join a link whose name is `Family` (Cyrillic i) against an existing `Family` : the name renders identically, the code differs, and the warning reads as it would for any second group.

REMEDIATION

Normalise with the existing `Vouch.key` helper (or at least NFKC plus a bidi strip) before storing and before comparing.

SECTION 04

Other 8 findings

LOW

Dark-room holder check is free to pass and cheap to starve: any throwaway wallet buys an RPC call, and ~40 requests take the gate down

95% Confidence
3.7 CVSS

`site/holder.php`

CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:L

`holder.php` gates the browser assistant behind a wallet signature and then spends one outbound `getTokenAccountsByOwner` call on a public Solana node. Nothing ties the request to the coin: a throwaway keypair signing the page's own message shape passes verification, and the wallet needs to hold nothing.

The audited revision has no rate limit, no cache and no single-flight, and maps every upstream non-200 to `node-error` ("The Solana node did not answer"). Production has since added a 429 limiter - but the effective capacity is unchanged at about nine callers per burst, so ~78% of a burst still fails, now with an honest "too many checks" message. `GET holder.php?challenge`, the cheap half of the flow, is not covered by that limiter at all: 30 concurrent mints all returned 200.

EVIDENCE

Lab, real Apache plus a counting mock RPC: 40 forged requests produced exactly 40 outbound calls.

Against production, 40 concurrent correctly-forged requests: 31 HTTP 429 busy 9 HTTP 200 not-holder 30 concurrent `GET holder.php?challenge` - 30/30 HTTP 200, unmetered. Replaying one used body answers `{"state":"stale"}` against the node's reach (no RPC spent).

REPRODUCTION

See the shipped harness in this repo: `forge_live.py` mints a body for a fresh empty wallet, `hammer_live.py 40` fires 40 at once and tallies the answers.

python3 forge_live.py > live.json curl -s -H 'Content-Type: application/json' --data @live.json https://darkproof.app/holder.php

{"state":"not-holder"} - signature accepted for a wallet holding nothing

python3 hammer_live.py 40

31 x 429 busy / 9 x 200

REMEDIATION

Put the challenge mint behind the same (or its own) bucket. Cache the verdict per address for ~60 s, `not-holder` included. Single-flight the RPC so N callers cost one call. Return `Retry-After` and keep upstream-429 handling explicit instead of folding it into a generic failure.



LOW

The repository is behind production: six live files are absent from the tree, and three audited findings were already fixed live

99% Confidence
3.1 CVSS

`site/`

CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:L/A:N

The deployed site is a different, newer program than the audited commit. `holder.js`, `holder-lib.php`, `notes.php`, `notes.js`, `coin.html` and `coin` are all served from darkproof.app and none of them exist in the repository. The holder flow was rebuilt around a one-time challenge (`GET holder.php?challenge` → `{number, time}`) with a `Site:` binding on the message.

Consequences: an audit against the default branch reports already-fixed issues as open (three were re-graded here); the shared verifier `holder-lib.php` is unreachable by fetch (403) so it cannot be read at all; and there is no single artefact that describes what is running.

EVIDENCE

`ls site/` in the clone has no coin/notes/holder.js. All six fetch 200 from production; `darkroom.js` on production is a refactor that delegates to `window.dpHolder`. Diffing the two `darkroom.js` files shows the wallet block removed and replaced by `holder.js` calls.

Replaying a used body: `{"state":"stale"}` - behaviour the audited revision does not have.

`curl -i https://darkproof.app/holder-lib.php` → 403; `holder-lib.ph%70` → 403.

REPRODUCTION

curl -s -o /dev/null -w '%{http_code}\n' https://darkproof.app/holder-lib.php # 403 for f in holder.js notes.js coin.js coin.html notes.php; do curl -s -o /dev/null -w "$f %{http_code}\n" https://darkproof.app/$f; done # all 200 ls site/ | grep -E 'holder.js|notes|coin' # nothing

REMEDIATION

Deploy from the audited commit, or push the deployed files back to the repository. Add the site to the CI so the tree cannot drift from what ships.

LOW

Eight connections from one address take the whole Wi-Fi board down, silently - the refusal closes the socket with no status line

95% Confidence
3.1 CVSS

`app/src/main/java/it/blackout/mesh/WifiBoard.kt:137`

CVSS:3.1/AV:A/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:L

The board's per-address allowance is enforced in the accept loop by closing the socket rather than answering it. `door` allows eight per address and the worker pool is eight, so eight held connections from one visitor - eight idle browser tabs is enough - exhaust both. Every other visitor is then accepted at TCP level and closed with no status line and no body, so their page spins and the browser reports a failed load with nothing to act on. A 30 s watchdog does eventually reap idle sockets, but nothing explains the refusal.

EVIDENCE

On an Android 14 emulator with the board started, nine "held" connections opened from one address (headers sent, no terminator, kept open):

GET /messages (9th connection) -> curl code 000, empty reply POST /send (10th connection) -> curl code 000, empty reply

With no held connections both answer normally (`200` and `{"ok":true}`).

REPRODUCTION

for i in $(seq 1 9); do (printf 'GET /messages HTTP/1.1\r\nHost: x\r\n'; sleep 12) | nc <phone>:8080 >/dev/null & done curl -s -o /dev/null -w '%{http_code}\n' http://<phone>:8080/messages # 000

REMEDIATION

On a refusal write `HTTP/1.1 503` with `Retry-After: 5` and the one-line JSON the board already uses, then close, so the page can say "the board is busy" instead of failing like a dropped network.

LOW

Board header read copies the whole buffer once per byte - 16x CPU, ~33 MB of churn per 8 KiB header

90% Confidence
3.1 CVSS

`app/src/main/java/it/blackout/mesh/WifiBoard.kt:192`

CVSS:3.1/AV:A/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:L

`readHead` accumulates one byte at a time and rebuilds the whole buffer after every byte to test for the terminating CRLFCRLF. An 8 KiB header is ~8 192 full copies, about 33 MB of allocation and copying per request, and the door allows eight concurrent connections from one address so a single visitor can hold ~270 MB of churn in flight.

Measured cost of the loop alone: 1 230 us per 8 KiB header; checking only after a newline answers the same question for 78 us, a 16x cut. End-to-end latency stays flat on fast hardware (0.86 s vs 0.84-0.90 s per 150 requests on an emulator), so this is a CPU and battery cost under concurrency rather than a response-time one.

EVIDENCE

JVM benchmark of the exact loop, 8 146-byte request, 200 iterations: 1 230 us/request as shipped, 78 us with the `\n`-only check.

On-device (emulator): 150 sequential requests took 0.86 s with short headers and 0.84-0.90 s with 8 KiB headers - inside the noise.

REPRODUCTION

Build an 8 KiB request and time it against a phone running the board; or run the loop standalone, which reproduces the 16x in milliseconds.

REMEDIATION

Only rebuild the buffer when a newline arrives:

if (c != '\n'.code) continue val a = b.toByteArray(); val n = a.size

Better still, scan the internal array for the terminator and avoid the copy entirely.



LOW

`/messages` is unmetered and recompiles two regexes per word - ~20 000 Pattern compiles per request

85% Confidence
3.1 CVSS

`app/src/main/java/it/blackout/mesh/WifiBoard.kt:185` · `WordFilter.kt:20`

CVSS:3.1/AV:A/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:L

`GET /messages` has no rate limit beyond the eight-per-address connection door, and each call runs `WordFilter.clean` over up to 100 messages x 2 fields. Every word is passed through `fold`, which calls `Regex(...)` per invocation, and a stretched word compiles up to three more.

A 500-character message of ~100 words is 200-400 compiles; one hundred messages is 20 000+ per HTTP request plus a full message-list snapshot. Measured at about 4 ms on a desktop, so tens of milliseconds of CPU per request on a phone - and the page polls every 4 s by itself.

EVIDENCE

`messages()` maps the snapshot through `WordFilter.clean` for name and text. `clean` splits on whitespace and calls `offensive` (up to 3 `Regex(...)` constructions) and `fold` (`Regex("\\p{Mn}+")`) per token.

40 000 compiles of the two hot patterns measured at 8 ms on a desktop JVM.

REPRODUCTION

Hold the board page open and poll: for i in $(seq 1 200); do curl -s -o /dev/null http://<phone>:8080/messages; done - every request re-filters every stored message from scratch.

REMEDIATION

Hoist the three patterns to `private val` object-level constants. Cache the rendered JSON briefly. Add `?since=<ts>` so a poll only pays for what it has not already seen.



LOW

A malformed percent escape in a board post aborts the response - the socket closes with nothing sent

95% Confidence
2.7 CVSS

`app/src/main/java/it/blackout/mesh/WifiBoard.kt:230`

CVSS:3.1/AV:A/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:L

`URLDecoder.decode` throws `IllegalArgumentException` on a bad escape such as `text=%zz`. The throw escapes `post()` into `handle`'s `catch (_: Exception)`, which has already written nothing, so the socket closes and the browser sits until its own timeout with no server-side clue. Every other path answers JSON.

EVIDENCE

On an emulator with the board running:

curl -s -i -d 'name=Ann&text=%zz' http://<phone>:8080/send -> nothing at all (no status line, no headers) curl -s -i -d 'name=Ann&text=hello' http://<phone>:8080/send -> 200 {"ok":true}

REPRODUCTION

curl -i -d 'name=A&text=%zz' -m 10 http://<phone>:8080/send ; echo "[exit $?]"

REMEDIATION

Wrap the decode (`runCatching` → raw string) or catch in `handle` and `respond(s, 400, …)`. The response is already JSON everywhere else, so `{"ok":false,"error":"Malformed message."}` fits.



LOW

iOS peer-to-peer accepts every invitation and never evicts, so neighbouring ids can occupy all of MCSession's peers

85% Confidence
3.1 CVSS

`ios/Blackout/MultipeerLink.swift:80`

CVSS:3.1/AV:A/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:L

`MultipeerLink` calls `invitationHandler(context == Self.context, session)` unconditionally: a matching context keeps other apps out, and payloads are bounds-checked afterwards by `Packet.decode`, so this is about slots rather than parsing. But there is no link cap, no per-peer share and no eviction. `MCSession` tops out at about eight peers, so roughly eight hostile `MCPeerID`s in range take every slot and no further iPhone can join; nothing ages out. `found` is also keyed by display name, so an attacker who reuses a victim's node id shadows the victim in the retry map.

The Android BLE path solves exactly this - six slots, stale un-HELLOed first, then a random link dropped at most every 30 s - and the Swift BLE link already implements the same policy, so `MultipeerLink` is the one transport left without it.

EVIDENCE

`MultipeerLink.swift:80–82` - unconditional accept. No `maxLinks`, no `makeRoom`, no rotation timer in the file, while `BleLink.swift:110–120` has all three and `WifiLink.swift:403` calls `LinkLimits.giveWay`. `found[peerID.displayName] = peerID` keys the retry map on an attacker-chosen string.

REPRODUCTION

Advertise many peer ids from one hostile iPhone (each `MCPeerID` with a new displayName) and invite repeatedly; after ~8 the phone accepts and cannot hold a real peer.

REMEDIATION

Mirror `LinkLimits`: cap the session's peers, evict an un-HELLOed or random link on overflow, and key `found` by `MCPeerID` rather than display name.

INFO

Relay operators learn more than the blurred date implies: exact send times, a hop count, and offline confirmation of a short code

80% Confidence
0.0 CVSS

`app/src/main/java/it/blackout/mesh/Core.kt (realTime/hideTime) · Nostr.kt:43`

The spec is honest that relays read id, ts and ttl, and `created_at` is deliberately rounded to ~17 minutes so events do not date themselves. Three things survive:

1. The inner header timestamp is exact and reconstructible - it is hidden only with the public constants in the spec, and `GroupCrypto.realTime` inverts it in eight lines. The blur buys nothing against a relay that base64-decodes the content.
2. The ttl is cleartext and decrements per hop, so `64 - ttl` is a distance oracle: 1 for the sender's own phone, more for anything that crossed the mesh.
3. The mailbox tag is derived from the code, so a captured tag can be tested against candidate codes offline at roughly one scrypt each - generated 80-bit codes are safe, a 4-6 character custom code is not.

None of it leaks authorship: the sender never appears in cleartext, no NIP-42 AUTH is sent, and each group rides its own Tor circuit.

EVIDENCE

`hideTime` is `(ts and mask.inv()) - WINDOW + ((ts % W) xor mask)` with `mask` and `W` in spec/FORMAT.md; `realTime` inverts it. `Packet.encode` writes the ttl as a cleartext header byte. `deriveMailbox` is `HMAC-SHA256(groupKey, "darkproof-relay-mailbox-v1")` truncated to 16 bits, and the group key is scrypt(code).

REPRODUCTION

Take any published event, base64-decode `content`, read bytes 8..15 as the ts and byte 17 as the ttl, then apply `realTime` from the spec. For a code guess, derive the key with scrypt and compare the first two mailbox bytes with the event's `t` tag.

REMEDIATION

Documentation only: say so in "Who sees what" and prefer the generated code form. Optionally round the inner ts to the same 1024 s window on the relay path.

SECTION 05

Authorization 2 findings

LOW

Chain: anyone on the Wi-Fi can write into Public through the board, signed by the host's phone, and a block does not stop it

90% Confidence
3.1 CVSS

`app/src/main/java/it/blackout/mesh/MeshEngine.kt:2118 · Moderation.kt:26`

CVSS:3.1/AV:A/AC:L/PR:N/UI:N/S:U/C:N/I:L/A:N

`/send` needs nothing but being on the same network prefix while the board is on. The post is sealed and signed by the *host* phone under `web-<host id>`, published to the Public channel and relayed on to the internet where it lives for seven days, with the display name as free text.

So a passer-by puts words in front of every member, near and far, carrying the host's own signature. A block does not silence them: a block saved from a direct post misses `web-<id>` and vice versa, so the same human under two ids slips past. It stops only when the board is switched off or the host leaves that network; the board's own limit of 10 posts a minute caps the volume.

Confirmed end to end: a post from an unauthenticated client appeared in the feed as "Auditor (Wi-Fi board)", signed, ready to relay.

EVIDENCE

`curl -d 'name=Auditor&text=end-to-end proof 4711' http://<phone>:8080/send` → `{"ok":true}`, then the feed showed `{'t': '20:40', 'n': 'Auditor (Wi-Fi board)', 'x': 'end-to-end proof 4711'}`.

`MeshEngine.post` uses `web-${k.me}` as the wire sender; `Moderation.block` stores `Blocks.key(channel, m.sender)` and `Blocks.hides` compares exactly, so `everyone|web-<id>` and `everyone|<id>` never meet.

REPRODUCTION

1. Stand on the same network as a phone with its board on.
2. curl -d 'name=<any colleague>&text=<anything>' http://<phone>:8080/send
3. The message is signed by the host and relayed onward; blocking the host does not remove it.

REMEDIATION

Strip the `web-` prefix before saving and before comparing a block, or store both keys. State on the board page and in the docs that a board post is the host's signature over a stranger's words.

INFO

A block does not cover Wi-Fi board posts, because they arrive under a different sender id

90% Confidence
0.0 CVSS

`app/src/main/java/it/blackout/mesh/Moderation.kt:26 · Protocol.kt:396`

A post typed on someone's board is signed and sent under `web-<host id>`, while that host's own messages go under `<id>`. `Moderation.block` stores `Blocks.key(channel, m.sender)` and `Blocks.hides` compares the sender exactly, so the two id spaces never meet: blocking a person from a board post leaves their direct posts visible, and blocking them directly leaves their board posts visible.

Neither is a privacy break - it is the same human under two ids - but "blocked" reading as hidden everywhere makes the gap surprising, and a blocked person can still reach you through any board they use.

EVIDENCE

`MeshEngine.post` uses `"web-${k.me}"` as the wire sender for board posts. `Moderation.block` (line 26) keys on the raw sender; `Blocks.hides` (Protocol.kt:396) is `sender in blocked || key(channel, sender) in blocked` - no prefix handling.

REPRODUCTION

Block someone from a direct Public message, then have them post to the same channel through a phone's board: their board post still shows.

REMEDIATION

Strip the `web-` prefix before saving and before comparing, or store both keys together.