FORENSIC INVESTIGATION · NOSTR CLIENT AUDIT

The Velvet *Filter*

How Primal's central cache moderator decides what you are allowed to read — and how little of it it tells you

> **Verdict.** Primal operates a genuine, disclosed-by-design content-moderation system, and it is more transparent than most Nostr clients. But the disclosure is **partial in a way that matters**: of 89 accounts its own API flagged as “hidden” in a 759-account sample, **only 15 appear on any list Primal publishes**. The remaining 74 are blocked through a path the platform neither publishes nor explains — and Primal's own “why is this hidden?” endpoint returns **silence** for every one of them.

Report date — 7 November 2026
Live API — `wss://cache2.primal.net/v1`
Source reviewed — `PrimalHQ/primal-server` @ `3dcf8b8` (MIT)

Client bundle — `index-c5647853.js`
Deploy — `1787831680315`
Scope: platform-wide, not one account

1 The short answer

> Ask most people what Nostr is and you get the same three claims: no server, no deplatforming, no algorithm. Primal quietly breaks all three. It is not a client that reads from relays in your browser. It is a **central indexer with a SQL database behind it**, run by a company (Primal Systems Inc.), and every answer it gives you — feeds, threads, search, notification badges, DM contact lists, even the list of who zapped a post — passes through a server-side filter *before it leaves the building*. Your client cannot tell what was removed, because what was removed was never sent.
>
> That architecture is not a scandal. It is documented, in MIT-licensed source, and it buys real performance. Primal is unusually honest about having a moderation layer at all: the settings page says so, the API exposes a blocklist, and there is even an endpoint that tells you *why* a given account is filtered. The finding of this report is narrower and, in some ways, less comfortable: **the honest parts and the quiet parts are not the same parts.**

163	74	0	100%
PUBKEYS ON THE FOUR LISTS PRIMAL PUBLISHES	ACCOUNTS BLOCKED IN OUR SAMPLE THAT APPEAR ON NONE OF THEM	EXPLANATIONS PRIMAL OFFERED FOR THOSE 74	OF THE NSFW LIST, INVISIBLE TO THE TRANSPARENCY ENDPOINT

Three things are true at once, and they pull in opposite directions:

1. **Primal is not secretly deplatforming people.** There is no hardcoded hash block list, no campaign-domain tripwire, no covert client-side hiding. Everything is server side, everything is in the public repo, and a blocked user's profile still loads normally — tweets, replies, follower counts, all of it.
2. **Primal is suppressing content at scale anyway.** Spam and adult accounts are kept out of the surfaces that drive discovery — trending, explore, search ranking, DM contact lists — and in the nsfw case this happens to an entire class of user defined only by being bridged from Mastodon.
3. **The suppression is not fully accountable.** The published blocklist is a minority of the actual blocklist. The gap is populated by industrial spam networks, so the *intent* is defensible; the *mechanism* means Primal holds an unpublished switch over who is visible on the largest Nostr client, with no audit trail and no notification to the affected user.

THE LOAD-BEARING SENTENCE

A user blocked through the undisclosed path is **never told**. Their posts appear normal when they look at their own profile — because the filter exempts authors from their own view — while other people may not see them in the places that matter. There is no notice, no appeal, no error, and no public list entry to point at.

2 Why Primal is not like other Nostr clients

The comparison that matters is architectural, and it is the thing that makes every later section possible. A relay-based client like Damus or Amethyst asks many relays for events and filters them locally. Primal asks **one** server — the Primal cache — over a WebSocket using a bespoke RPC protocol called *funcall*. The cache answers from its own Postgres database, which it has been filling from relays for years.

LAYER 1 — THE PUBLIC RELAYS

Nostr as advertised: signed events, gossiped between relays. Primal reads from here. Nothing is removed at this layer. An account "blocked on Primal" is still perfectly reachable on `nos.lol`, `nostr.mom` or `relay.ditto.pub` — we verified this repeatedly (Appendix A).

LAYER 2 — THE PRIMAL CACHE

A licensed, MIT-published OCaml indexer plus a Postgres database, holding every event Primal has ever seen, with its own tables for followers, mute lists, zap receipts and — the important one — `filterlist`. This is where viewing decisions are made. It is a single point of control that a relay-based client simply does not have.

LAYER 3 — THE PER-ANSWER FILTER

`api/lib/moderation.ml`. Every funcall response is parsed, checked against the database, and re-serialised before transmission. Messages that fail are dropped silently. The client receives a well-formed, complete-looking answer that happens to be missing rows. This is the mechanism the whole report turns on.

The design has a consequence that is easy to miss. Because filtering happens *after* the query and *before* the send, it is invisible in a way that client-side filtering is not. When Nostrich hides something in the browser, a curious user can open devtools and see the missing event. When Primal hides something, there is nothing to see — the wire simply never carried it. Auditing therefore requires either the source code or the database's own opinions, which is what we used.

METHOD NOTE

The server has a quirk that defeated the obvious approach: requests are sent as plain JSON but **responses are always zlib-compressed binary frames**, and sending a compressed request closes the connection. Rather than reimplement the codec, we drove the real WebSocket from a headless browser using the platform's `DecompressionStream`, which let us issue arbitrary funcalls and read the raw answers. Every live claim below was produced this way against `cache2.primal.net` on 7–8 November 2026.

3 The published blocklist, in full

Primal ships a funcall called `get_filterlist`. It is not hidden, it is not authenticated, and it returns the platform's blocklist as a synthetic event of kind `10000126`. We called it. It returned twelve list names, of which four are populated and eight are empty placeholders:

Published list	Entries	What it means in practice
`access_pubkey_blocked_nsfw`	**143**	Demoted on trending surfaces only. Content still served.
`access_pubkey_blocked_spam`	**15**	Hidden from content *and* trending, with exemptions.
`access_pubkey_unblocked_spam`	4	Manual overrides. An unblock beats a block.
`access_pubkey_unblocked_nsfw`	1	Single override; a Primal staff account.
`import_pubkey_blocked`, `access_pubkey_blocked`, `analytics_pubkey_blocked`, `access_event_blocked`, `analytics_event_blocked`, `access_pubkey_unblocked`, `analytics_pubkey_unblocked`, `access_event_blocked_spam`	— all empty. The naming implies a much larger moderation pipeline (import, access, analytics tiers) than is currently populated.	

163 distinct pubkeys across the four populated lists. This is the entirety of what Primal publishes about who it filters. The source is candid about the mechanism: the four lists are read from a membership table by `Queries_mem.list_pubkeys` over the internal lists `spam_allow`, `spam_block`, `nsfw_allow` and `nsfw_block` — a curator edits the database, not a document.

3.1 The spam list is impersonation defence, and it is good at it

Fifteen accounts. On first read the names are alarming — *Gigi, Lyn Alden, Optimism, Etherfi*. On inspection every one of them is a counterfeit, and several are running textbook advance-fee scams:

Name on Primal	What gives it away	What the account was doing

Gigi ⚡🧡	Real Gigi is verified at `dergigi.com` as `6e4684…ee93`. This key is `04a504…2c031` and carries no NIP-05.	Copied bio, avatar and website. The real key is **not** blocked; only the impersonator is.
Lyn Alden	No NIP-05; posts `lyn@primal.net` as though it were hers.	Rode the reputation of a macro analyst with a cloned profile.
Optimism	Not the OP Labs key.	"✅ Optimism Airdrop Round 2 Is Live! … Claim your free $OP." — a claim-your-airdrop phishing lure.
Etherfi	Squatted `etherfi@iris.to`; the real project does not use NIP-05 there.	Identity squat with a restaking-brand profile.
TextNow + 11 fullwidth digits	Full-width digits to evade name filters; claims `jack@nostrverified.com`.	"Private chat account for @jack" — impersonating Twitter's founder.
4× ReplyGuy#…	Same display name, four different relay suffixes.	"Master of GM's" — engagement-farming reply spam, one key per relay.
Rachel / Sophia silina	Romance-scam copy, no identity claim.	"I can teach you how to turn your $150 into $7500 with Bitcoin."

This list is a credit to the platform. Impersonation is the one moderation problem Nostr genuinely cannot solve at the protocol level, and Primal solved it the right way: block the counterfeit key only, leave the real person alone, and let the block apply to content as well as to trending.

Two absences are worth recording. **Four exemptions** are published alongside the blocks — Vitor Pamplona (Amethyst's developer), Stacy Herbert, the Nostr News Network and one staff account — and the source confirms why they exist: `filterlist_hidden_pubkeys` contains `and not exists (… not fl.blocked)`, so a single unblock row overrides any number of blocks. That is a sensible escape hatch, and it is also an unlogged one. No published exemption is currently *used* on a key that is also blocked, so we cannot say whether the hatch has ever been exercised in anger.

3.2 The nsfw list is not what it says it is

143 accounts. The name promises adult content. Sampling them shows something else — and this is the first place where the published framing fails to match the data.

60.8%
OF THE NSFW LIST ARE BRIDGED MASTODON ACCOUNTS (87 OF 143)

0.0%
OF ACTIVE TRENDING AUTHORS ARE BRIDGED ACCOUNTS (0 OF 50)

18
LISTED ACCOUNTS WITH NO NIP-05 IDENTITY AT ALL

17
LISTED ACCOUNTS WITH AN EMPTY PROFILE DESCRIPTION

The bridge statistic is the finding. We pulled 50 distinct authors of notes that Primal itself served in its *own* 24-hour trending feed and resolved their profiles: **zero** of them were ActivityPub bridges. We then pulled 314 active authors directly from raw relays as a second baseline: again zero of 267 resolvable profiles carried a `proxy` tag. Against those two baselines, an adult list that is 61% bridged accounts is not describing "accounts that post adult content." It is very nearly describing **"accounts that arrived from Mastodon."**

Concrete cases, with their most recent notes read from the public relays directly:

Account	Listed as	Actual recent output on Nostr
Star Trek Minus Context	nsfw	Star Trek screencaps with alt text. No adult material, no content warning in 25 recent notes.
František Fuka (Fuxoft)	nsfw	Film reviews, Czech gaming history, "oops" posts about Linux security. A well-known Czech developer.
Yahia Lababidi	nsfw	Poetry, a book trailer, an open letter to Israel. 4,713 notes, 5,413 media items, zero adult tags.
Girino Vey	nsfw	Self-hosted photos and commentary; NIP-05 on his own domain.
Lt. Gov. jet tracker	*undisclosed*	A bot that posts the tail number and position of a US state official's aircraft.
Bell Farben bot family	*undisclosed*	Seven coordinated Russian-language roleplay bots (Гермес, Гост, Джарвис, Белфорд, Шедоу, Паладин…).
Netasgard News	*undisclosed*	Ten near-identical regional news-paste accounts, one per country.

The pattern is consistent with keyword-driven classification rather than human review: a sex-education account, a trans artist, a Star Trek fan account and a genre-film reviewer all land in the same bucket because their posts contain vocabulary a classifier associates with adult content. **A list whose name describes the content rule but whose contents describe an interface population is a list that has not been audited.**

> **WHAT WE CAN AND CANNOT SAY**
>
> We cannot prove *why* any individual account is on the nsfw list — Primal does not publish reasons and the classifier is not in the open source. What we can show is that the list, taken as a whole, over-selects bridged Mastodon accounts by roughly two orders of magnitude relative to their share of visible Nostr activity, and that several named members have no adult content at all. The burden of explanation belongs with the list's publisher.

4 What the filter actually does — six layers

The blocklist is only the input. The behaviour comes from `moderation.ml`, which the source describes as a port of the Julia cache's `content_moderation_filtering_2`. Its dispatch table in `funcall.ml` records which answers pass through it. We read all six layers and tested the ones that are testable from outside.

4.1 — THE PER-ANSWER FILTER, APPLIED ALMOST EVERYWHERE

If a funcall runs with `moderate:true` (the default), the finished response is walked message by message. Each message carrying a `pubkey` is checked against the blocklist; each carrying an `id` is checked against the `filterlist` event table and the `deleted_events` table. Failures are dropped — `None` — not flagged, not counted, not reported. The client has no way to distinguish “nothing matched” from “every match was removed.”

The exemptions are the interesting part of the source. Functions that already take the viewer's pubkey and push moderation into SQL run with `~moderate:false`: the main feeds, thread view, `scored`, `events`, `scored_users`. So the *scope* of moderation varies by endpoint, and so does the *exemption set*, which is exactly the kind of asymmetry a reader would never notice.

4.2 — THREE VIEWER MODES, ONE OF WHICH IS NOT THE USER

A viewer with `applyContentModeration: false` gets mode **Off** — nothing is hidden, not even platform blocks. A user who has compiled mute rules in the database gets mode **Compiled**, and their own lists are enforced in SQL. Everyone else — new users, users who never touched settings, and **every anonymous reader** — gets mode **Default**, i.e. the platform's rules.

The default settings Primal serves to a fresh client are explicit about the shape of those rules and were pulled live from `get_default_app_settings`:

```
{"contentModeration": [
  {"name": "primal_spam", "scopes": ["content", "trending"]},
  {"name": "primal_nsfw", "scopes": ["trending"]}
]}
```

So **spam** blocks bite everywhere while **nsfw** blocks bite only in trending. That is a defensible product decision. It is also the reason our feed probes found adult accounts' posts still being served: outside trending surfaces, the nsfw list is inert.

4.3 — THE FOLLOWER EXEMPTION, AND THE ASYMMETRY INSIDE IT

`hidden_pubkeys_default_for` exempts the viewer themself and everyone the viewer follows — but *only from the spam group*. The SQL is unambiguous: the follower subquery sits inside the spam branch, while the nsfw branch is joined by `or` and exempts nobody.

Practically: if you follow an account that Primal has quietly flagged as spam, you keep seeing it. Primal's feed never tells you the account is flagged; your follow graph silently overrides the platform's judgement. For nsfw, following is not enough — but since nsfw only applies to trending, which is not a follows-based surface, the asymmetry rarely becomes visible.

4.4 — THE AUTHOR EXEMPTION: BLOCKED USERS CANNOT SEE THE BLOCK

`author_exempt` is set whenever the request carries `notes: "authored"` — which is what a client sends when rendering your own profile. The SQL then excludes the author's own key from the hidden set.

The consequence is the single most consequential fact in this report: **a spam-blocked user viewing their own profile sees everything.** No gap, no warning, no banner. They have no way to discover the block from inside the product — and we confirmed the effect live: replaying the client's exact profile call for two blocked impersonation accounts returned their own notes normally.

4.5 — SYNTHETIC KINDS CARRY HIDDEN PUBKEYS AND ARE EDITED INDIVIDUALLY

Several responses are not events but invented objects whose *content* is a map of pubkeys. The filter handles each separately, and the behaviour differs:

Kind	Behaviour when a pubkey is hidden	Why it matters
`user_scores`	**entry removed**	Trending ranks are silently recomputed without the account. The account is not merely absent — its score is gone, so no one can see it was ever ranked.
`directmsg_counts`	**contact dropped**	A DM conversation with a flagged account disappears from the contacts list without notice.
`notification`	**whole notice dropped**	You are never told that someone interacted with you. Mentions, zaps and replies simply never arrive.
`zap_event`	**row dropped**	The nastiest one: a zap is hidden if the sender is hidden **for the viewer** *or* by **the receiver's own** mute rules. A third party's list can erase a payment from your public record.

The zap case deserves its own sentence because it is the only place where the filter applies **someone else's preferences to your data**. Everything else is scoped to the viewer. Here, the receiver's compiled rules are evaluated server-side and used to suppress a row in an answer served to a different person.

4.6 — MEDIA IS REMOVED PHYSICALLY, NOT FILTERED

Text moderation is a view. Media moderation is a deletion, and it is worth stating plainly that Primal runs it: `moderation_http.ml` exposes `/purge-media`, which takes a pubkey, a URL, a reason, an initiator, and two flags — `block_all_uploads` and `clear_cache`. It renames local files into `cdn-hidden/`, copies to a backup bucket, **deletes the object from S3**, purges Cloudflare's cache in batches of 30, and stamps a `media_block` UUID onto every referencing row so the hash cannot be re-uploaded.

A comment in the same file records that an empty profile picture once attempted to move the entire media root into `cdn-hidden/`, because `/uploads2/0` names a shard rather than a file. The guard is there now. The episode is a useful reminder that this is a small team with a large, sharp lever.

4.7 Two quiet behaviours worth flagging

The anonymous viewer is Primal. For the `feed` funcall, if a request omits `user_pubkey`, the server substitutes `primal_pubkey` — Primal's own account — and evaluates moderation as though that account were the reader. The consequence is subtle: an anonymous reader does not get the no-viewer branch of the filter, they get *Primal's own follow-graph exemptions*. Staff accounts and anything Primal's account follows behave differently for logged-out readers than the code comments would suggest.

Searches fail loudly where filters fail silently. Several funcalls (`user_search`, `is_hidden_by_content_moderation`) raised `NOTICE "error"` on every parameterisation we tried, while the filter they are meant to explain continued to operate. A platform whose moderation is silent but whose moderation tooling is broken has, in effect, silent moderation with no tooling.

5 The transparency gap

This is the empirical core. Primal publishes 163 keys. Does it block only those? We took 759 pubkeys — the 163 published ones plus 596 accounts active on public relays and in Primal's own trending feed — and asked Primal's own auditing funcall `check_filterlist` about every one. It returns a kind-`10000130` message listing the pubkeys it considers hidden.

Measurement	Count
Pubkeys audited	759
Reported hidden by `check_filterlist`	**89**
… of which appear on a published `get_filterlist` list	15
… of which appear on no published list	**74**
Of those 74, explained by `search_filterlist`	**0**
Of the 143 nsfw-listed keys, reported hidden by `check_filterlist`	**0**

Two findings, and they are separate.

First: 83% of what Primal flags — in this sample — is invisible. Of the 89 accounts its API called hidden, 74 are on none of the four published lists. This is not a rounding error or a stale cache; it is a second, larger corpus of blocks reached through the `filterlist` table directly, and there is no published artefact that names it.

Second: the "why is this hidden?" endpoint answers nothing. `search_filterlist` is the tool Primal's own settings page uses to explain a filter to a user, and it works for the published spam list — asking about the fake Gigi returned a clean `{"action":"block","group":"primal_spam"}`. We then asked it about all 74 undisclosed accounts. **It returned nothing for every single one.** The

code path explains why: it looks the key up in the *compiled moderation rules* and falls back to the membership lists `spam_block` and `nsfw_block`; if the key is in neither, the funcall returns an empty response rather than a third category. A block that exists in `filterlist` but in no list is, to the explanation endpoint, a block that does not exist.

> **THE ASYMMETRY IN ONE LINE**
>
> The account is hidden. The tool that explains hiding says it is not hidden. The user is not told either way.

5.1 Who are the 74?

In fairness to Primal, they are overwhelmingly spam. Of 74, we resolved profiles for 42; the rest have no kind-0 metadata at all, which is itself a spam signal. The identifiable clusters:

- **Bell Farben** — seven Russian-language roleplay bots sharing one naming convention and one bio template (Гермес, Гост, Джарвис, Белфорд, Шедоу, Паладин, and one more). Textbook coordinated inauthentic behaviour.
- **Netasgard News** — ten near-identical regional news accounts, one per country flag, all promising “breaking news, updated continuously.”
- **A Brazil/Spanish-language content farm** — Cypherpunk BTC BR, Pulso Sports BR, Estóico Soberano BR, Carnívoro Protocol, **Protocolo Primal BR**, SportsPulse, wingoaaal. Note the last two: a network willing to squat the platform's own name in Portuguese.
- **Crypto squatters** — `ethfi` (a second Etherfi key, distinct from the published one), BSX Market Bot, moneydesk, 4EVR.ink, Mempool Madness.
- **Genuine oddities** — a jet-tracking bot for a Florida official, an NWC wallet service, a Japanese misskey bot, Studio 3.14, Mary of Nostrmag.

So the undisclosed list is *substantively reasonable*. Primal is blocking spam networks, not dissidents. Nothing in the 74 looks like viewpoint suppression, and we found no political, religious or national pattern; the two Palestine-related accounts we examined sit on the disclosed nsfw list, not the hidden one.

But reasonableness is not the test. The test is whether the platform can be held to account for the switch it holds. It cannot: there is no published list, no reason given, no notification to the blocked user, no appeal, and the auditing endpoint actively returns “not hidden” for exactly the blocks that most need auditing. A well-intentioned undisclosed mechanism and a badly-intentioned one are indistinguishable from outside, and the only reason we can say these 74 are spammers is that we went and looked at their profiles by hand.

5.2 What is *not* happening

Several hypotheses we held going in did not survive testing, and they belong in the record:

Hypothesis	Result	Evidence

Primal hides blocked accounts' profiles entirely	**disproved**	All six blocked profiles loaded with names, bios, follower counts and note lists intact.
Blocked accounts' posts vanish from their own profile page	**disproved**	Replaying the client's exact `mega_feed_directive` call returned the blocked authors' own notes (21 each).
Whole event sets are stripped when a blocked author appears	**disproved**	A spam-blocked account appears as a *repost* on its own profile (via a third-party repost event); only its own notes are withheld.
nsfw-listed accounts' content is withheld	**partly**	38 of 40 probed nsfw accounts had notes served normally; the list is a trending-only demotion, not a content block.
There is a hardcoded hash block list in the client	**disproved**	The web app contains no such list — only a call to `search_filterlist` in `src/lib/search.ts` and the moderation settings UI.
Moderation is applied client-side and can be bypassed	**disproved**	All filtering is server-side; the client receives already-filtered answers.
Trending excludes blocked accounts	**confirmed**	791 events, 166 authors in the 24h trending feed: zero blocked pubkeys, and zero bridged accounts.

The last line of that table is the counterweight to the rest of the section. Trending is the surface where Primal's moderation is strongest and cleanest, and the absence of bridged accounts there is what makes the nsfw list's composition so striking: the platform can and does keep a whole population out of its discovery surfaces. The question is only whether it is the right population.

6 Why the incentives point this way

Nothing in Nostr's protocol required any of this. Primal chose it, and the choices are legible once you see what the company is selling.

Primal's own tagline is *"Live Free."* Its pitch is that Nostr is unusable raw — slow, full of spam, threaded badly — and that a fast indexer fixes that. The commercial logic follows: if your product promise is *a clean feed*, then spam and adult content are not merely unpleasant, they are a defect in the product. Filtering becomes a feature, not a moral position. Every list in this report can be explained by that single commercial incentive, and no conspiracy is needed to explain any of it.

That is precisely why the transparency gap matters more than the motive does. When the incentive to filter is structural and permanent, the only thing standing between "filtering spam" and "filtering whatever is inconvenient" is accountability. Primal has built the first and skipped most of the second.

Accountability mechanism	Present?	Detail
Public blocklist	partly	`get_filterlist` publishes 163 keys. It omits at least 74 blocks that `check_filterlist` itself confirms.
Reason given to the affected user	no	No notification, banner, or status anywhere in the product. The author exemption means a blocked user usually sees nothing wrong at all.
Reason available on request	no	`search_filterlist` works for published lists and returns empty for the rest; the server logs a silent `NOTICE "error"`.
Appeal or review path	no	None in the client, none in the API. The unblock lists exist but are curated internally.
Disclosure in Terms	generic	The ToS forbids “illegal, infringing, fraudulent, harmful … obscene” content and reserves removal rights. It does not describe spam filtering or the bridged-account demotion.
Open-source mechanism	yes	The entire server is MIT and public. The *mechanism* is auditable; the *data* and the classifier are not.
User-side control	yes	`applyContentModeration: false` disables everything, including platform blocks. This is unusually strong and rare.

The last two rows are real and should be counted. A platform that open-sources its moderator and gives users a global off switch has done more than the industry norm. But note the shape of what has been disclosed: **the code is open, the policy is not.** You can read exactly how a key gets filtered; you cannot find out which keys are filtered without interrogating the API, and the API's own accounting is incomplete.

7 Verdict

> **Is Primal censoring people?** Not in the sense the word is usually meant. There is no evidence of viewpoint suppression, no hardcoded blacklist, no covert client-side hiding, and the accounts it targets — impersonators, airdrop phishers, coordinated bot families, adult-content networks — are exactly what it says it targets. Its spam list is one of the better-constructed pieces of moderation we have examined on Nostr, and its identification of impersonation keys while sparing the real accounts is precise work.
>
> **Does Primal have a censorship problem?** Yes, and it is structural rather than malicious. One company operates the index that most Primal users actually read from. That index applies a filter to nearly every answer, silently, before transmission. The filter draws on a blocklist that is **at least 83% unpublished** in our sample (74 of 89), and the

> API's dedicated explain-why endpoint returns nothing for precisely the unpublished majority. Users cannot see the filter, blocked users cannot discover they are blocked, and there is no appeal path. The mechanism is open source; the accountability is not.

The single most important number in this investigation is not 143, or 74, or 89. It is **zero** — the number of explanations Primal's own transparency tool offered for the blocks that most needed explaining. A platform can be trusted with an unpublished blocklist only if it can account for it on demand. Primal has built the blocklist and has not built the accounting.

And a second number deserves equal weight: **60.8%**, the share of the adult-content list that consists of accounts bridged in from Mastodon, against 0% in both of our baselines for ordinary active users. Whatever process produced that list is not measuring what its name says it measures. Until Primal publishes its criteria and its full block set, the most likely reading of that number is that an automated classifier has quietly constructed a penalty on an interface population.

WHAT WOULD CHANGE THE VERDICT

Three things would move this from “accountability gap” to “sound moderation”: (1) make `get_filterlist` the complete set, so published = enforced; (2) make `search_filterlist` return a reason for every hidden key, including the internal ones, and stop returning empty when a block exists; (3) notify a user when they are filtered, and say which list. Any one of these is a few days of work in a codebase that already has all the required pieces.

8 Limits of this evidence

Stated plainly, because the report's own standard demands it.

- **The sample is a sample.** 759 pubkeys is large enough to establish that unpublished blocks exist in quantity, but the ratio **83% is not a platform-wide figure**. Pubkeys drawn from published lists inflate the denominator of matches and depress the apparent gap; pubkeys drawn from active relays may over-represent spam. The defensible claim is the existence and scale of the category, not a precise percentage.
- **We cannot read the blocklist table directly.** Everything about the 74 comes from Primal's own API answering about them. If `check_filterlist` were buggy, some of the 74 might not truly be hidden. Against that: we independently confirmed the mechanism on named accounts, and the profiles of the 74 are consistent with spam.
- **The classifier is closed.** Why any given account landed on the nsfw list is unknown. Our bridge statistic proves over-selection, not the rule that produced it.
- **Author-exemption testing was indirect.** Our replays returned blocked authors' own notes from profile calls and withheld the same authors' notes in feed contexts, which matches the source; but a logged-in session with a known blocked identity would be needed to demonstrate the self-view experience end to end.

- **Several funcalls returned** `NOTICE "error"` (`user_search`, `is_hidden_by_content_moderation`, `advanced_search` with the demo feed). Their behaviour is inferred from source, not observed.
- **One snapshot.** The blocklist changes. All figures are as served by `cache2.primal.net` on 7–8 November 2026 and should be re-run, not quoted, later.

9 Appendix — reproducibility

All probes ran against `wss://cache2.primal.net/v1`. The protocol is Nostr-shaped: `["REQ", subid, {"cache": [funcall, params]}]` as plain JSON; responses are **always zlib binary frames** regardless of negotiation, and compressed requests are rejected. The reference client below decodes them in-browser.

```
// issue arbitrary Primal funcalls; responses are deflate-compressed
async (args) => {
  const inflate = async (buf) => {
    for (const fmt of ['deflate','deflate-raw','gzip']) {
      try { const ds = new DecompressionStream(fmt);
            const w = ds.writable.getWriter(); w.write(new Uint8Array(buf)); w.close();
            return await new Response(ds.readable).text(); } catch(e){}
    }
    return '__undecodable__';
  };
  const ws = new WebSocket(args.url); ws.binaryType = 'arraybuffer';
  await new Promise((res,rej)=>{ ws.onopen=res; ws.onerror=rej; });
  let buf = [];
  ws.onmessage = async ev => buf.push(typeof ev.data === 'string'
                                      ? ev.data : await inflate(ev.data));
  ws.send(JSON.stringify(["REQ","hs",{"cache":["set_primal_protocol",{"compression":"zlib"}]}]));
  await new Promise(r=>setTimeout(r,500));
  const out = {};
  for (const [sub, call] of args.calls) {
    buf = [];
    ws.send(JSON.stringify(["REQ", sub, {"cache": call}]));
    await new Promise(r=>setTimeout(r, args.wait || 5000));
    out[sub] = buf.slice(); ws.send(JSON.stringify(["CLOSE", sub]));
  }
  ws.close(); return out;
}
```

Funcall	What it returned
`get_filterlist`	Kind `10000126`; four populated lists, eight empty; 163 distinct pubkeys.
`check_filterlist {pubkeys}`	Kind `10000130`; the subset Primal considers hidden. 89 of 759 in our sweep; 0 of 143 nsfw entries.
`search_filterlist {pubkey,user_pubkey}`	Kind `10000131`; `{"action":"block","group":"primal_spam"}` for published spam keys; **empty** for all 74 undisclosed.

`get_default_app_settings`	Default rules: `primal_spam` on content+trending; `primal_nsfw` on trending only.
`explore_global_trending_24h`	791 events / 166 authors; 0 blocked, 0 bridged accounts.
`feed {notes:"authored"}`	Blocked authors' own notes **returned** for protected profiles; withheld in feed contexts.

Source examined: `PrimalHQ/primal-server` at commit `3dcf8b8`, MIT licence — `api/lib/moderation.ml` (221 lines), `api/lib/moderation_http.ml` (251), `common/lib/moderation/cmr.ml` (269), `api/lib/queries.ml`, `api/lib/funcall.ml`, `api/lib/handlers.ml`, `api/lib/internal_http.ml`. Client examined: `PrimalHQ/primal-web-app` (TypeScript), bundle `index-c5647853.js` as deployed. Bridge classification via the `proxy` tag written by the Mostr ActivityPub bridge. Baselines: 50 authors of Primal's own trending notes; 314 authors of recent notes pulled directly from `nos.lol`, `nostr.mom` and `relay.ditto.pub`.

A closing note on method. This investigation set out to find censorship and mostly found plumbing. The interesting result is not that Primal hides things — every client hides things — but that it has built the machinery of accountability and then routed the decisions that matter around it. That is a less dramatic finding than a conspiracy and a more useful one: it is fixable, it is specific, and the fixes are listed above.

The accounts on the published lists are overwhelmingly what Primal says they are. The accounts on the unpublished list appear to be spam networks. The problem is that “appears” is doing all the work, and no one outside the company can check.