SECAUDIT 043 - Pixelix OAuth and GRAMOFSMACK client-posting attack surface

SECURITY AUDIT / PUBLIC REMEDIATION RECORD

Field	Value
Date	2026-08-13
Scope	New Pixelix/Pixelfed client API, OAuth authorization-code flow, GRAM client authoring helper, API Access client revocation UI, PWA composer changes
Baseline	Working tree based on commit `577f6440`; reviewed changes were uncommitted
Status	**All six code findings remediated. Static release gate passed; live MariaDB concurrency and real-device interoperability remain required before production rollout.**
Positive controls	GRAMOFSMACK mode fails closed; owner offline-posting switch gates registration, consent, upload and publish; codes/secrets/tokens are random and stored as SHA-256 hashes; consent POST has session CSRF protection; access tokens expire after 30 days; refresh rotation and administrative revocation exist; upload MIME and size are checked; SQL uses prepared statements.
Disclosure	No exploitation known. Before remediation, registration and discovery availability findings were unauthenticated; media ownership, budget, refresh and scope findings required an owner-authorized bearer credential.

1. Executive result

The initial implementation had the right outer boundaries but was not safe to ship. The remediation pass closed the six identified code findings. The strongest property is the mode boundary: every client route checks `site_mode === 'carousel'` before schema, OAuth or content work. Disabling the existing offline-posting switch also stops new registrations, consent, media uploads and posts.

Staged media is now bound to the originating OAuth token record and locked during publication. Registration and authoring budgets use serialized database rows, request-time DDL has been removed, invalid uploads are rejected before budget is charged, refresh credentials expire absolutely after 90 days, and every bearer route enforces its read or write scope.

2. Trust-boundary map

```
unauthenticated network
-> POST /api/v1/apps dynamic client registration
-> GET /api/v1/instance read-only discovery

owner browser session
-> GET/POST /oauth/authorize password/2FA login plus CSRF consent

Pixelix bearer token
-> POST /api/v2/media staged snap_images row
-> PUT /api/v1/media/:id staged ALT update
-> POST /api/v1/statuses snap_posts + snap_post_images + publish
-> GET identity/timeline/status thin read surface
```

The OAuth bearer is a posting credential. It is deliberately weaker than an admin session, but it can create public content and cause federation delivery. It therefore needs both object-level authorization and robust volume controls.

3. Finding A - staged media has no token or client ownership (HIGH, CLOSED)

`POST /api/v1/statuses` validates each supplied ID with only:

```
SELECT id FROM snap_images
WHERE id IN (...) AND post_id IS NULL
```

There is no OAuth app ID, token ID, user ID, upload nonce, source marker, or even `img_status='draft'` condition. The subsequent shared post creator links those rows and the endpoint changes them to `published`.

Impact. Any valid Pixelix token can claim any unattached image row it can identify, including an abandoned upload from a different client and potentially a pre-existing or legacy ungrouped image. It can change the image caption and comment setting, wrap it in a new public post, and make it eligible for the existing federation sweep. Multiple authorized devices are not equivalent to one shared mutable staging area: compromise of one device must not grant control over another device's drafts.

Remediation. Added `snap_oauth_media`, keyed uniquely by image ID and linked to the OAuth token record. Upload records ownership; ALT update and status creation require the same token ID. Status creation begins a transaction before selecting every media row with `FOR UPDATE`, and the existing `post_id IS NULL` condition prevents reuse after commit. Legacy and other-client image rows have no matching ownership record and are rejected.

The original required fix was to add explicit staging ownership, preferably a dedicated table mapping media ID to OAuth token/app, owner user ID, creation time and consumed time. Status creation must select and lock every media row by both ID and token ownership inside the same transaction, require draft state, then atomically mark the staging records consumed. ALT updates need the same ownership predicate. Expire and garbage-collect abandoned staged uploads.

4. Finding B - dynamic client registration is an unauthenticated database write with no throttle (MEDIUM, CLOSED)

While offline posting is enabled, `POST /api/v1/apps` accepts an unauthenticated request and inserts a new `snap_oauth_apps` row every time. It has no IP/client rate limit, duplicate coalescing, registration token, proof-of-work, or bound on registrations per installation. `client_name` is capped, but `redirect_uris` is not length-checked before the database write.

Impact. A remote attacker can grow the OAuth tables until storage or database quotas are exhausted. This is especially relevant on the shared hosting profile SnapSmack supports. The owner switch narrows the exposure window but is expected to remain enabled for normal Pixelix use.

Remediation. Added a per-source-address, transactionally locked registration bucket capped at ten registrations per hour. Registration validates the client name, a single URI-shaped redirect and database field lengths before charging the limiter or writing an application record.

The original required fix was to apply an unauthenticated registration limiter using the trusted client-address helper, enforce strict field lengths before SQL, cap inactive unconsented registrations, and garbage-collect registrations that never receive a token. Do not key this only by attacker-controlled headers.

5. Finding C - runtime schema DDL is reachable from public discovery (MEDIUM, CLOSED)

Every client request calls `px_schema()`. That function executes two `CREATE TABLE IF NOT EXISTS` statements and an `ALTER TABLE ... ADD COLUMN IF NOT EXISTS`. Discovery (`GET /api/v1/instance`) is intentionally public, so an unauthenticated request repeatedly reaches database DDL.

Impact. Even idempotent DDL takes metadata locks and performs catalog work. Repeated public discovery requests can contend with posting and normal page queries. The catch around `ALTER TABLE` hides failure and makes operational diagnosis harder. Schema installation is already owned by the canonical schema and updater; doing it in the request path creates a second authority.

Remediation. Removed `px_schema()` and every `CREATE`/`ALTER` statement from the request adapter. OAuth tokens, media ownership and limiter structures are defined only in the canonical schema and are applied through the existing schema sync/update path.

The original required fix was to remove request-time schema creation entirely. Ship both OAuth tables and all columns through `snapsmack_canonical.sql` and schema sync. If a deployment is incomplete, return a controlled 503 diagnostic rather than trying DDL on a public request.

6. Finding D - shared authoring budget is raceable and invalid uploads consume it (MEDIUM, CLOSED)

`snapsmack_gram_authoring_budget()` reads the window start and count in separate queries, compares in PHP, then performs upserts. There is no transaction, `SELECT ... FOR UPDATE`, atomic conditional update, or database advisory lock. Concurrent uploads can all observe the same old count and overwrite each other, bypassing the stated 300 images/hour ceiling.

The Pixelix upload route also calls the budget before checking whether the multipart `file` field exists or has an upload error. A bearer token can consume the owner's entire budget with 300 empty requests without storing one image.

Impact. Parallel requests can bypass the storage/flood control; conversely, a compromised client can cheaply deny legitimate offline posting for an hour. Because Pixelix and SYBU intentionally share the counter, denial affects both.

Remediation. The shared budget now seeds its rows, locks both in a database transaction with `SELECT ... FOR UPDATE`, checks the cap, updates the counter and commits atomically. Pixelix checks upload presence, error, MIME and size before reserving capacity.

The original required fix was to validate the request before charging it. Reserve capacity with one atomic database operation under a row lock/transaction, then refund only for server-side failures where no durable image was created. Add a concurrency test, not only a sequential threshold test.

7. Finding E - refresh tokens have no absolute expiry (MEDIUM, CLOSED)

Access tokens receive `token_expires_at = NOW() + 30 days`. Refresh validation checks the refresh-token hash and `revoked_at`, but no refresh expiry. Each use rotates the refresh token and creates another 30-day access token indefinitely.

Impact. A refresh token copied from a device remains a standing credential until the owner manually disconnects that client. The advertised 30-day token lifetime is therefore not a credential lifetime. Rotation limits replay after a legitimate refresh but does not bound a stolen token that refreshes first.

Remediation. Added `refresh_expires_at`. Initial code exchange sets a fixed 90-day deadline; rotation preserves that deadline and refresh queries require it to be in the future. Existing tokens without a deadline fail closed and must reconnect. The API Access panel displays the reconnect deadline.

The original required fix was to add `refresh_expires_at`, set an explicit absolute lifetime, enforce it during refresh, and show that expiry in the connected-client UI. Consider a shorter inactivity timeout using `last_used_at`. Keep rotation and administrative revocation.

8. Finding F - OAuth scopes are recorded but never enforced (LOW, CLOSED)

Client-supplied scopes are stored on the app and token, and returned in token responses. `px_bearer()` authenticates the token but does not check required scope. Read, media update, upload and publish endpoints therefore all accept the same token regardless of its recorded scope.

Impact. Today the adapter intentionally has a narrow endpoint set, so this does not expose admin functionality. It does make the least-privilege claim false and creates a latent escalation when endpoints are added later.

Remediation. Registration now accepts only `read` and `write`, rejects unknown scope strings and stores the normalized grant. Identity, timeline and status reads require `read`; upload, ALT update and status creation require `write`.

The original required fix was to normalize registration scopes to a server allowlist, bind the authorized grant to approved scopes, and require `read` or `write` explicitly at each route. Reject unknown scopes rather than preserving arbitrary text.

9. OAuth-specific review notes

- The authorization code is random, single-use in normal sequential operation,

bound to client ID and exact redirect URI, and expires after ten minutes.

- Client secrets, authorization codes, access tokens and refresh tokens are

stored as SHA-256 hashes. Their source entropy is sufficient; password hashing is not required for random 256-bit credentials.

- Consent uses the existing authenticated admin session and global CSRF check.
- The login-return value is limited to a relative `/oauth/authorize` path, which

avoids a general post-login open redirect.

- Pixelix's current native flow does not send PKCE or OAuth `state`. SnapSmack

mirrors that contract. The per-registration client secret still gates code exchange, but native-app secrets are not a substitute for PKCE. This is a compatibility risk to document and revisit with Pixelix upstream rather than silently claiming a fully modern native OAuth profile.

- Disabling offline posting blocks writes even for an otherwise valid bearer.

Read access and refresh remain available until expiry or revocation.

10. Verification performed

- Traced current Pixelix source for app registration, authorization, token

exchange/refresh, credential verification, v2 media upload, media metadata update and JSON status creation.

- Reviewed every new route from Apache rewrite through mode gate, owner gate,

OAuth authentication, database write and response shaping.

- Reviewed password and 2FA return handling, consent CSRF, token hashing,

revocation, upload MIME/size checks, SQL parameterization and shared post transaction boundaries.

- Ran PHP syntax checks on every changed PHP file, Node syntax checking on the

composer JavaScript, the expanded Pixelix security/contract regression check, and `git diff --check`. All passed.

- The regression guard now fails if request-time DDL returns or if ownership,

absolute refresh expiry, scope enforcement, registration throttling, and the locked shared authoring budget are removed.

- Ran the repository-wide canonical schema checker. It recognized the new OAuth

tables but still failed on eleven unrelated tables already referenced by historical/spec/worktree files and absent from the baseline canonical schema. That pre-existing repository-wide failure is not counted as a pass.

- No live MariaDB installation or real Pixelix device was available in this

workspace. No claim of end-to-end interoperability or exploit reproduction is made.

11. Release gate

The source-level release gate is satisfied: Findings A-F are closed in code and the targeted regression gate passes. Before production rollout, run a live MariaDB test proving concurrent-budget serialization and a real Pixelix device test covering login with 2FA, one-photo post, carousel, ALT text, refresh, revocation, mode rejection, disabled-owner rejection and staged-media isolation.