SECAUDIT 039 - GET YOUR SHIT SORTED desktop client: webview trust boundary, credential-at-rest, and API key lifetime

SECURITY AUDIT / PUBLIC REMEDIATION RECORD

Field	Value
Audit ID	2026-08-06-039
Date	2026-08-06
Severity	**HIGH** - the app exposed three unrestricted file-system commands to its own webview while running with no Content Security Policy and `withGlobalTauri` on. Combined with unescaped rendering of server-supplied data, a hostile or intercepted site response could execute script in the webview and write an arbitrary file anywhere the user can write - a local code-execution path. Three MEDIUM items: the API bearer key is stored base64-obfuscated, key expiry was never enforced, and the key travels over plain `http://` if the operator types one.
Component	`tools/gyss/` desktop client - `src-tauri/src/lib.rs` (file commands), `src-tauri/tauri.conf.json` (CSP / global Tauri), `src/scripts/profiles.js` (credential storage), `src/scripts/main.js` (rendering + connection), `src/scripts/api.js` (transport); server side `core/gyss-api.php` (bearer auth, endpoints)
Status	Findings A, B, C **CLOSED in this pass** (code committed; GYSS needs a rebuild to ship). Finding D (credentials at rest) **OPEN - owner decision**, same class as 036/037. Finding E (transport) **MITIGATED** by an explicit warning + override; a hard `https://` requirement remains an owner decision. Finding F (CSP) **OPEN**, needs a live test. Finding G (rate limiting) **ACCEPTED RISK** - a $10 provider billing cap already bounds the loss.
Reporter	Sean (asked which apps still needed audits, then commissioned this one) + Claude (walked the desktop trust boundary and adversarially reviewed its own 0.7.504 additions)
Related	**037 (sibling desktop client SUYB - same credential-at-rest class, closed by a passphrase vault)**, **036 (SmackPress desktop client, cleartext credentials, closed by OS keychain)**, 024 (SYBU recovery - GYSS copied SYBU's base64 key convention), 023 (Unzucker desktop importer). GYSS version at audit: **0.1.1-alpha**; core at **0.7.504**.
Disclosure	No exploitation known. GYSS is a single-operator tool and every finding requires either local read access to %APPDATA%, an active network position (only if the operator uses `http://`), or the operator connecting the app to a hostile server. Not remotely reachable by an unauthenticated attacker.

1. Summary

GET YOUR SHIT SORTED (GYSS) is the operator's desktop sorter: it pulls a filtered batch of photos - or, since 0.7.504, the GRAMOFSMACK post grid - arranges them, and pushes the result back through `core/gyss-api.php` with a scoped `gyss` bearer key. The Rust layer is deliberately thin; all logic lives in a webview.

The server side held up well. Every query is a prepared statement, `limit` and `offset` are bounded, the AI enrichment endpoint is one-image-per-request by design so the server never runs a long batch, and because authentication is a Bearer header rather than a cookie, the permissive `tauri://` / `file://` CORS allowance cannot be ridden by a hostile page. Status and toast messages route through `textContent`, so the obvious error-message injection path was already closed.

The problems are concentrated at **the boundary between the webview and the machine it runs on**, and in **how long the key lives and how it is protected**.

Seven findings; three closed in this pass.

The meaningful one (**Finding A**): `read_file`, `write_file`, and `list_dir` are registered on `invoke_handler`, and app-defined Tauri commands are *not* gated by the capability system - the app's own `capabilities/default.json` says so. They accepted an arbitrary absolute path, and `write_file` created parent directories. Meanwhile the app ships `"csp": null` and `withGlobalTauri: true`, so any script running in the page reaches `window.__TAURI__.core.invoke`. The webview renders JSON fetched from an operator-supplied URL, so a hostile server - or a network attacker, if that URL is `http://` - only needed one unescaped field to turn "sort my photos" into "write a file into the Startup folder".

Finding B supplied that field: five places interpolated server- or file-supplied values into `innerHTML` without escaping.

Finding C: the platform mandates <=4-week API keys and `core/api-auth.php` rejects past-expiry keys, but this handler checked only `is_active` - so an expired GYSS key kept working forever. Worth noting the blast radius grew the same day: 0.7.504 added write endpoints, so that key can now reorder the feed, merge posts, delete the emptied source singles, and trigger federation retractions.

Finding D is the familiar one: the bearer key sits in %APPDATA% as base64, which the file's own comment honestly calls "not encrypted". This is the third sibling with the same habit (036, 037).

2. Finding A - unrestricted file-system commands reachable from the webview (HIGH) - CLOSED

`src-tauri/src/lib.rs` exposed:

```
fn read_file(path: String) // any path
fn write_file(path: String, content: String) // any path, creates parent dirs
fn list_dir(path: String) // any path
```

No validation, no scoping. Three compounding conditions made this reachable:

1. **App commands bypass capabilities.** `capabilities/default.json` grants only

`core:default`, which correctly withholds the fs/dialog/shell *plugin* commands - but commands registered through `generate_handler!` need no capability at all. The config's own description states this.

2. **No CSP.** `tauri.conf.json` sets `"csp": null`, so injected inline script

executes and remote script can be loaded.

3. **Global Tauri bridge.** `withGlobalTauri: true` puts `window.__TAURI__` in

reach of every script in the page, including an injected one.

Chain: hostile or intercepted API response -> unescaped field rendered into `innerHTML` (Finding B) -> script executes -> `invoke('write_file', {...})` -> file written anywhere the user can write (Startup folder, a `.bat`, a config) -> code execution at next login. `read_file` equally allows exfiltration of any user-readable file, with the network already available to send it.

Fix applied. Both commands now resolve through `resolve_in_app_dir()`, which rejects any `..` component and requires the path to sit inside the app data directory, using `Path::starts_with` (component-wise, so a sibling directory sharing a name prefix cannot slip through). Verified with `cargo check` (clean). Residual risk documented in the code: a symlink planted *inside* the app data directory could still point out, which requires local filesystem access - already past the boundary this guard defends.

3. Finding B - server-supplied data rendered into innerHTML unescaped (MEDIUM) - CLOSED

Five sinks in `src/scripts/main.js`:

Sink	Source	Note
`conflictFields()` - `sort_order`	server conflict payload	title/description beside it *were* escaped; this one was missed
`gram-badge` - `image_count` (x2)	server `gram-posts`	added in 0.7.504 - my own code, caught in review
`fmtDate()` catch branch	session / profile JSON	returned the raw input on a malformed date
`fmtFilter()` in the session card	session JSON	
`data-conflict-id` attribute	server conflict payload	

From an honest SnapSmack server these fields are integer-cast and harmless, so the realistic attacker is a hostile server or an `http://` interception - which is exactly the scenario Findings D/E leave open. Defence in depth: all five now pass through `escHtml()`, and `fmtDate()` escapes its fallback at the source since every caller interpolates it into HTML.

4. Finding C - API key expiry never enforced (MEDIUM) - CLOSED

`smack-api-keys.php` stamps every generated key with `expires_at`, and `core/api-auth.php` rejects past-expiry keys ("mandatory <=4-week keys, 0.7.263"). `core/gyss-api.php` authenticated on `key_hash + key_type + is_active` only, so a key the operator believed had expired continued to work indefinitely - defeating the whole point of short-lived keys, and leaving a leaked key (Finding D) valid forever unless manually revoked.

Fix applied. The auth query now mirrors `api-auth.php`: `AND (expires_at IS NULL OR expires_at > NOW())`, with a fallback to the old query if the column is absent, so a site mid-migration keeps working.

4a. Sweep of every `snap_ohsnap_keys` consumer - CLOSED

The same question was put to every bearer-key handler in the codebase. Results:

Handler	Expiry checked	Key type scoped	Action
`core/api-auth.php`	yes	yes	none - the reference implementation
`core/threeacross-api.php`	yes	yes	none
`core/gyss-api.php`	**no**	yes	fixed (Finding C)
`core/flkrfckr-api.php`	**no**	yes	fixed
`core/smackpress-api.php`	**no**	yes	fixed
`core/ohsnap-api.php`	**no**	**no**	fixed - see below

`core/ohsnap-api.php` **was the worst of them**, and a distinct finding rather than more of the same: it authenticated on `key_hash + is_active` alone, with **no** `key_type` **filter at all**. Any key minted for any other tool - a `gyss` sorter key, a `sybu` batch key, a `flkrfckr` key - authenticated successfully against the Oh Snap API. The key-generation UI presents these as separate, per-tool scopes, so an operator handing out or storing a "just a photo sorter" key was in fact handing out Oh Snap access. Now scoped to `key_type = 'ohsnap'` **and** expiry-checked; legacy keys predating the `key_type` column default to `'ohsnap'`, so no existing install is locked out.

All four fixes use the same expiry-aware query with a fallback to the previous query if `expires_at` is missing, so a site mid-migration keeps working.

5. Finding D - bearer key stored base64-obfuscated (MEDIUM) - OPEN, owner decision

`src/scripts/profiles.js` stores the key as `btoa(raw)` in `%APPDATA%\GetYourShitSorted\profiles\<host>.json`. The comment is honest - "not encrypted - just keeps it off plaintext" - and it copies SYBU's convention. Base64 is encoding, not protection: anyone who reads the profile file recovers a working key, and until Finding C it never expired.

This is the same class as **036** (SmackPress, closed by OS keychain) and **037** (SUYB, closed by a scrypt + Fernet passphrase vault). GYSS should follow one of those, not invent a third approach.

Not implemented here because it is a genuine UX decision with a migration: a passphrase vault means unlocking on launch, and existing profiles need converting. Recommended: match 037's vault, or the OS credential store. Sean's call.

6. Finding E - key transmitted over plain http:// (MEDIUM) - MITIGATED

`api.js` builds the endpoint from whatever URL the operator saved, with no scheme requirement. Over `http://` the bearer key is readable by anyone on the path, who can then use it against the site *and* tamper with the JSON the app renders - which is what makes Findings A+B a live chain rather than a theoretical one.

Mitigation applied. `confirmInsecureUrl()` now warns explicitly at both save and connect, naming the consequence, and requires a deliberate override. `localhost` / `127.0.0.1` / `::1` are exempt (never leaves the machine). A hard block is deliberately *not* imposed - same reasoning as 037's FTPS default: it would break legitimate local and shared-host setups. Owner decision.

7. Finding F - no Content Security Policy (MEDIUM) - OPEN, needs a live test

`"csp": null`. A restrictive policy (`script-src 'self'`) would independently kill the Finding A chain by preventing injected script from executing at all.

Deliberately not applied blind. `index.html` carries an inline `<script type="importmap">`, which `script-src 'self'` would block, and the `@tauri-apps/api` shims *require* `withGlobalTauri: true` - so flipping either without testing would brick the app, and I cannot run the built app here. Correct sequence: hash or nonce the importmap (or bundle the shims), set the CSP, launch, confirm all four tabs work, then consider `withGlobalTauri: false`.

8. Finding G - no rate limiting or AI spend control (LOW) - ACCEPTED RISK, no action

No GYSS endpoint has a rate limit, and `gyss/enrich-one` performs a paid AI vision call per request. `core/threeacross-api.php` gates its authoring routes behind an owner-consent flag and an hourly image budget; GYSS has neither. A leaked key can therefore burn AI credit and enumerate the library at full speed.

Decision (Sean, 2026-08-06): accept, do not build a limiter. The operator runs a **$10 provider-side billing cap**, so the entire financial blast radius of a stolen key is $10 - already bounded, and bounded lower than any limit we would have written. SnapSmack adds a second, independent blast-radius cap in `core/ai-provider.php`: `ai_enabled_until` disables AI automatically unless the owner renews, and `snap_ai_cost_accepted()` gates it entirely.

Building a per-key hourly budget would therefore spend real effort, and risk breaking legitimate bulk enrichment (the REPAIR tab is designed to walk a thousand images), to defend a loss that is already capped at $10 by the payment provider. Wrong trade.

Residual risk is availability, not money: if a leaked key burns the cap, the owner's *own* enrichment stops working until the cap resets. That surfaces as enrichment failures rather than a bill, and revoking the key in `smack-api-keys.php` resolves it. Revisit only if AI spend limits are ever raised substantially or a fleet/multi-operator deployment appears.

9. What held up

- **SQL**: every statement parameterised; no interpolation of user input.
- **Pagination**: `limit` clamped to 500 (photos) / 1000 (gram-posts), `offset` floored at 0.
- **CORS**: permissive for `tauri://` and `file://`, but auth is a Bearer header,

not a cookie, so a hostile page gains nothing without the key.

- **Messages**: `setStatus()` and `toast()` use `textContent`; error strings from

the server cannot inject.

- **AI enrichment**: one image per request by design; the desktop owns queueing,

so the server never holds a long batch.

- **New 0.7.504 write endpoints**: mode-gated to `site_mode='carousel'`, and

`gram-carousel` re-validates `post_type`, `trigram_id` and `status` at write time rather than trusting the client's selection.

- **SMACKTALK**: refused outright (409) on both read and write paths.

10. Closure checklist

- [x] A - file commands scoped to the app data dir (`cargo check` clean)
- [x] B - all five innerHTML sinks escaped
- [x] C - key expiry enforced, with a schema fallback
- [x] E - insecure-transport warning with explicit override
- [x] G - **accepted risk, no action** (a $10 provider billing cap already bounds it; see section 8)
- [x] Sweep - expiry enforced across all `snap_ohsnap_keys` consumers; `ohsnap-api.php` additionally scoped to its own key type (section 4a)
- [x] AI spend is now declared before a REPAIR run (image count + "one paid call per image") with a live paid-call counter during it
- [ ] D - credential vault (owner decision: match 037's vault or OS keychain)
- [ ] F - CSP + importmap handling, verified against a running build
- [ ] **GYSS rebuild required** - the Rust and webview fixes only ship in a new build
- [ ] Public `buzzers.php` entry + PDF once the fixes are in a released build