SNAPSMACK_EOF_HEADER Last non-empty line of this file MUST be the canonical EOF marker for this file type: an HTML comment containing five equals, space, the literal string 'SNAPSMACK EOF', space, five equals. Missing or different = truncated/corrupted. Restore before saving.

SECAUDIT 042 - desktop transport coverage: Unzucker, SUYB, SYBU

SECURITY AUDIT / PUBLIC REMEDIATION RECORD

Date: 2026-08-07 Scope: `tools/unzucker/` (0.7.39), `tools/smack-up-your-backup/`, `tools/sybu/` (0.1.32), `tools/_shared/snap_stepup.py` Status: **five findings, all closed in this pass**

The through-line

Every finding here is one bug wearing three hats: **a shared safety helper only protects the tools that call it, and nothing in the tree could tell the difference between "covered" and "never wired up".**

SECAUDIT 040 fixed plaintext-HTTP credential transport in `tools/_shared/snap_stepup.py` and recorded that it closed the same gap across five tools, plus the `hub_discovery.py` item SECAUDIT 037 had deferred. The fix was correct. **Both coverage claims were wrong**, and the audit trail read green for three days.

The fix in this pass is not just the guards. It is that each one now has a test asserting the *import and the call site and the ordering*, so the next false closure fails a suite instead of surviving a review.

Why Unzucker

FLKR FCKR's `config.py` states it was *forked from* `tools/unzucker/config.py`. SECAUDIT 040 found four issues in that fork. The working hypothesis was that some were still live in the original. Two were. The more interesting result is the third, which is about the previous audit rather than the code.

Finding A - the API key crossed plaintext HTTP, and 040 believed otherwise (CLOSED)

Severity: credential disclosure on every request.

`UnzuckerClient.__init__` sets `Authorization: Bearer <key>` as a session header and every call inherits it. There was **no scheme check anywhere** in `main.py`, `poster.py` or `config.py` - a grep for `https` across all three returned nothing. Configure the site URL as `http://` and the scoped `threeacross` key - which can create posts and upload images - went across the network in the clear on ping, upload, post, every request, forever.

The part worth recording. SECAUDIT 040 fixed exactly this class of bug in `tools/_shared/snap_stepup.py` and recorded that the shared fix "closes the same gap in **Unzucker**, GYSS, SUYB, SYBU and Oh Snap at the same time."

For Unzucker that was **not true**. Unzucker never imported `snap_stepup` at all. The fix was real, the helper was correct, and the blast radius was wrong - a shared safety helper only protects the tools that call it, and nothing in the tree could tell the difference between "covered" and "never wired up". The audit trail said protected; the code said nothing.

Fix. Unzucker now imports `confirm_insecure_transport()` from the shared helper and calls it in `_on_connect()` **before** `UnzuckerClient` is constructed - because constructing it puts the key in a session header and the next line sends it. Warn-and-confirm rather than hard refusal, which is the line SECAUDIT 039 drew for GYSS: refuse outright for an account password, warn and confirm for a scoped key. Loopback is exempt; there is no network path to sit on.

If the shared helper cannot be imported (an old build without `_shared/`), the local fallback **fails closed** and refuses any non-`https://` URL.

Finding B - a failed keyring write destroyed the API key (CLOSED)

Severity: data loss, no security impact.

`config.save()` did:

```
if _HAS_KEYRING:
_kr_set(_api_key_account(url), api_key)
ini_api_key = '' # wipe any old base64 value so it doesn't linger
```

`_kr_set()` returns `False` on **any** keyring failure - no backend installed, a locked keychain, D-Bus absent under a headless Linux session - and it swallows the exception to do it. That return value was ignored. So when the keyring refused the secret, the ini was wiped anyway: the keyring did not have the key, the ini no longer had it, and the next launch came up blank with no explanation.

The fallback path was correct; it just was not reachable when it was needed.

Fix. `if _HAS_KEYRING and _kr_set(...)` - the ini is only wiped when the keyring confirms it actually took the secret, otherwise the base64 fallback is written as designed. A behavioural test forces the keyring to refuse and asserts the key still round-trips.

Finding C - `unzucker.ini` was world-readable (CLOSED)

SECAUDIT 040 applied an owner-only permission floor to `flkrfckr.ini` because that file can hold the API key as base64. Unzucker's ini can hold exactly the same thing on any machine without a working keyring, and never got the floor.

Fix. `os.chmod(path, 0o600)` after write, best-effort and swallowed - a no-op on FAT, only partly meaningful on NTFS, and a permissions hiccup must never cost the operator their settings. It is the floor. The keyring is the ceiling.

Reviewed and sound - no findings

- `ig_parser.py` **path containment.** An Instagram export is a third party's

archive and its `media[].uri` values are untrusted. The parser resolves each through `os.path.normpath(os.path.join(export_root, uri))` and requires the result to sit under the export root, rejecting absolute paths and `../..` traversal with an explicit error. This was already right, and is now pinned by a regression so it stays right.

- **Keyring scoping.** Secrets are stored per site URL

(`{url}:api_key` under service `unzucker`), so two sites cannot collide or read each other's key.

- **Bearer-only auth.** No session scraping, no password handling, so the harder

step-up refusal in `snap_stepup.request_authorization()` does not apply here.

Regression - 35 tests across three tools

Suite	Tests	Failures against the pre-fix tree
`tools/unzucker/tests/test_security_regressions.py`	8	5
`tools/smack-up-your-backup/tests/test_transport_regressions.py`	21	8
`tools/sybu/tests/test_transport_regressions.py`	6	6

Every suite was run against the pre-fix tree and confirmed to fail before being accepted. A guard test that has never seen the bug it guards against is a guess.

These deliberately assert **wiring, not just behaviour** - the import, the call site, and the ordering relative to the credential being sent. The helper was always correct; what was missing was anything connecting it to the code that sends the secret, and behaviour-only tests of the helper would have passed throughout the entire vulnerable period.

Specific pins worth keeping:

- `test_the_guard_is_actually_wired_in` (Unzucker) and

`test_hub_discovery_is_wired_and_refuses` (SUYB) - the two false-closure sites.

- `test_backup_engine_checks_before_the_password_is_assigned` - ordering, since a

check after `self._password = password` protects nothing.

- `test_all_three_client_sites_are_gated` (SYBU) - asserts the guard count is at

least the client-construction count, so a fourth connect path added later without a gate fails the suite. That is precisely how this regressed the first time.

- `test_a_missing_helper_fails_closed` (all three) - an old build without

`_shared/` must refuse, never assume.

- `test_the_bearer_key_path_is_not_gated_as_a_password` (SUYB) - pins the

deliberate asymmetry so a later "consistency" cleanup does not flatten the key/password distinction that 039 established.

Finding D - SUYB sent the ACCOUNT PASSWORD over plaintext HTTP, in two files (CLOSED)

Severity: credential disclosure. The heaviest of the five - this is a password, not a scoped key.

Two paths POST `{username, password}` to the login slug with no scheme check of any kind:

File	Path
`backup_engine.py`	`SnapSmackSession.login()`
`hub_discovery.py`	`HubDiscovery` session bootstrap

`hub_discovery.py` is **the exact file SECAUDIT 037 named** when it deferred "refusing (or hard-warning on) plaintext-`http://` admin login", and **the exact item SECAUDIT 040 recorded as resolved**. It had never imported the helper. The item was open the entire time, and the second file was never mentioned by either audit.

A scoped key can be revoked. An operator's account password cannot be, without locking them out of their own site - and SUYB's is an admin password.

Fix. Both paths now **refuse** rather than warn, which is the line SECAUDIT 039 drew: warn-and-confirm for a key, refuse outright for a password. The check runs before the password is assigned to the session object, let alone posted. Loopback exempt. `backup_engine.py`'s Bearer-key early-return still precedes the guard on purpose - a key profile never reaches the password path and is not subject to the harder rule.

This needed a **GUI-free entry point**, since neither file is GUI code and neither should import tkinter to ask whether a URL is safe: `snap_stepup.insecure_transport_reason()` is the existing audited check exposed publicly, returning the reason string rather than prompting.

Finding E - SYBU sent its Bearer key over plaintext HTTP, at three call sites (CLOSED)

Severity: credential disclosure, scoped key.

SYBU *does* import `snap_stepup` - for the step-up password dialog - which is almost certainly why 040 counted it as covered. It never called the transport gate. "Imports the module" and "is protected by the module" are different claims, and only one of them survives a grep.

Three places construct a key-carrying client, and they needed different handling:

Site	Treatment
Connect button	warn-and-confirm on the main thread, before the client is built
Settings *test* button	same, but gated **before the worker thread is spawned**
Auto-connect at startup	refuses quietly with an explanation - **no modal**, because this is not a user action and a dialog raised from a background thread at launch is worse than not connecting

The third is the one worth noting: the correct fix was not "add the same call in three places". A prompt is only appropriate where a human just asked for something.

The other four tools 040 named - checked, not assumed

040 recorded GYSS, SUYB, SYBU and Oh Snap as covered by the same shared fix. Rather than repeat the mistake of trusting that sentence, each was checked for **both** the shared helper and any equivalent local guard:

Tool	Before this audit	Now
Unzucker	no check anywhere	**fixed** - warn-and-confirm (key)
SUYB	no check anywhere, **password** in 2 files	**fixed** - hard refusal (password)
SYBU	imports the module, never calls the gate	**fixed** - 3 sites, treatment per site
GYSS	own `confirmInsecureUrl()` in `src/scripts/main.js`	**already covered** - Tauri app, the Python helper never applied; SECAUDIT 039 gave it its own gate
Oh Snap	not determined	**still unchecked** - ~4 GB directory, scan timed out

GYSS is why this table exists rather than a blanket claim: a grep for the shared helper alone would have marked it vulnerable when it is fine. The question is "does this tool refuse or warn on `http://` **somehow**", not "does it import our module". Getting that wrong in either direction is how audits lose their meaning.

Oh Snap is the one remaining gap and should be SECAUDIT 043. It was not skipped for a good reason - the scan simply timed out - and an unchecked tool is exactly the state that produced this audit.