SECAUDIT 037 - Smack Up Your Backup desktop client: credential-at-rest and transport-authentication attack surface

SECURITY AUDIT / PUBLIC REMEDIATION RECORD

Field	Value
Audit ID	2026-08-04-037
Date	2026-08-04
Severity	**MEDIUM** - the desktop client stores live FTP passwords, admin passwords, scoped API bearer keys, and Google Drive OAuth refresh tokens on a portable disk with no confidentiality protection (base64 or plaintext), so possession of the SUYB folder yields working credentials to the site and its cloud backups. Two lower items: transport peer authentication is disabled by default on both FTP and SFTP, and the restore path trusts attacker-controllable manifest target paths.
Component	`tools/smack-up-your-backup/` desktop client - `profile_manager.py` (profile/credential storage), `cloud_client.py` (OAuth/Box token caches), `ftp_client.py` + `sftp_client.py` (transports), `restore_engine.py` (restore target paths), `hub_discovery.py` (admin-login discovery)
Status	Finding A (credentials at rest) **CLOSED in SUYB 0.7.19** by a passphrase-derived credential vault (scrypt + Fernet); 0.7.18 first added owner-only permissions + honest labelling. Finding C (restore path traversal) **CLOSED in 0.7.18**. Finding B (SFTP host-key pinning) **HARDENED in 0.7.18**; the FTPS cert-verify default remains an owner decision (flipping it breaks shared-host profiles).
Reporter	Sean (asked for a security audit of SUYB) + Claude (walked the desktop client's local trust boundary end to end)
Related	**036 (sibling desktop client SmackPress stored live credentials in cleartext SQLite - same class, closed by OS-keychain storage)**, 034 (removed the redundant `type=keys` credential export from the SUYB *server* endpoint; this audit covers the *desktop client*, which 034 did not), 031 (removed web-host cloud push), 023 (Unzucker desktop importer attack surface). SUYB version at audit: **0.7.17**.
Disclosure	No exploitation known. Every finding requires either local read access to the SUYB folder (Findings A, C) or an active network position between the operator and their own server (Finding B). SUYB is a single-operator tool run by the site owner. The exposure is real but not remotely reachable by an unauthenticated attacker.

1. Summary

Smack Up Your Backup (SUYB) is the operator's desktop backup client. It logs into one or more SnapSmack blogs, pulls a SQL dump plus the media tree over FTP/SFTP, packages a recovery kit, and optionally pushes the package to Google Drive, Box, or Backblaze B2. It is explicitly a **portable** utility: `config.py` and `profile_manager.py` both state that all state "rides next to the executable, never in %APPDATA%, the registry, or anywhere else" - the design target is a thumb drive that moves between machines.

The transfer machinery is sound. Every SQL statement on the server side was covered by earlier audits; on the client side the scheduling path, the AI file matcher, the manifest reader, and the crash-recovery checkpoint all held up (see section 5). The problems are concentrated in two places: **what the client keeps on disk**, and **whether it authenticates the server it connects to**.

Three findings.

The meaningful one (**Finding A**): the portable state includes *live secrets* - the FTP password, the SnapSmack admin password, the scoped `suyb` API bearer key, and the Google Drive / Box OAuth **refresh** tokens - and none of them are protected. FTP and admin passwords are base64-encoded, which the code's own docstring correctly calls "obfuscated (not encrypted)"; the bearer key and the OAuth refresh tokens are stored as literal plaintext JSON. Anyone who reads the

SUYB folder - a lost thumb drive, a synced or cloud-mirrored copy, a shared or recovered machine - obtains working credentials to the live site and to the cloud account holding its backups. This is the same class SECAUDIT 036 closed for the SmackPress desktop client on this same date; SUYB is the sibling tool with the same habit and a wider blast radius (it holds the FTP password *and* the admin password *and* a cloud refresh token, not just an application password).

Two lower findings. **Finding B**: both transports disable peer authentication by default - FTPS runs with `CERT_NONE` / `check_hostname=False`, and SFTP runs with paramiko's `AutoAddPolicy` and no persisted `known_hosts`, so an unknown host key is accepted silently on every connection. The channel is encrypted but not authenticated, which leaves an active network attacker able to intercept the SQL dump (password hashes and secrets) and the media, or to feed a tampered restore. **Finding C**: the restore pipeline uses the manifest's `restores_to` values and `directory_structure` list as remote upload paths without validating them, so a recovery kit from an untrusted source can direct writes outside the intended remote directory (bounded by the FTP/SFTP account's own permissions).

2. Finding A - live credentials stored unprotected on portable media (MEDIUM)

2.1 The code

`profile_manager.py` persists one JSON file per blog under `profiles/`. Passwords are run through base64 and back:

```
def _obfuscate(plain: str) -> str:
return base64.b64encode(plain.encode()).decode()

def _deobfuscate(blob: str) -> str:
...
return base64.b64decode(blob.encode()).decode()
```

`save_profile()` writes `ftp_pass_enc` and `snap_admin_pass_enc` (base64), but the `api_key` (the scoped `suyb` bearer token) and every cloud field are written as-is. The module docstring is candid: *"Passwords are base64-obfuscated (not encrypted) - matches SYBU convention."* Base64 is an encoding, not a secret; it reverses with one function call and gives a false impression of protection to anyone who opens the file and sees an opaque-looking blob.

`cloud_client.py` writes the Google Drive OAuth result to a token cache beside the credentials file:

```
token_file = credentials_file.replace(".json", "_token.json")
...
with open(token_file, "w") as f:
f.write(creds.to_json())
```

That file contains the **refresh token** - a long-lived credential that mints new Drive access tokens without re-consent. Box tokens (`_box_token.json`) are cached the same way. These caches live wherever the operator put their credentials JSON, which for the portable design is inside the SUYB folder.

2.2 Impact

Reading the SUYB folder yields, per configured blog:

- the **FTP/SFTP password** (base64 - trivially decoded), giving filesystem-level

write access to the site;

- the **SnapSmack admin password** (base64), giving full CMS administration;
- the **scoped** `suyb` **API bearer key** (plaintext), valid until it expires;
- the **Google Drive / Box OAuth refresh token** (plaintext), giving standing

access to the cloud account that stores every backup - i.e. the site's entire content and database history, including the `snap_users` dump with password and recovery-code hashes.

The realistic exposure paths for a *portable* tool are exactly the ones that make portability attractive: a thumb drive that is lost or left in a machine; the folder synced into a cloud drive or a general file backup; a shared, resold, or forensically recovered computer. No network access and no privilege escalation is required - just a read of the folder.

2.3 The tension (why this is not a copy-paste of the 036 fix)

SECAUDIT 036 closed the identical class for SmackPress by moving secrets into the OS keychain (Windows Credential Manager / macOS Keychain / Linux Secret Service) with a plaintext-DB fallback. That fix is **architecturally incompatible with SUYB's stated design**: the OS keychain is machine-bound, and SUYB deliberately keeps all state next to the executable so a thumb drive works on any machine. Keychain storage would break the portability that is the tool's premise.

So this finding is a genuine product decision, not a mechanical port. The defensible options are:

1. **Passphrase-derived encryption at rest.** Encrypt the profile store and token

caches with a key derived (scrypt/PBKDF2) from an operator passphrase entered at unlock. Keeps portability; the thumb drive alone is no longer sufficient. This is the recommended direction.

2. **Keychain when present, encrypted-file fallback when portable.** Mirror 036

but make the portable fallback *encrypted* (option 1), not base64.

3. **Document honestly and stop implying protection.** At minimum, rename the

base64 fields so they do not read as encryption, and state plainly in the README and UI that the SUYB folder *is* a secret equivalent to the passwords in it and must be handled like a password file. This is the floor, acceptable only as an interim.

What is **not** defensible is leaving base64 in place while calling the field `_enc`, because it tells the operator their credentials are encrypted when they are not.

3. Finding B - transport peer authentication disabled by default (LOW / defense-in-depth)

3.1 FTP

`ftp_client.py` defaults `verify_cert=False`, and in that mode builds the TLS context as:

```
ctx = ssl.create_default_context()
ctx.check_hostname = False
ctx.verify_mode = ssl.CERT_NONE
```

The connection is encrypted but the server certificate is neither validated nor matched to the hostname. The in-code rationale ("shared hosting certs often don't match the domain ... same as clicking Trust this certificate in FileZilla") is a real operational fact, but the effect is that an active man-in-the-middle can present any certificate and the client proceeds.

3.2 SFTP

`sftp_client.py` defaults `auto_add_host_key=True` and, with no `known_hosts` file supplied (the default), installs `paramiko.AutoAddPolicy()`. Crucially the accepted key is **never persisted** (`save_host_keys` is never called) and a fresh `SSHClient` is built on every `connect()`/`reconnect()`. So this is weaker than trust-on-first-use: there is no first-use *pin* to compare against later - an unknown host key is accepted silently on **every** connection, and a substituted key on a later connection is indistinguishable from the real one.

3.3 Impact

SUYB moves the most sensitive artifact the site has - a full SQL dump including `snap_users` (password and recovery-code hashes, 2FA state) plus the entire media tree. An attacker positioned on the network path (hostile Wi-Fi, compromised router/ISP segment) can, by default configuration:

- terminate the TLS/SSH session at their own endpoint and read the dump and media

in clear as they pass through;

- tamper with a **restore** stream so the operator uploads attacker-chosen bytes

to their own live server.

Severity is held to LOW because it requires an active on-path attacker, the operator controls both endpoints, and encryption (against passive capture) is still on. But "encrypted to an unverified peer" is exactly the gap MITM exploits.

3.4 Recommendation

Make verification the **default**, with an explicit, per-profile, clearly-labelled opt-out for the shared-hosting reality:

- FTP: default `verify_cert=True`; when the operator opts out, surface it as a

visible per-profile "accept invalid certificate (not recommended)" choice rather than a silent default.

- SFTP: implement real TOFU - on first connect, record the host key to a persisted

`known_hosts` in the SUYB folder; on later connects, **reject** a changed key and tell the operator. `RejectPolicy` after the first pin, not `AutoAddPolicy` every time.

A related sub-item: `hub_discovery.py` POSTs the admin username and password to `{site_url}/{login_slug}`. If a profile's `site_url` is `http://`, those credentials cross the wire in clear. Consider refusing plaintext-HTTP admin login (or warning hard) rather than relying on the operator to type `https://`.

4. Finding C - restore trusts manifest-supplied remote paths (LOW)

`restore_engine.py` drives uploads straight from the manifest:

```
ftp.ensure_directory_tree(manifest.directory_structure) # attacker-controlled list
...
ftp.upload_file(match.local_path, record.restores_to, ...) # attacker-controlled target
```

and the transports build the remote path by concatenation:

```
remote_full = f"{self.remote_dir}/{remote_rel_path}".replace("//", "/")
```

`restores_to` and `directory_structure` come from the recovery kit's manifest. For a kit the operator produced themselves this is fine. But restore also accepts a **local ZIP** and a **cloud-downloaded ZIP**; if an operator ever restores a kit they did not create, a crafted manifest with `restores_to` values like `../../../../home/other/public_html/shell.php` (or absolute / drive-letter paths) would direct writes outside the intended `remote_dir`. The blast radius is bounded by the FTP/SFTP account's own permissions and any server-side chroot, which is why this is LOW - but the client should not extend the reach of a hostile manifest.

Recommendation: validate every `restores_to` and `directory_structure` entry before use - reject absolute paths, drive letters, NUL bytes, and any `..` segment; confirm the normalized join stays under `remote_dir`. This is the same default-deny path discipline SECAUDIT 034 applied to skin ZIP extraction, and it is an unconditional fix with no design tradeoff.

5. What was already correct (and stays)

Recorded so the findings do not obscure the parts that held up:

Property	Where	Verdict
OS-schedule registration validates `HH:MM` and uses list-form `subprocess` (no shell, no injection)	`os_schedule.py`	Correct
Scheduled invocation string built from `sys.executable`/`__file__`, not user input	`os_schedule.py`	Correct

Property	Where	Verdict
AI file matcher is fully local (sentence-transformers on device); no path or filename leaves the machine	`ai_matcher.py`	Correct - no exfiltration
Manifest tar/zip read into memory for `manifest.json` only; recovery kit is never `extractall`-ed to disk	`manifest_reader.py`	Correct
Outer backup ZIP extracted with stdlib `zipfile.extractall` (modern Python strips `../`absolute)	`restore_engine.py`	Correct (stdlib-sanitized)
Crash checkpoint is JSON via temp-file + atomic rename; no `pickle`, no code execution on load	`checkpoint.py`	Correct
Hub discovery uses `requests` with default TLS verification (certs checked)	`hub_discovery.py`	Correct
Profile and checkpoint filenames sanitize `/` and `\` out of blog names	`profile_manager.py`, `checkpoint.py`	Correct
Cloud (Google API client, Box via `requests`, B2) all verify TLS by default; the only `CERT_NONE` in the tool is the FTP one in Finding B	tool-wide	Correct - scoped
Backup prefers the least-privilege scoped `suyb` bearer key over admin login	`backup_engine.py`, `hub_discovery.py`	Correct

6. Remediation applied in 0.7.18

- **Finding C (CLOSED in both directions).** `restore_engine.py` now runs every manifest

`restores_to` and `directory_structure` entry through `_is_safe_rel_path()` before use, rejecting absolute paths, drive-letter/UNC paths, NUL bytes, and any `..` segment. Unsafe directory entries are dropped (with a warning), unsafe file targets fail explicitly and are surfaced in the restore log, and the post-upload verify loop skips them. A hostile recovery kit can no longer steer writes outside the profile's `remote_dir`. The same shared containment rule now protects `backup_engine.py`: a hostile server inventory cannot use an absolute, drive/UNC, or traversing `restores_to` value to overwrite a local file outside staging.

- **Finding B (HARDENED).** `sftp_client.py` now designates a portable, SUYB-owned

`suyb_known_hosts` file as the writable host-key store, so paramiko's AutoAddPolicy persists the server key on first connect and paramiko rejects a changed key on every connection thereafter (`BadHostKeyException`). This converts the previous blind "accept any key, every time" into real trust-on-first-use with no impact on the first honest connection. `suyb_known_hosts` is gitignored (per-machine runtime state). The FTPS `verify_cert=False` default is **unchanged** - flipping it would break operators on shared hosts with mismatched certs; see below.

- **Finding A (0.7.18: HARDENED; 0.7.19: CLOSED).** 0.7.18 tightened profile and

token files to owner-only (`chmod 0600`) and made the storage honest (no app-baked "encryption" theatre). **0.7.19 closes it** with a real credential vault (`secret_vault.py`): a master key is derived from an operator passphrase with **scrypt** (N=32768, r=8, p=1) and secrets are sealed with **Fernet** (AES-128-CBC + HMAC-SHA256). The FTP password, admin password, scoped API key (`profile_manager.py`), cloud-sync Backblaze keys (`sync_manager.py`), and Google Drive / Box OAuth refresh tokens (`cloud_client.py`) are all sealed. The passphrase is never written; only `vault.meta` (salt, KDF params, an encrypted verifier) rides on the drive, so a lost or synced drive no longer yields credentials.

- **Portability preserved.** No machine-bound keychain is needed for interactive

use - the operator enters the passphrase at startup (`App._gate_encryption_unlock`).

- **Unattended path.** A scheduled/headless run has no one to type the passphrase,

so - only on explicit opt-in - the master key is cached in *this machine's* OS keychain (`keyring`), off the portable drive. `headless.run_backup_all` unlocks from there; with no keychain backend, scheduled backups are skipped with a clear log line rather than running credential-less.

- **Backward compatible.** Legacy base64/plaintext profiles load unchanged;

enabling re-seals every profile, disabling restores legacy encoding; token caches self-describe (`suyb_sealed`) so a mixed store always decodes.

- **Crash-safe lifecycle.** Profiles, sync jobs, every referenced cloud-token

variant, vault metadata, and machine-key state are committed as one recoverable migration. A durable owner-only rollback journal is processed before GUI or headless startup loads credentials. Encryption failures and locked-vault writes fail closed rather than falling back to legacy/plaintext storage.

7. Recommended follow-ups (owner decision)

- **Finding B - FTPS verification default + HTTP admin login.** Consider defaulting

FTPS to verify with a visible per-profile "accept invalid certificate (not recommended)" opt-out, and refusing (or hard-warning on) plaintext-`http://` admin login in `hub_discovery.py`. Deferred here because the current default exists for a real shared-hosting reason and flipping it can break live profiles.

7.1 FTPS transport decision (2026-08-05)

The FTPS certificate-verification default is **left OFF**, deliberately. SnapSmack's audience runs on budget shared hosts that ship self-signed or expired certificates; defaulting FTPS to full CA validation would break the majority of real profiles on first connect, and a binary "verify / don't verify" toggle only trades one bad outcome for another. This is a documented, accepted tradeoff - not an oversight.

The forward path is **not** mandatory CA validation. It is **TOFU certificate- fingerprint pinning**, mirroring the SFTP host-key pinning already shipped in 0.7.18: on first connect, record the server's presented certificate fingerprint next to the executable; on every later connect, proceed silently if it matches and hard-warn if it changes. That gives real man-in-the-middle detection **without ever requiring a CA-signed certificate**, which is exactly the protection the cheap-host reality needs. Scheduled as a future SUYB enhancement; no code change in this release.

8. Verification

- `_is_safe_rel_path()` was exercised against ten traversal/absolute/drive-letter/

NUL/empty payloads (all rejected) and five legitimate relative paths including a leading `./` (all allowed).

- The credential vault was tested end to end: seal/open roundtrip (incl. unicode

and colon-bearing secrets), wrong-passphrase rejection, ciphertext-tamper rejection, lock-hides-secrets, change-passphrase + re-seal, enable/disable migration with **no plaintext left on disk**, and the full unattended chain (enable-with-machine-key -> lock -> headless unlock via keychain -> decrypt -> revoke -> unlock refused). scrypt KDF measured ~90 ms (interactive-appropriate).

- A persistent adversarial regression suite exercises 13 cases: complete

profile/sync-job/Google/Box lifecycle migration, global- and sync-only token references, injected write failures with rollback, machine-key preservation, locked-vault downgrade refusal, reverse local traversal, and unwritable SFTP pin storage. All 13 pass.

- All changed modules compile clean (`py_compile`) and import; the GUI constructs

with the Credential Encryption card rendered; the SFTP pin path resolves next to the executable.

- `chmod 0600` is best-effort and wrapped so it never breaks writes on FAT/Windows.

The server-side SUYB export surface was already addressed in SECAUDIT 034; this report is the first to cover the **desktop client's** local attack surface, and its findings are local-storage and transport-authentication defaults, not server vulnerabilities.