SECAUDIT 036 - SmackPress WordPress migration attack surface (import sanitisation, upload re-encoding, credential storage)

SECURITY AUDIT / PUBLIC REMEDIATION RECORD

Field	Value
Audit ID	2026-08-04-036
Date	2026-08-04
Severity	**MEDIUM** - the migration API is authenticated and its database access is fully parameterised, but imported WordPress HTML was stored verbatim, so any script a WP plugin (or a compromised source blog) left in a post body would have run on the destination site for every visitor. Stored-XSS class; confidentiality/integrity impact scoped to a hostile source blog. Two low-severity hardening items alongside.
Component	`core/smackpress-api.php` (media/upload, posts, pages, mosaics, categories), `core/image-ingest.php` (shared upload pipeline), `tools/smackpress-wp-companion/` (WordPress REST bridge), `tools/smackpress/` desktop client (`config.py`), `smack-api-keys.php` (key issuance), `api.php` (router)
Status	**CLOSED in 0.7.496** (companion plugin 1.1.1) - import sanitiser, GIF re-encoding, OS-keychain credential storage
Reporter	Sean (migrating the real Bad Day With A Camera WordPress blog into SnapSmack; asked for an audit of SmackPress and specifically raised imported malicious JS/CSS - "lord only knows what some WP plugins insert") + Claude (walked the SmackPress trust boundary end to end)
Related	**024A (imported-caption XSS - same class: content brought in by a migration must be sanitised before display)**, 023 (Unzucker desktop importer attack surface), 025 (skin inline-script injection), 034 (credential-at-rest discipline)
Disclosure	No exploitation known. SmackPress is a one-shot migration tool run by the site owner with an expiring key; the exposure requires importing a source blog whose post bodies carry hostile markup, which is plausible for any WordPress install carrying third-party plugin output. Fixed before the first real migration ran.

1. Summary

SmackPress migrates a WordPress blog into a SMACKTALK install. The desktop client reads posts from a WordPress companion plugin, optionally rewrites them, and pushes them to an authenticated JSON API (`api.php?route=smackpress/*`) that creates longform posts, static pages, mosaics, and gallery images on the destination site.

The core of the API is sound. Authentication is enforced before any route runs; keys are 256-bit CSPRNG tokens stored as SHA-256 hashes, shown once, with a mandatory expiry capped at four weeks; every SQL statement across every route is parameterised; and the image-upload path regenerates a safe filename and re-encodes JPEG/PNG/WebP through GD, so an uploaded name cannot inject a path or a `.php`, and an image cannot carry an executable payload.

Three weaknesses were found, one meaningful and two hardening.

The meaningful one: **imported post and page HTML was stored verbatim.** A WordPress post body is full HTML and routinely carries whatever plugins inject - tracking pixels, embeds, inline styles, and scripts. SmackPress passed that through `smack_autop_long()`, which returns content unmodified when it already begins with a block tag, and wrote it straight to the database. Because that content renders on the public site, a `<script>` in a source post body would execute in every visitor's browser on the destination. This is a stored-XSS vector gated only by "the source blog contained hostile markup" - which for a migration of arbitrary WordPress content is not a strong gate.

The two hardening items: GIF uploads skipped the GD re-encode that neutralises the other formats, and the desktop client stored the WordPress application password, the SnapSmack API key, and the AI key in cleartext in a local SQLite file.

A separate correctness defect in the companion plugin, fixed in the same pass, is recorded in section 6 because it directly affected whether images imported at all.

2. Finding A - imported HTML stored without sanitisation (MEDIUM)

2.1 The code

`core/smackpress-api.php`, the `posts` and `pages` routes both did:

```
$content_html = smack_autop_long($raw_content);
// ... INSERT/UPDATE snap_posts.content = $content_html
```

`smack_autop_long()` wraps bare paragraphs and escapes them, **but returns the input untouched the moment it already starts with a block tag**:

```
if (preg_match('/^\s*<p/i', $text)) return $text; // raw HTML passes straight through
```

A migrated WordPress body virtually always begins with <p> (or another block element), so the escaping branch never fired and the raw HTML - scripts, iframes, event handlers, inline CSS and all - was stored as-is and later rendered on the public post/page.

2.2 Impact

Stored cross-site scripting on the destination's public pages. The payload source is the imported blog: any `<script>`, `<img onerror=...>`, `javascript:` link, `<iframe>`, or `<style>` present in a WordPress post body - whether authored, injected by a plugin, or left by a prior compromise of the source site - executes in the browser of every visitor to the migrated post. The API's authentication does not mitigate this: the authenticated party is importing content they did not necessarily write.

This is the same class as SECAUDIT 024A (imported captions), which established the rule that content crossing an import boundary must be sanitised before it can be displayed. Longform post and page bodies were not yet on that rule.

3. Finding B - GIF uploads not re-encoded (LOW)

`core/image-ingest.php` re-encodes JPEG, PNG, and WebP through GD, which discards any bytes appended after the image data (a polyglot / trailing-payload defence). GIF was not handled by that pipeline: `imagecreatefromgif` was never called, so a `.gif` upload was stored byte-for-byte as received.

The filename is regenerated with a validated extension and the file is served as `image/gif`, so this is not a direct execution vector; the residual risk is a GIF/HTML polyglot being content-sniffed if the uploads directory is served without `X-Content-Type-Options: nosniff`. Low, but inconsistent with the treatment of every other accepted format.

4. Finding C - desktop client stored live credentials in plaintext (LOW)

The SmackPress desktop tool persisted `wp_app_password`, `snap_api_key`, and `ai_api_key` as cleartext rows in `smackpress.db`, a SQLite file next to the application (`tools/smackpress/smackpress/config.py`). Anyone able to read that file - a synced backup, a shared machine, a cloud-drive folder - obtained the WordPress application password and the SnapSmack import key directly.

Blast radius is limited by design: the SnapSmack key expires (four weeks maximum) and the WordPress application password is independently revocable. But storing live secrets in cleartext when the operating system offers a keychain is avoidable.

5. What was already correct (and stays)

Recorded so the closure does not obscure the parts that held up:

Property	Where	Verdict
Auth enforced before every route	`smackpress-api.php` (bearer check precedes routing)	Correct
256-bit CSPRNG keys, SHA-256 at rest, shown once, mandatory <=4w expiry	`smack-api-keys.php`	Correct
All SQL parameterised (posts, pages, mosaics, categories, maps)	`smackpress-api.php`	Correct - no injection found
Upload filename regenerated; `is_uploaded_file`; extension allowlist; JPEG/PNG/WebP re-encoded	`image-ingest.php`	Correct
Client transport TLS-verified (default `urllib` context, no cert bypass)	`smacktalk_client.py`	Correct
Whitelisted enums (status, image size/align), integer-cast IDs	`smackpress-api.php`	Correct

6. Companion-plugin correctness defect (fixed alongside)

Not a security finding, but it governed whether the migration worked and was fixed in the same pass. The Bad Day With A Camera blog had moved from `baddaywithacamera.ca` to `old.baddaywithacamera.ca`, but post bodies carried hardcoded absolute image URLs on the old host. The companion plugin resolved inline `<img>` images by matching against the *current* upload base URL, so old-host URLs matched nothing, never resolved to attachments, never migrated, and rendered broken on the still-online old site. Companion plugin 1.1.1 adds a host-agnostic normaliser that rewrites any host in front of the uploads path to the current one, applied both before resolution (so images migrate) and as a `the_content` display filter (so the live old blog stops showing broken images).

7. Remediation

1. **Sanitise imported HTML before storage.** A DOM-based allowlist sanitiser,

`smackpress_sanitize_html()`, now runs on both `posts` and `pages` content. It removes `script`, `style`, `iframe`, `object`, `embed`, `form`, `svg`, `math`, `link`, `meta`, and `base` with their subtrees; strips `on*` handlers, `style`, `class`, and `id`; strips `javascript:`, `vbscript:`, and `data:` URLs from `href`/`src`; unwraps unknown tags while keeping their safe text; forces `rel="noopener noreferrer"` on `target="_blank"` links; and degrades to text-only on a parse failure so raw HTML is never stored. Regex HTML sanitising was deliberately not used - it is bypassable; the sanitiser parses the document with `DOMDocument`.

2. **Re-encode GIF uploads.** `image-ingest.php` now re-encodes `image/gif`

through GD in place, matching the other formats. GD flattens to the first frame - an accepted trade for a photoblog.

3. **Store client credentials in the OS keychain.** `config.py` now writes the

three secret keys to the OS keychain (Windows Credential Manager / macOS Keychain / Linux Secret Service) via `keyring` when a backend is present, clearing any prior plaintext DB copy, and falls back to the DB only when no backend exists so the tool never breaks. The README documents that if the fallback is in use, `smackpress.db` must be treated as a secret.

8. Verification

The sanitiser was exercised against twelve attack payloads and four preservation cases before shipping. Every payload was neutralised - `<script>`, `<style>` with a CSS `javascript:` url, `<iframe>`, `<form>`, `<svg><script>`, inline `onclick`/`onerror`/`onmouseover`, `style`/`class`/`id` attributes, a `javascript:` href, and a `data:text/html` href - and all four preservation cases passed: paragraphs with inline emphasis, safe links (which additionally gained `rel="noopener noreferrer"`), safe images, and SnapSmack `[mosaic:]`/`[img:]` shortcodes, which survive untouched as text.

9. Disposition

CLOSED in 0.7.496, companion plugin **1.1.1**. Finding A is closed by the DOM allowlist sanitiser applied on import; Finding B by the GIF re-encode; Finding C by OS-keychain storage with a safe fallback. The parameterised queries, authenticated routing, expiring-key model, and safe upload filename handling that the API already had remain in place. The import sanitiser applies going forward, so every post and page migrated from this point is cleaned before it can render.