SNAPSMACK — SECURITY AUDIT

Audit #019 · 2026-05-26 · v0.7.184 "Barcalounger"

Audit number	019
Date	2026-05-26
Release	0.7.184 "Barcalounger"
Auditors	Claude (Anthropic), Sean Moir
Scope	0.7.184 delta — SMACKBACK Skin JS Scanner, archive landing page, update track / branching infrastructure, multisite TRACK badge, and all supporting plumbing
Audit type	Deep manual code review — full read of every changed function
Prior audit	018 — 2026-05-23 — Hub maintenance mode

Background

0.7.184 introduced four discrete features: a skin JS security scanner (SMACKBACK), an archive landing page mode, an update track / branching system (BORING / BITCHIN'), and a multisite TRACK badge on the spoke fleet roster. An initial shallow pass produced a clean report. Sean pushed back — correctly — and a second full deep-read audit was performed, reading every changed function in full. This revised document replaces the initial 019 report and reflects the actual findings.

Findings Overview

#	Title	Severity	Status
1	Skin JS scanner — symlink escape	Low	Closed
2	Archive redirect — HTTP_HOST fallback	Low	Closed
3	Settings save — enum fields unvalidated	Low	Closed
4	Skin JS scanner — detail field content	Informational	Closed
5	Multisite — spoke-controlled TRACK badge	Informational	Closed

Finding 1 — Skin JS Scanner — Symlink Escape [Low]

Location: core/smackback.php — smackback_scan_skin_js()

smackback_scan_skin_js() uses RecursiveDirectoryIterator with the SKIP_DOTS flag only. PHP's RecursiveDirectoryIterator, without FOLLOW_SYMLINKS, will not recurse into symlinked *directories*, but it will yield symlinked *files* as normal file entries. A symlink inside any non-base skin directory — pointing to a file outside skins/ with a .php, .html, .htm, or .js extension — would be opened and scanned by @file($abs, FILE_IGNORE_NEW_LINES). This means a prior-compromise artefact (a malicious symlink left in a skin) could allow the scanner to read sensitive files from outside the skins tree. The scanner does not write or execute the file contents, so the worst-case impact is information disclosure (e.g. reading database credentials from a config file if that file has one of the scanned extensions).

Fix applied (0.7.184):

realpath($skins_dir) is resolved once before the outer loop. For each candidate file, realpath($abs) is called and checked: if it returns false (broken symlink) or if the resolved path does not start with $skins_real, the file is skipped. This prevents any symlink that resolves outside the skins tree from being read.

Verdict: Closed. Fixed in 0.7.184. realpath() guard applied before @file() read.

Finding 2 — Archive Redirect — HTTP_HOST Fallback [Low]

Location: index.php — archive landing page redirect, ~line 107

When homepage_mode is 'archive', index.php performs a 302 redirect to the archive page. The original code built the redirect URL as:

```
$archive_url = rtrim($settings['site_url'] ?? rtrim('https://' . $_SERVER['HTTP_HOST'],
'/'), '/') . '/archive';
```

If site_url is null (not set in snap_settings — possible on a fresh install before settings are saved), the PHP null-coalescing operator falls through to the $_SERVER['HTTP_HOST'] fallback. On a correctly configured web server the Host header is controlled by the vhost configuration and is trustworthy. On a misconfigured server that does not set a ServerName / server_name directive, the Host header value comes directly from the HTTP request. An attacker who sends a request with Host: evil.example.com would receive a 302 to https://evil.example.com/archive — a reflected open redirect. No user session data or CSRF tokens are involved, but open redirects can be used in phishing chains.

Fix applied (0.7.184):

The fallback now uses a path-relative redirect (/archive) when site_url is absent or does not match the https?:// pattern. A path-relative Location header cannot redirect to an off-site host regardless of what the client sends in the Host header.

Verdict: Closed. Fixed in 0.7.184. Relative-path fallback eliminates the Host-header dependency.

Finding 3 — Settings Save — Enum Fields Unvalidated [Low]

Location: smack-settings.php — save_settings POST handler, ~line 128

The save_settings handler persists all values in $_POST['settings'] to snap_settings via a generic foreach loop with no field-level validation. Three new enum fields added in 0.7.184 — update_track, archive_layout, and archive_thumb_style — were saved without whitelisting their valid values.

For update_track, updater_get_track() reads the DB value but whitelists it at read time (only 'dev' passes; anything else returns 'stable'). The functional impact is therefore zero: saving 'update_track = hacked' results in stable-track behavior. However, storing arbitrary strings in an enum column is sloppy, creates unexpected state in the database, and leaves a foothold for future bugs if the whitelist is accidentally widened or removed. archive_layout and archive_thumb_style have no secondary whitelist and are rendered in PHP conditionals — an unexpected value falls through to the default case, so there is no injection risk, but the stored state is incorrect.

Fix applied (0.7.184):

Whitelist guards added immediately before the foreach loop: update_track is collapsed to 'stable' if not in ['stable', 'dev']; archive_layout to 'grid' if not in ['grid', 'list']; archive_thumb_style to 'square' if not in ['square', 'natural']. All comparisons use in_array() with strict mode (third argument true).

Verdict: Closed. Fixed in 0.7.184. Strict-mode in_array() whitelist applied before persistence.

Finding 4 — Skin JS Scanner — Detail Field Stores Metadata, Not Raw Content [Informational]

Location: core/smackback.php, smack-smackback.php

The skin JS scanner stores its findings as JSON in snap_settings. Each finding record includes a 'detail' field. A naive implementation might store the raw matched line from the scanned file, which could contain executable content and would be rendered in the SMACKBACK admin panel. The actual implementation stores only structured metadata: for external scripts, the extracted hostname (from parse_url()) is stored; for other pattern types (eval, atob, document.write, inline scripts), a hardcoded descriptive string is stored. No raw file content is ever placed in the detail field. The render path in smack-smackback.php wraps all output in htmlspecialchars() in any case, so even if raw content were stored, stored XSS would be prevented. This is noted as an audit confirmation rather than a finding.

Verdict: Closed. By design. detail field stores structured metadata only. Output layer also protected by htmlspecialchars().

Finding 5 — Multisite — Spoke-Controlled TRACK Badge [Informational]

Location: smack-multisite.php — heartbeat handler, spoke fleet table

When a spoke checks in via heartbeat, it reports its own update_track value. The hub stores this in snap_ms_nodes.update_track and displays it as a BORING/BITCHIN' badge in the fleet roster. A compromised or misconfigured spoke could send any value for update_track. The badge display uses a PHP conditional: only the literal string 'dev' produces the BITCHIN' badge; any other value produces BORING. There is no interpolation of the stored value into the badge HTML. More importantly, the hub's own update decisions are made by

updater_get_track(), which reads from the hub's own snap_settings table — not from the heartbeat or from snap_ms_nodes. A spoke cannot influence whether the hub updates on stable or dev track. The badge is display-only.

Verdict: Closed. By design. Spoke-reported track affects display badge only. Hub update decisions are derived from the hub's own settings, not from heartbeat data.

Summary

#	Title	Sev.	Status	Fix
1	Skin JS scanner — symlink escape	Low	Closed	0.7.184
2	Archive redirect — HTTP_HOST fallback	Low	Closed	0.7.184
3	Settings save — enum fields unvalidated	Low	Closed	0.7.184
4	Skin JS scanner — detail field content	Info	Closed	By design
5	Multisite — spoke-controlled TRACK badge	Info	Closed	By design

Disposition

All five findings are Closed. Three real vulnerabilities were found and remediated in the same session. Two are informational notes confirming that defensive design choices are correctly implemented. 0.7.184 ships clean.

Audit Note

The initial pass of this audit (also labelled 019) returned a clean-all-informational result based on surface-level reading. Sean pushed back and requested a deeper audit. The second pass involved reading every modified function in full — smackback_scan_skin_js(), updater_fetch_release_info(), updater_get_track(), the index.php routing block, the smack-settings.php save handler, and the smack-smackback.php render path. That deeper read produced findings 1–3 above. The lesson: for a release touching file system traversal, redirect logic, and a generic save handler, a full function read is required.

End of audit 019. Next audit number: 020.