SnapSmack Security Audit Report

Codebase: C:\dev\snapsmack
Audit Date: 2026-04-25
Scope: Complete PHP codebase including core/, migrations/, smack-*.php, setup.php, install.php
Focus Areas: SQL injection, XSS, CSRF, authentication, authorization, command injection, file operations, path traversal, insecure deserialization, open redirects, input validation, rate limiting

CRITICAL Critical Issues 3

1. Open Redirect Vulnerability in community-auth.php

`File: community-auth.php, Line 31`

Issue: Unvalidated redirect parameter allows attackers to redirect users to arbitrary domains, enabling phishing attacks.

```
// Line 26-33 if ($action !== 'logout' && $action !== 'verify') {
$community_user = community_current_user(); if ($community_user) {
$redirect = $_GET['redirect'] ?? '/'; header('Location: ' . $redirect);
// NO VALIDATION! exit; } }
```

Attack Vector: A logged-in user visits `community-auth.php?redirect=http://evil.com` and gets redirected to a phishing site.

Fix: Validate the redirect URL to ensure it's a relative path or same-origin URL:

```
$redirect = $_GET['redirect'] ?? '/'; // Only allow relative paths or
same-origin if (!preg_match('~^/[^/]~', $redirect) &&
!filter_var($redirect, FILTER_VALIDATE_URL)) { $redirect = '/'; } if
(filter_var($redirect, FILTER_VALIDATE_URL) && parse_url($redirect,
PHP_URL_HOST) !== $_SERVER['HTTP_HOST']) { $redirect = '/'; }
header('Location: ' . $redirect);
```

2. Unescaped Output of Database Setting - Stored XSS

File: core/meta.php, Line 274

Issue: The `custom_head_scripts` setting is echoed directly into the page without any escaping, allowing stored XSS attacks.

```
// Line 272-274 if (!empty($settings['custom_head_scripts'])): ?>
```

Attack Vector: An admin saves malicious JavaScript in "Smack Your Scripts Up!" settings page. The JS executes on every page for every user.
Impact: This is intentional functionality (admin can inject custom scripts), but it creates a stored XSS vector if admin account is compromised.

Consideration: This is a design feature, not a bug. However, consider:

- Add warning in the admin UI that custom scripts are trusted content only
- Consider Content Security Policy (CSP) headers to mitigate impact of injected malicious scripts
- Add audit logging when this setting is modified
- Restrict access to smack-scripts.php to admin role only (not editor)

3. Logo File Upload Without Extension Validation

File: smack-settings.php, Lines 101-106

Issue: Logo upload accepts any file extension and moves it to web-accessible directory without proper MIME type validation.

```
// Line 101-106 $ext = strtolower(pathinfo($_FILES['logo_upload']
['name'], PATHINFO_EXTENSION)); $target_file = $target_dir . "logo." .
$ext; if (move_uploaded_file($_FILES['logo_upload']['tmp_name'],
$target_file)) { $_POST['settings']['header_logo_url'] = "/" .
$target_file; }
```

Attack Vector: Upload a PHP file as "logo.php" and execute arbitrary code, or upload an HTML file with XSS payload.

Fix: Validate file types strictly:

```
$allowed_logo_exts = ['png', 'jpg', 'jpeg', 'gif', 'svg', 'webp'];
$ext = strtolower(pathinfo($_FILES['logo_upload']['name'],
PATHINFO_EXTENSION)); if (!in_array($ext, $allowed_logo_exts)) {
$error = "Logo must be PNG, JPG, GIF, SVG, or WebP."; } else { //
Also validate MIME type $finfo = finfo_open(FILEINFO_MIME_TYPE);
```

```
$mime = finfo_file($finfo, $_FILES['logo_upload']['tmp_name']);
finfo_close($finfo); $allowed_mimes = ['image/png', 'image/jpeg',
'image/gif', 'image/svg+xml', 'image/webp']; if (!in_array($mime,
$allowed_mimes)) { $error = "Invalid image file."; } else { // Safe
to upload... } }
```

HIGH

High Severity Issues 4

1. Missing CSRF Token Validation in Admin Forms

```
File: smack-settings.php (and most smack-*.php files)
```

Issue: Form submission handlers check for `isset($_POST['save_settings'])` but don't validate CSRF tokens. Any website can craft a form that saves settings.

```
// Line 94 - NO CSRF TOKEN CHECK if (isset($_POST['save_settings'])) {
// Process form directly... }
```

Impact: An attacker can trick an admin into visiting a malicious website that silently modifies site settings, injects scripts, changes email addresses, etc.

Fix: Add CSRF token generation and validation to all admin forms:

```
// In PHP (before form output): if
(empty($_SESSION['admin_csrf_token'])) {
$_SESSION['admin_csrf_token'] = bin2hex(random_bytes(32)); } // In
form: // In POST handler: if (isset($_POST['save_settings'])) { if
(!hash_equals($_SESSION['admin_csrf_token'], $_POST['csrf_token'] ??
'')) { die('CSRF token mismatch'); } // Process... }
```

2. Potential Path Traversal in Skin Manifest Include (smack-edit.php)

```
File: smack-edit.php, Lines 40-50
```

Issue: While skin name is validated to /skins/{name}/manifest.php, the 'edit_page' value from manifest is concatenated with 'smack-edit-' without sanitization.

```
// Line 42 $_ss_manifest_path = __DIR__ . '/skins/' . $_ss_active_skin . '/manifest.php'; // ... // Line 48 $_ss_edit_file = __DIR__ . '/smack-edit-' . $_ss_edit_override . '.php';
```

Risk: If a skin manifest is compromised and contains `'edit_page' => '../../../etc/passwd'`, it could include arbitrary files (though limited to .php extension).

Fix: Validate the edit_page value against allowed list:

```
$allowed_edit_pages = ['solo', 'carousel', 'panorama']; if (in_array($_ss_edit_override, $allowed_edit_pages)) { $_ss_edit_file = __DIR__ . '/smack-edit-' . $_ss_edit_override . '.php'; // Safe to include }
```

3. Open Redirect in community-auth.php Line 182

```
File: community-auth.php, Line 182
```

Issue: Post-login redirect uses filter_var() to check if it's a URL, but this still allows redirects to any domain if the URL is valid.

```
// Line 181-183 header('Location: ' . (filter_var($redirect, FILTER_VALIDATE_URL) ? $redirect : '/'));
```

Attack Vector: User submits form with `redirect=http://attacker.com`, which passes FILTER_VALIDATE_URL and redirects them to the attacker's site.

Fix: Only allow relative paths:

```
// Only allow relative paths starting with / if (!preg_match('~^/[a-zA-Z0-9/_\-\.?=&]*$~', $redirect)) { $redirect = '/'; } header('Location: ' . $redirect);
```

4. Rate Limiting Not Enforced on Password Reset Emails

```
File: password-reset.php, Lines 51-69
```

Issue: No rate limiting on password reset requests. Attacker can enumerate valid user emails and flood them with reset emails (email DoS).

```
// Line 51-69 if ($step === 'request' && $_SERVER['REQUEST_METHOD'] === 'POST') { $email = strtolower(trim($_POST['email'] ?? '')); if (!filter_var($email, FILTER_VALIDATE_EMAIL)) { // ... } else { $reset_token = snapsmack_generate_reset_token($pdo, $email); // Email sent without rate limit check } }
```

Fix: Add rate limiting by IP address:

```
$ip = $_SERVER['REMOTE_ADDR']; $reset_limit = $pdo->prepare("SELECT COUNT(*) FROM snap_password_resets WHERE ip = ? AND created_at > DATE_SUB(NOW(), INTERVAL 1 HOUR)"); $reset_limit->execute([$ip]); if ($reset_limit->fetchColumn() >= 5) { $err = 'Too many reset requests. Please try again in an hour.'; } else { // Process reset... }
```

MEDIUM

Medium Severity Issues 3

1. Favicon Upload Missing MIME Type Validation

```
File: smack-settings.php, Lines 110-123
```

Issue: Favicon upload checks extension (ico, png, svg) but not actual file MIME type. A PHP file renamed to .png could be executed.

```
// Line 115-117 $fav_ext = strtolower(pathinfo($_FILES['favicon_upload']['name'], PATHINFO_EXTENSION)); $allowed_fav = ['ico', 'png', 'svg']; if (in_array($fav_ext, $allowed_fav)) { // No MIME type check before upload }
```

Fix: Validate MIME type before upload:

```
$finfo = finfo_open(FILEINFO_MIME_TYPE); $mime = finfo_file($finfo, $_FILES['favicon_upload']['tmp_name']); finfo_close($finfo); $allowed_mimes = ['image/x-icon', 'image/png', 'image/svg+xml']; if (!in_array($mime, $allowed_mimes)) { $error = 'Invalid favicon file'; }
```

2. Session Fixation Risk in 2FA Verification Flow

File: smack-2fa-verify.php, Lines 38-47

Issue: Session is not regenerated after successful 2FA verification. An attacker with an active session cookie could potentially bypass 2FA if they know a valid TOTP code.

```
// Line 35-46 if (empty($_SESSION['totp_pending_user_id'])) {
header("Location: login.php"); exit; } // Session ID is never
regenerated after successful TOTP verification
```

Fix: Regenerate session ID after successful 2FA:

```
// After TOTP verification succeeds: session_regenerate_id(true); //
Invalidate old session $_SESSION['user_login'] = $user['username'];
// ... rest of session setup
```

3. Weak Input Validation on Page Slug

File: install.php, Lines 238-251

Issue: Page slug field in snap_pages table allows any varchar content. Slug could contain path traversal characters or special characters.
Note: This is low-risk since pages are accessed via database queries, not direct file paths, but could cause issues if slug is used in URLs without proper encoding.

Recommendation: Add slug format validation when pages are created/updated in the admin UI.

LOW Low Severity Issues 2

1. Error Messages May Reveal Database Details

File: install.php, Line 144

Issue: PDOException messages are displayed directly to users during installation, potentially revealing database connection details.

```
// Line 144 $errors[] = 'Connection failed: ' . htmlspecialchars($msg);
```

Note: This is during installation (typically in private setup), but should be generic in production.

> **Recommendation:** Log detailed errors server-side; show generic messages to users in production.

2. Missing Content-Security-Policy Headers

```
All pages
```

Issue: No CSP headers are set site-wide. This limits protection against XSS attacks, especially given the custom_head_scripts feature.

> **Recommendation:** Add CSP headers in core/header.php or as middleware:
>
> ```
> header("Content-Security-Policy: default-src 'self'; script-src 'self' 'unsafe-inline'; style-src 'self' 'unsafe-inline';");
> ```
>
> *Note: 'unsafe-inline' is needed for existing inline scripts but should be minimized.*

Summary

Total Issues Found: 12

CRITICAL: 3 | **HIGH:** 4 | **MEDIUM:** 3 | **LOW:** 2

Recommendation: Address CRITICAL issues immediately before deploying to production. CRITICAL issues enable account compromise, arbitrary code execution, and data exfiltration. HIGH severity issues should be addressed in the next release cycle.

Positive Security Observations

- ✓ All PDO queries use prepared statements with parameterized queries (excellent SQL injection protection)
- ✓ Password reset tokens are properly validated and expire correctly

- ✓ TOTP 2FA implementation includes brute-force protection (5 attempt limit)
- ✓ All admin pages require authentication via core/auth.php
- ✓ Session security settings are properly configured (secure, httponly, samesite=Lax)
- ✓ File permission checks on .htaccess and session directory are in place
- ✓ Package signature verification in setup.php uses Ed25519 cryptography
- ✓ Community user passwords use PASSWORD_BCRYPT with cost=12
- ✓ ZIP extraction in setup.php validates against path traversal attacks