Risk ID	Technical Risk	Technical Risk Indicators	CVE/CWE/OSVDB ID	Impact Rating	Impact	Mitigation	Validation Steps
1	Form inputs allow for SQL Injection attacks	Excessive/missing/modified database contents, performance issues stemming from DB operations in app: board.php lines 30/56/62, includes/dblib.php line 56, scoreboard/index.php lines 50/60	CWE ID 89	H	Sensitive user data can be leaked to the public, important data can be deleted, and the databases can be bogged down with rows of meaningless inserted data	Use only pre-constructed queries so that malicious form input doesn't reach the database, sanitize inputs to scrub any characters or keywords that signify attacks (like single quotes, comment characters, INSERT/DELETE/etc.	In a contained or protected environment, attempt the attacks yourself (such that they're non-destructive to actual data) to confirm they no longer work
2	Hard-coded password used in application	Unintended access by non-administrative people to inner application in app: board.php line 18, includes/dblib.php line 6, scoreboard/index.php lines 34/114	CWE ID 259	H	A compromised password would put every instance on the market at risk for attack since the password coded into the application, and a fix would require a major patch	Password storage should not be inside application code - rather in some external database	Static analysis of code (using manpower or something like grep) to make sure all instances of passwords in code are scrubbed, analysis of logs for unaccounted-for instances of somebody logging in
3	Password stored in plaintext	Unintended access by non-administrative people to inner application in app: board.php line 15, includes/dblib.php line 3, scoreboard/index.php lines 31/111	CWE ID 256	H	Anybody looking at the source code of the application gains access to some part that they shouldn't where they could unleash attacks	Along with not storing the password in the application code, all instances of password use should be securely encrypted	Static analysis of code (using manpower or something like grep) to make sure all instances of passwords in code are scrubbed, analysis of logs for unaccounted-for instances of somebody logging in
4	XSS Vulnerabilities	Persistent inorganic JavaScript in application, content of application modified in app: board.php lines 43/44/50/58/59/64, scoreboard/index.php line 119	CWE ID 80	H	The application can be rendered unusable by excessive javascript interference or content change, and users are put at risk through cookie tampering, redirects, and information leaking	Use entity encoding on output based on the environment (ie HTML encoding for outputs to the HTML body), and/or sanitize user input to remove things like <script> tags or other malicious keywords	In a controlled environment, attempt non-malicious forms of the attacks to ensure they don't work as intended.
5	Information leaks through error messages	Dynamic error messages that may reveal information about DB structure, such as table names or locations in app: board.php line 18, includes/dblib.php lines 8/27, scoreboard/index.php lines 34/114	CWE ID 209	L	By directly outputting an error message to the user, you risk having the error message contain (and leak) sensitive information, such as DB structure	Use application-level pre-formed generic error messages to prevent leakage of data	Perform error on application and see that error message contains no compromising information