Risk ID	Technical Risk	Technical Risk Indicators	CVE, CWE, or OSVDB IDs	Impact Rating	Impact	Mitigation	Validation Steps
1	Code can be injected into the application via a function that dynamically evaluates user input as code	User input is supplied to a require() statement	CWE-98	H	Attacker can input code into the application that can be run; can potentially cause a number of problems, including data leakage, denial of service, and admin access for attacker	Do not execute user input as code; verify that input is of the expected format and that it does not contain code or that if it does, the code will not be evaluated	Input code into the application and verify that the code is not evaluated
2	SQL injection vulnerabilities present	User input is inserted into SQL query without any input validation in www/board.php lines 30, 56, and 62, www/includes/dblib.php line 23, www/scoreboard/index.php line 50, www/.../SimplePie/Cache/MySQL.php line 344	CWE-89	H	Attacker can query the database, allowing the attacker to view and modify data, as well as potentially to gain administrative access to the database. They may also gain access to the filesystem and have the ability to run system commands	Validate user input to ensure that it is of the proper format; parameterize SQL statements instead of dynamically constructing the queries	Input text that will finish SQL statements and cause unexpected results if unsanitized into input fields, e.g. input " ' or '1=1". If unexpected results are returned, the vulnerability still exists. Check that in the code no user input is used without first being validated and sanitized.
3	Credentials are hard-coded in	The code includes credentials such as usernames and passwords as variables or constants; see board.php lines 43, 44, 50, 58, 59, 64, scoreboard.php line 119	CWE-259	M	If the password is compromised, it cannot be easily changed because the source code itself will have to be changed, potentially allowing attackers to gain access	Do not hard-code credentials; store them separately from source code; store credentials in configuration files	Check that no credentials are written directly in the code
4	Cross-site scripting vulnerability exists at many user input endpoints	User input is inserted into page (e.g. in HTML tags) without first being validated; see board.php lines 43, 44, 50, 58, 59, 64, scoreboard.php line 119	CWE-80	M	Users' cookies can be stolen or modified; page content can be altered; content can be rendered that can make the page difficult or impossible to use, resulting in DoS	Validate and sanitize user input (e.g. by escaping specific characters) before rendering it on the page	Input a script tag, e.g. "<script>alert("Testing");</alert>", into a user input field and check if the script runs. If it does, the vulnerability still exists.
5	User can traverse directories	User input is used to reference files without validating the input	CWE-73	M	Attacker may be able to view and modify resources that are meant to be restricted; for example, if asked to choose a file name, they may be able to use "../" to specify a file in a different directory	Validate and sanitize user input before using it. If feasible, use a centralized data validation routine	Check that user input is not being used before being validated. Enter a file path that should be disallowed and check that the app does not allow the user to view or access in any other manner files outside of the intended directory
6	Information leakage due to an error message	A SQL error generated by MySQL is displayed directly to the user in board.php line 18, includes/dblib.php lines 8 and 27, scoreboard/index.php lines 34 and 114, wp-admin/plugins.php line 284, network/themes.php line 153	CWE-209	L	If the error message includes information that should be restricted, such as information about the structure of the database or information about a user, the attacker may be able to use that information to make a more direct attack. For example, if they find out the name of a table and the application is vulnerable to SQL injection, they may be able to construct queries that target that table	Instead of directly displaying error messages generated by MySQL, display custom error messages that do not contain any sensitive information. Only use the generated error messages for internal purposes; do not expose them to users.	Check that only generic error messages are displayed by examining the places in the code where errors are displayed. Acting as a user, force an error and make sure that the message displayed to you does not contain sensitive information
7	Relies on data in cookies that can be tampered with	main.php checks value of cookie 'lg' and changes page that is rendered based on the value of the cookie	CWE-472	M	The attacker may see information they are not supposed to have access too, but currently the information on the page is fairly innocuous.	Do not use data from cookies to determine whether or not to display sensitive information	Use a tampering application to view and modify the application's cookies and ensure that changing the value of the cookies does not expose any sensitive information and does not give the user privileges they should not have