Risk ID	Technical Risk	Technical Risk Indicators	Impact Rating	Impact	Mitigation	Validation Steps
1	SQL Injection	1. Veracode static analysis states that SQLi vulnerabilities can be found in the following files (assume ctf-f2014/www/): - board.php (line 30) - includes/dblib.php (line 23) - scoreboard/index.php (lines 50, 60) -wp-includes/SimplePie/Cache/MySQL.php (line 344) - wp-includes/wp-db.php (lines 795, 797) 2. SQLi was abused by our Risk Assessment Team during the CTF game to find flags 4, 5, 6, 7, and 8.	High	SQLi can give an attacker the ability to access or even alter data stored in the database. They may be able to see sensitive or off-limits data, and in the case of this site, they can access the admin account.	Validate user input in one or more of a number of ways, for example: - check for special characters used in SQL queries, such as the single quote, " ' " - use a "white list" to make sure that user input conforms to your requirements and expectations.	Attempt to input invalid characters into a submission box or the url bar; if your attacks fail, input should have been successfully validated.
2	Hard-Coded Password	1. Veracode static analysis states that passwords are hard-coded in the following files: - board.php (lines 15, 18) - includes/dblib.php (lines 3, 6) - scoreboard/index.php (lines 31, 34, 106, 109) - wp-admin/network/site-new.php (line 74)	Medium	Passwords (for example, the root password) are visible and usable to anyone with access to the source code. This is a serious security flaw, especially since the source code is publically available online.	This should be un-done. The code should be re-designed to function without a hard-coded password, and passwords (as well as other privileged information) should not be stored in the code in plain text.	Look at the code. If one or more passwords are still hard-coded in plain text, the risk is not yet mitigated.
3	Cross-Site Scripting (XSS)	1. Veracode static analysis states that at least 75 separate instances of XSS-vulnerable code are present in the source code. There are, frankly, too many instances too be listed here, and the Veracode static analysis report should be consulted for further details.	Medium	In an XSS attack, the site publishes some input from the user directly via the HTTP response, allowing an attacker to inject javascript (for example with <script> tags). This can be used to present other users with fraudulent or otherwise unwanted or innapropriate content, among other uses.	Like SQLi, the best way to handle XSS vulnerabilities is to validate/escape user input. Input should be vetted for special characters such as "<" and ">", <script> or other HTML tags, etc. Input containing illegal or suspicious characters should either be turned away or edited to prevent script execution.	Attempt to attack the site via XSS; if your attacks fail, input has been properly validated.
4	Weak Crypto	1. Veracode static analysis states that at least 95 lines in the source code utilize a weak or broken cryptographic algorithm. There are too many to list here; please see the static analysis report for more information.	Medium	Using a weak or broken algorithm to encrypt data presumably makes said data relatively easy to break into and access for an attacker with the proper know-how. Attackers should be prevented from accessing sensitive data whenever possible.	Use stronger cryptographic algorithms. This includes not "rolling your own" crypto (i.e. not using a proprietary system over a tried-and-true, mathematically sound algorithm).	Hard to verify; use a tool such as Veracode's static analysis to determine the relative strength of your algorithm(s).

5	Directory Traversal	1. Veracode static analysis lists 3 vulnerable instances in the source: -wp-admin/includes/class-wp-upgrader.php (line 1780) -wp-includes/Text/Diff/Engine/shell.php (lines 42, 43) 2. Our Risk Assessment Team located flags 1 and 9 during the CTF game using directory traversal	Medium	The code uses filenames/paths contructed either totally or partially from user-supplied input. This may allow an attacker to access files outside of the area to which their access should be restricted (for example, using ../../../ and so on may give an attacker access to system files).	Validate user input. Ensure that all user input does not contain dangerous characters, or that such characters are escaped. Treat all user input as if its source is malicious.	Attempt to input invalid characters and observe the result to determine whether input has been properly validated.
6	Information Leakage Through Error Messages	1. Veracode static analysis lists 7 lines in the source where this vulnerability may occur; see the analysis report for details.	Low	Error messages visible to the user may give out information to which a user should not be privy (for example, SQL error messages that may give an attacker information regarding the structure of the database). Typically, this information does not represent a fully exploitable vulnerability on its own, but it may help an attacker gain access when coupled with another vulnerability.	Ensure that error messages provide only relevant information that should be available to anyone who might be able to receive the messages.	Look in the code and at the content of error messages.