Risk Table for Capture the Flag 2015 - Using Vercode Static Analysis, MITRE CWE Database

Column Name	Information	SQL Injection	Improper File Inclusion	Cross Site Scripting (XSS)	User Controlled File System Operations	User Evaluated Code ('eval')	Sensitive information stored and sent unencrypted	Information Exposure in Error Message	Broken/Risky cryptographic algorithm used	Initialization based on sources that are not checked for authenticity
Risk ID	Sequential identifier for risk	1	2	3	4	5	6	7	8	9
Technical Risk	Brief Description	Input is not sanitized so user input can contain SQL code. Examples found in *MySQL.php:344* configuration file, *board.php:30*, *index.php:50*	Files included are done so in a method that is not protected. Examples found in configuration files in *update.php:90*	User input can contain code that can modify what other users see. Examples found in *user-edit.php: 234*, *user-new.php: 299*, *board.php: 64* where user input is taken and stored with sanitization	User can change certain paths use for operations. Examples found in *shell.php: 43* where a php file is loaded using a variable location	Code can be evaluated that was not originally developed for usage. Examples found in plugins used (silverlight)	If source code is exposed passwords are visible. Examples shown in *index.php: 34, index.php: 111, board.php: 18,* where passwords are in plaintext within the source code	Technical details are exposed in error message. Examples found in *board.php: 18* and *index.php: 114* where explicit MySQL errors are returned unsanitized (could contain sensitive info)	Brute forcing of passwords is possible. Examples found in *user-new.php:75* where MD5 is used to hash the password of a user	If files on the server are modified, new code can be loaded. Examples found in *shell.php: 50* where a shell_exec is used based on variables which could be manipulated
Techical Risk Indicators	Evidence of the risk occurring.	Bad data in SQL server, improper access in logs	Server logs indicate retrieval of unexpected files	Pages shown on page (such as a chat board) redirect or do other actions not defined by the server code	Server log show configuration changes outside of scope of intended development	Server log shows code being transmitted by users, website runs code that wasn't intended	Logins from unknown sources	Server shows personal user data, or technical information about technology used	Multiple attempts of incorrect logins or logins from strange locations	Files on server show that they have been modified by users (don't allow server as root so this can be apparent)
Related CVE, CWE, or OSVDB IDs	Use comma to separate multiple IDs	CWE-89	CWE-98	CWE-80	CWE-73	CWE-95	CWE-259, CWE-311	CWE-209	CWE-327	CWE-454
Impact Rating	Use NIST standard: (H)igh, (M)edium, (L)ow	H	M	M	M	L (mostly because the source is from silverlight, not the direct source code)	M	L	M	L
Impact	Possible results of the risk, impacting the goals of the product.	Data leakage, data loss	User could be redirected to incorrect module code	User could be redirected to malicious site	Configuration files (and directories) used could be changed by a user	Data loss, evaluation of arbitrary code which depending on server rights may be able to do almost anything	Data loss, data manipulated, data leakage	Data may be exposed to users about technology used or other users data	Broken authentication, malicious users may be able to access system	Information of site may not appear as desired - a malicious user could perform many kinds of actions
Mitigation	Action taken to reduce the probability of the risk actually occurring (or how to actually fix the bug). There may be more than one way to mitigate a risk.	Sanitize Input for SQL code (convert before passing to SQL database), i.e.: when getting the id for the board, sanitize $_GET["id"]	setting allow_url_fopen to false	Sanitize Input for code such as Javascript (convert before performing any actions)	Remove cases where a URL takes in a field where a user can manipulate the file used on the backend (normally this can be avoided)	Sanitize input, remove usages of eval, especially on places where user can input data (which could be a function to evaluate). For this particular case remove silverlight	Store passwords outside of code base and strongly encrypt the configuration files that contain sensitive data	Sanitize the error messages before they are sent to the user (build them manually and only expose necessary information)	Use more secure forms of cryptography (since algorithms can be broken over time, this must be constantly monitored if security is a priority for the application)	Avoid external variable initialization or trust the files that initialize
Validation Steps	How do you ensure that risk was mitigated?	Attempt SQL code on login and other pages that use SQL calls	Attempt to pass in a module name variable to a chosen source	Put javascript code in an input, verify that the code is not executed	Attempt to change a configuration file on the URL susceptible (ie: WP Attachment edit)	Send user function to URL that contain this issue. For this case verify that silverlight has been removed as the current version has an eval security bug	Search for all instances of important data throughout code base (ie: usernames and passwords for database, etc)s	Verify server log has no user data or technical details (easier said than done)	Code base uses improved cryptographic technique	Attempt to change external files and verify that server reports an error