Technical Risk Analysis on Capture the Flag

Carter Casey

Note: The extra categories and descriptions below are provided by Veracode's report. I thought it would be a good way to organize the tables, and potentially make it more readable.

High (8 Flaws)

Code Injection (1 flaw)

Code injection is the process of injecting untrusted input into an application that dynamically evalutes and executes the input as code. Common examples of code injection include Remote File Includes and Eval Injection into applications implemented in an interpreted language such as PHP.

Do not allow untrusted input to be evaluated or otherwise interpreted as code.

Risk ID	1
Locations	`ctf-f2014/www/wp-admin/update.php`
Technical Risk	Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion'): CWE ID 98 (1 Flaw)
Techical Risk Indicators	The PHP application receives user-supplied input but does not properly restrict the input before using it in require(), include(), or similar functions.
Impact Rating	H
Impact	Can allow an attacker to specify a URL to a remote location from which the application will retrieve code and execute it.
Mitigation	Validate all user-supplied input to ensure that it conforms to the expected format, using centralized data validation routines when possible. Use white lists to specify known safe values rather than relying on black lists to detect malicious input.
Validation Steps	Attempt XSS attacks on your own website. Review all points of access to the site, verify that the input is sanitized.

SQL Injection (7 flaws)

SQL injection vulnerabilities occur when data enters an application from an untrusted source and is used to dynamically construct a SQL query. This allows an attacker to manipulate database queries in order to access, modify, or delete arbitrary data. Depending on the platform, database type, and configuration, it may also be possible to execute administrative operations on the database, access the filesystem, or execute arbitrary system commands. SQL injection attacks can also be used to subvert authentication and authorization schemes, which would enable an attacker to gain privileged access to restricted portions of the application.

Several techniques can be used to prevent SQL injection attacks. These techniques complement each other

and address security at different points in the application. Using multiple techniques provides defense-in-depth and minimizes the likelihood of a SQL injection vulnerability.

Risk ID	2

Locations	`ctf-f2014/www/board.php`
	`ctf-f2014/www/includes/dblib.php`
	`ctf-f2014/www/scoreboard/index.php`
	`ctf-f2014/www/wp-includes/SimplePie/Cache/MySQL.php`
	`ctf-f2014/www/wp-includes/wp-db.php`

Technical Risk	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection'): CWE ID 89 (7 Flaws)
Techical Risk Indicators	This database query contains a SQL injection flaw. The function call constructs a dynamic SQL query using a variable derived from user-supplied input.
Impact Rating	H
Impact	An attacker could exploit this flaw to execute arbitrary SQL queries against the database. This means getting private data, or manipulating data in the database.
Mitigation	Avoid dynamically constructing SQL queries. Instead, use parameterized prepared statements to prevent the database from interpreting the contents of bind variables as part of the query. Always validate user-supplied input to ensure that it conforms to the expected format, using centralized data validation routines when possible.
Validation Steps	Attempt sql injections on the site (one option here is sqlmap). Verify that all sql queries are properly sanitized.

Medium (188 Flaws)

Credentials Management (9 flaws)

Improper management of credentials, such as usernames and passwords, may compromise system security. In particular, storing passwords in plaintext or hard-coding passwords directly into application code are design issues that cannot be easily remedied. Not only does embedding a password allow all of the project's developers to view the password, it also makes fixing the problem extremely difficult. Once the code is in production, the password cannot be changed without patching the software. If a hard-coded password is compromised in a commercial product, all deployed instances may be vulnerable to attack, putting customers at risk.

Avoid storing passwords in easily accessible locations, and never store any type of sensitive data in plaintext. Avoid using hard-coded usernames, passwords, or hash constants whenever possible, particularly in relation to security-critical components. Store passwords out-of-band from the application code. Follow best practices for protecting credentials stored in alternate locations such as configuration or properties files.

Risk ID	3

Locations	ctf-f2014/www/board.php ctf-f2014/www/includes/dblib.php ctf-f2014/www/scoreboard/index.php ctf-f2014/www/wp-admin/network/site-new.php
Technical Risk	Use of Hard-coded Password: CWE ID 259 (9 Flaws)
Techical Risk Indicators	A method uses a hard-coded password that may compromise system security in a way that cannot be easily remedied. The use of a hard-coded password significantly increases the possibility that the account being protected will be compromised.
Impact Rating	M
Impact	The password cannot be changed without patching the software. If a hard-coded password is compromised in a commercial product, all deployed instances may be vulnerable to attack. An attacker can find the password by gaining access to only one of the files, and need not rely on any decription, as the password is already in plain-text.
Mitigation	Store passwords out-of-band from the application code. Follow best practices for protecting credentials stored in locations such as configuration or properties files. Furthermore, store passwords using encryption.
Validation Steps	A simple grep for all your known passwords should reveal whether those passwords are stored anywhere in your application.

Cross-Site Scripting (75 flaws)

Cross-site scripting (XSS) attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. Flaws that allow these attacks to succeed occur whenever a web application uses untrusted data in the output it generates without validating or encoding it. XSS vulnerabilities are commonly exploited to steal or manipulate cookies, modify presentation of content, and compromise sensitive information, with new attack vectors being discovered on a regular basis. XSS is also commonly referred to as HTML injection.

Several techniques can be used to prevent XSS attacks. These techniques complement each other and address security at different points in the application. Using multiple techniques provides defense-in-depth and minimizes the likelihood of a XSS vulnerability.

Risk ID	4
Locations	ctf-f2014/www/wp-admin/includes/ajax-actions.php ctf-f2014/www/wp-admin/async-upload.php ctf-f2014/www/board.php ctf-f2014/www/wp-admin/includes/class-ftp-pure.php ctf-f2014/www/wp-admin/includes/class-ftp-sockets.php ctf-f2014/www/wp-admin/includes/class-ftp.php ctf-f2014/www/wp-includes/class-phpmailer.php ctf-f2014/www/wp-includes/class-smtp.php ctf-f2014/www/wp-admin/includes/class-wp-comments-list-table.php ctf-f2014/www/wp-includes/class-wp-editor.php ctf-f2014/www/wp-admin/includes/class-wp-list-table.php ctf-f2014/www/wp-admin/includes/class-wp-ms-users-list-table.php

	ctf-f2014/www/wp-admin/edit-comments.php ctf-f2014/www/wp-admin/includes/file.php ctf-f2014/www/wp-admin/import.php ctf-f2014/www/scoreboard/index.php ctf-f2014/www/wp-admin/install.php ctf-f2014/www/wp-admin/link-manager.php ctf-f2014/www/wp-admin/load-scripts.php ctf-f2014/www/wp-admin/load-styles.php ctf-f2014/www/wp-includes/media-template.php ctf-f2014/www/wp-admin/media.php ctf-f2014/www/wp-admin/includes/media.php ctf-f2014/www/wp-admin/my-sites.php ctf-f2014/www/wp-admin/includes/nav-menu.php ctf-f2014/www/wp-admin/plugin-editor.php ctf-f2014/www/wp-admin/plugins.php ctf-f2014/www/wp-admin/includes/post.php ctf-f2014/www/wp-admin/press-this.php ctf-f2014/www/wp-admin/setup-config.php ctf-f2014/www/wp-admin/theme-editor.php ctf-f2014/www/wp-admin/network/themes.php ctf-f2014/www/wp-admin/themes.php ctf-f2014/www/wp-admin/update-core.php ctf-f2014/www/wp-admin/upgrade.php ctf-f2014/www/wp-admin/upload.php ctf-f2014/www/wp-admin/user-edit.php ctf-f2014/www/wp-admin/user-new.php ctf-f2014/www/wp-admin/network/users.php ctf-f2014/www/wp-admin/widgets.php ctf-f2014/www/wp-includes/js/tinymce/wp-tinymce.php
Technical Risk	Improper Neutralization of Script-Related HTML Tags in a Web Pag (Basic XSS): CWE ID 80 (75 Flaws)
Techical Risk Indicators	A function call contains a cross-site scripting (XSS) flaw. The application populates the HTTP response with user-supplied input, allowing an attacker to embed malicious content, such as Javascript code, which will be executed in the context of the victim's browser.
Impact Rating	M
Impact	XSS vulnerabilities are commonly exploited to steal or manipulate cookies, modify presentation of content, and compromise confidential information, with new attack vectors being discovered on a regular basis. By changing the script of the web page, the attacker can change the behavior of the site itself. It can redirect your browser to other pages, attempt to pull information from your browser, etc.
Mitigation	Use contextual escaping on all untrusted data before using it to construct any portion of an HTTP response. As a best practice, always validate user-supplied input to ensure that it conforms to the expected format, using centralized data validation routines when possible.

Validation Steps	Again, attempt XSS attacks yourself. Verify that all HTML tags are properly verified and sanitized.

Cryptographic Issues (100 flaws)

Applications commonly use cryptography to implement authentication mechanisms and to ensure the confidentiality and integrity of sensitive data, both in transit and at rest. The proper and accurate implementation of cryptography is extremely critical to its efficacy. Configuration or coding mistakes as well as incorrect assumptions may negate a large degree of the protection it affords, leaving the crypto implementation vulnerable to attack.

Select the appropriate type of cryptography for the intended purpose. Avoid proprietary encryption algorithms as they typically rely on "security through obscurity" rather than sound mathematics. Select key sizes appropriate for the data being protected; for high assurance applications, 256-bit symmetric keys and 2048-bit asymmetric keys are sufficient. Follow best practices for key storage, and ensure that plaintext data and key material are not inadvertently exposed.

Risk ID	5
Locations	`ctf-f2014/www/wp-admin/includes/class-ftp-sockets.php` `ctf-f2014/www/wp-admin/includes/class-wp-filesystem-ftpext.php`
Technical Risk	Missing Encryption of Sensitive Data: CWE ID 311 (3 Flaws)
Techical Risk Indicators	The application exposes potentially sensitive data by passing it into a function unencrypted.
Impact Rating	Could allow private data such as cryptographic keys or other sensitive information to be exposed.
Impact	M
Mitigation	Ensure that the application protects all sensitive data from unnecessary exposure. In other words, encrypt sensitive data, or just don't pass it around.
Validation Steps	

Risk ID	6

Locations	`ctf-f2014/www/wp-includes/SimplePie/Author.php`
	`ctf-f2014/www/wp-includes/bookmark.php`
	`ctf-f2014/www/wp-includes/SimplePie/Caption.php`
	`ctf-f2014/www/wp-includes/SimplePie/Category.php`
	`ctf-f2014/www/wp-admin/includes/class-pclzip.php`
	`ctf-f2014/www/wp-includes/class-phpass.php`
	`ctf-f2014/www/wp-includes/class-phpmailer.php`
	`ctf-f2014/www/wp-includes/class-pop3.php`
	`ctf-f2014/www/wp-includes/class-simplepie.php`
	`ctf-f2014/www/wp-includes/class-smtp.php`

	ctf-f2014/www/wp-includes/class-snoopy.php ctf-f2014/www/wp-includes/class-wp-embed.php ctf-f2014/www/wp-admin/includes/class-wp-ms-themes-list-table.php ctf-f2014/www/wp-admin/includes/class-wp-plugins-list-table.php ctf-f2014/www/wp-includes/class-wp-theme.php ctf-f2014/www/wp-admin/includes/class-wp-upgrader.php ctf-f2014/www/wp-includes/class-wp.php ctf-f2014/www/wp-includes/comment.php ctf-f2014/www/wp-includes/SimplePie/Copyright.php ctf-f2014/www/wp-includes/SimplePie/Credit.php ctf-f2014/www/wp-includes/cron.php ctf-f2014/www/wp-admin/includes/dashboard.php ctf-f2014/www/wp-includes/default-constants.php ctf-f2014/www/wp-includes/SimplePie/Enclosure.php ctf-f2014/www/wp-admin/includes/file.php ctf-f2014/www/wp-includes/general-template.php ctf-f2014/www/wp-includes/ID3/getid3.php ctf-f2014/www/wp-includes/SimplePie/gzdecode.php ctf-f2014/www/wp-admin/includes/image.php ctf-f2014/www/wp-includes/SimplePie/Item.php ctf-f2014/www/wp-includes/link-template.php ctf-f2014/www/wp-includes/SimplePie/Cache/Memcache.php ctf-f2014/www/wp-includes/ID3/module.tag.apetag.php ctf-f2014/www/wp-includes/ID3/module.tag.id3v2.php ctf-f2014/www/wp-includes/ms-blogs.php ctf-f2014/www/wp-includes/ms-files.php ctf-f2014/www/wp-includes/ms-functions.php ctf-f2014/www/wp-admin/includes/ms.php ctf-f2014/www/wp-includes/pluggable.php ctf-f2014/www/wp-admin/includes/plugin-install.php ctf-f2014/www/wp-includes/post.php ctf-f2014/www/wp-includes/query.php ctf-f2014/www/wp-includes/SimplePie/Rating.php ctf-f2014/www/wp-includes/SimplePie/Restriction.php ctf-f2014/www/wp-includes/rss.php ctf-f2014/www/wp-admin/includes/schema.php ctf-f2014/www/wp-includes/SimplePie/Source.php ctf-f2014/www/wp-includes/taxonomy.php ctf-f2014/www/wp-admin/includes/update-core.php ctf-f2014/www/wp-admin/includes/upgrade.php ctf-f2014/www/wp-admin/user-new.php
Technical Risk	Use of a Broken or Risky Cryptographic Algorithm: CWE ID 327 (95 Flaws)
Techical Risk Indicators	The use of a broken or risky cryptographic algorithm is an unnecessary risk that may result in the disclosure of sensitive information.
Impact Rating	M
Impact	An attacker with access to the encrypted data could decrypt the data with potentially unnecessary ease.
Mitigation	Fairly simply, use better encryption algorithms. A list of broken algorithms can be found on the web.
Validation Steps	Verify that all encryption algorithms meet proper standards (don't use broken algorithms like SHA-1).

Directory Traversal (4 flaws)

Allowing user input to control paths used in filesystem operations may enable an attacker to access or modify otherwise protected system resources that would normally be inaccessible to end users. In some cases, the user-provided input may be passed directly to the filesystem operation, or it may be concatenated to one or more fixed strings to construct a fully-qualified path.

Assume all user-supplied input is malicious. Validate all user-supplied input to ensure that it conforms to the expected format, using centralized data validation routines when possible. When using black lists, be sure that the sanitizing routine performs a sufficient number of iterations to remove all instances of disallowed characters and ensure that the end result is not dangerous.

Risk ID	7
Locations	`ctf-f2014/www/wp-admin/includes/class-wp-upgrader.php` `ctf-f2014/www/wp-includes/Text/Diff/Engine/shell.php`
Technical Risk	External Control of File Name or Path: CWE ID 73 (4 Flaws)
Techical Risk Indicators	This call contains a path manipulation flaw. The argument to the function is a filename constructed using user-supplied input.
Impact Rating	M
Impact	If an attacker is allowed to specify all or part of the filename, it may be possible to gain unauthorized access to files on the server, including those outside the webroot, that would be normally be inaccessible to end users. In essence, an attacker can get to files that aren't even part of the website on the hosting server.
Mitigation	Validate all user-supplied input to ensure that it conforms to the expected format, using centralized data validation routines when possible. When using black lists, be sure that the sanitizing routine performs a sufficient number of iterations to remove all instances of disallowed characters. Again, the core idea is to sanitize input.
Validation Steps	At all places in your code that require a pathname, verify that it doesn't come unchecked from a user.

Low (7 Flaws)

Information Leakage (7 flaws)

An information leak is the intentional or unintentional disclosure of information that is either regarded as sensitive within the product's own functionality or provides information about the product or its environment that could be useful in an attack. Information leakage issues are commonly overlooked because they cannot be used to directly exploit the application. However, information leaks should be viewed as building blocks that an attacker uses to carry out other, more complicated attacks.

Configure applications and servers to return generic error messages and to suppress stack traces from being displayed to end users. Ensure that errors generated by the application do not provide insight into specific backend issues.

Risk ID	8

Locations	`ctf-f2014/www/board.php`
	`ctf-f2014/www/includes/dblib.php`
	`ctf-f2014/www/scoreboard/index.php`
	`ctf-f2014/www/wp-admin/plugins.php`
	`ctf-f2014/www/wp-admin/network/themes.php`

Technical Risk	Information Exposure Through an Error Message: CWE ID 209 (7 Flaws)
Techical Risk Indicators	The software generates an error message that includes sensitive information about its environment, users, or associated data. The sensitive information may be valuable information on its own (such as a password), or it may be useful for launching other, more harmful attacks.
Impact Rating	L
Impact	If an attack fails, an attacker may use error information provided by the server to launch another more focused attack. For example, if a stack traces lists file paths, the attacker could use those for a direct traversal attack.
Mitigation	Ensure that only generic error messages are returned, which don't reveal any additional details. In this case, the message is sanitize your output.
Validation Steps	Attempt to cause errors in your program, and see what error messages are produced on the user's end.

Untrusted Initialization (8 flaws)

Applications should be reluctant to trust variables that have been initialized outside of its trust boundary. Untrusted initialization refers to instances in which an application allows external control of system settings or variables, which can disrupt service or cause an application to behave in unexpected ways. For example, if an application uses values from the environment, assuming the data cannot be tampered with, it may use that data in a dangerous way.

Compartmentalize the application and determine where the trust boundaries exist, then treat any input or control outside the trust boundary as potentially hostile. In general, do not allow user-provided or otherwise untrusted data to control sensitive values.

Risk ID	9
Locations	`ctf-f2014/www/wp-includes/class-phpmailer.php` `ctf-f2014/www/wp-includes/ID3/getid3.lib.php`

	`ctf-f2014/www/wp-includes/ID3/getid3.php` `ctf-f2014/www/wp-includes/Text/Diff/Engine/shell.php`
Technical Risk	External Initialization of Trusted Variables or Data Stores: CWE ID 454 (8 Flaws)
Techical Risk Indicators	A function is used to process options passed into a command line application. The optarg variable is used to store any additional arguments that an option requires.
Impact Rating	L
Impact	If optarg is used in an unbounded string copy, an attacker can specify overly long command line arguments and overflow the destination buffer, potentially resulting in execution of arbitrary code. Basically, this is only an issue if there's a buffer overflow problem – if strcpy is used in the program and is used on the command line arguments.
Mitigation	Be sure to limit the size of data copied from the optarg variable.
Validation Steps	Attempt to pass in a huge volume of command-line arguments. If the space used by the program doesn't balloon out to a larger amount than usual, you should be safe.