|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Risk ID | Technical Risk | Risk Indicators | Impact Rating | Impact | Mitigation | Validation Steps |
| CWE ID 121 | Stack Buffer Overflow | Use of unsafe memory allocation | Very High | Arbitrary Code Execution | Always use bounded rather than unbounded string manipulation functions, e.g. strncpy() and strncat() instead of strcpy() and strcat().Ensure that buffers are of sufficient size, either by allocating it dynamically or by restricting the size to a reasonable length. | Use stack smashing protection when compiling code. |
| CWE ID 95 | Code Injection |  | Very High |  | 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. | Validate all user-supplied input to ensure that it conforms to the expected format, using centralized data validation routines when possible. In general, avoid executing code derived from untrusted input. |
| CWE ID 89 | SQL Injection |  | High |  | 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. | 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. |
| CWE ID 259 | Hard-Coded Password |  | Medium |  | 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. | Store passwords out-of-band from the application code. Follow best practices for protecting credentials stored in locations such as configuration or properties files. |
| CWE ID 80 | Basic XSS |  | Medium |  | 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. | Use output filtering to sanitize all output generated from user-supplied input, selecting the appropriate method of encoding based on the use case of the untrusted data. For example, if the data is being written to the body of an HTML page, use HTML entity encoding. However, if the data is being used to construct generated Javascript or if it is consumed by client-side methods that may interpret it as code (a common technique in Web 2.0 applications), additional restrictions may be necessary beyond simple HTML encoding. Validate user-supplied input using positive filters (white lists) to ensure that it conforms to the expected format, using centralized data validation routines when possible. |
| CWE ID 209 | Information Exposure through an Error Message |  | Low |  | 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. | Ensure that only generic error messages are returned to the end user that do not reveal any additional details. |