**Overall Checks:**

1. Any code-level constructs that are used in support of security should be consistently marked with in-line documentation.

|  |
| --- |
| **Category: Input Data** |
| **Vulnerabilities/Checks:**   1. Buffer overflows: Identify critical data structures and how overflows are handled 2. Using non-validated input in the Hypertext Markup Language (HTML) output stream 3. Using non-validated input used to generate SQL queries 4. Relying on client-side validation: Check that every client-side check is replicated on server side 5. Using input file names, URLs, or user names for security decisions or as input for file or XPath access 6. Trusting data read from databases, file shares, and other network resources 7. Failing to validate input from all sources including cookies, query string parameters, HTTP headers, databases, and network resources 8. When XML is being processed    1. What defenses are in place to protect the parser?    2. What measures are in place to ensure parser-related processing is managed? 9. For every trust boundary location:    1. Specify input: length, range, format, type, etc.    2. Implement layered defenses for specification    3. Use whitelisting wherever possible. Identify any location where whitelisting cannot be used    4. Examine resulting error response: Constrain, reject, and/or sanitize input Shift to Error Handling category for each error response       1. If sanitization used, ensure justified |

|  |
| --- |
|  |

|  |
| --- |
| **Category: Sensitive Data** |
| **Vulnerabilities/Checks:**   1. Has a complete audit of application been performed to identify and characterize data sensitivity? 2. Identify where accessing sensitive data in storage    1. Include temporary locations such as passivated session beans and swap files 3. Identify where accessing sensitive data in memory (including process dumps) 4. Review for storing secrets when you do not need to 5. Review for storing secrets in code 6. Review for secrets in clear text 7. Review for passing sensitive data in clear text over networks |

|  |
| --- |
| **Category: Cryptography** |
| **Vulnerabilities/Checks:**   1. Ensure organization's cryptography resources have been involved in design and implementation 2. Identify every location in application and application's infrastructure where cryptography is being used. 3. Ensure using well-vetted cryptographic suites 4. Ensure proper management of decryption keys 5. Ensure proper management of encryption keys 6. Ensure using appropriate algorithms or key sizes that are too small 7. Ensure not using the same key for a prolonged period of time. Ensure keys are periodically changed (periodicity based on sensitivity and exposure). 8. Ensure not distributing keys in an insecure manner 9. Ensure use of the **true secure random** method to generate random numbers 10. Identify and support critical review of any custom key management. |

|  |
| --- |
| **Category: Session Management** |
| **Vulnerabilities/Checks:**   1. Check all session lifetimes, keeping them to minimum length to accomplish associated task 2. Check that all locations where sessions are used provide explicit logout option for user. 3. Ensure judicious use of sessions 4. Ensure use of well-vetted session management and ID generation 5. Ensure session IDs are random 6. Ensure forensics capture in place for invalid session submission 7. Review for insecure session state stores 8. Review for placing session identifiers in query strings 9. Review for proper partitioning of site by anonymous, identified, and authenticated users 10. Review for storing sensitive data in session stores 11. Check for secure the channel to the session store 12. Authenticate and authorize access to the session store |

|  |
| --- |
| **Category: Auditing and Logging** |
| **Vulnerabilities/Checks:**   1. Check for proper and well-managed auditing of failed logons. Ensure candidate usernames and passwords are handled from a sensitivity and unvalidated input perspective 2. Ensure properly securing audit files based on sensitivity of information 3. Check auditing across application tiers where applicable 4. Support forensics on recognized error conditions 5. Use application instrumentation to expose behavior that can be monitored |

|  |
| --- |
| **Category: Authentication** |
| **Vulnerabilities/Checks:**   1. Is there a username and password policy in place? Is it going to promote strong passwords? Does it comply with current best practices at CERN or OWASP? Does the policy preclude a successful dictionary attack? 2. Is the username/password policy being enforced at username and password generation? 3. Is the username/password policy being used as part of layered defense on candidate usernames and password? 4. Is there an industrial strength rate limiting defense in place for authentication that provides both a defense and does not lead to a denial of service scenario? 5. Are all candidate usernames and passwords protected? 6. Ensure user is provided with last login date and time after a successful authentication process. 7. Check for passing single sign on (SSO) over unencrypted channels as well as having unencrypted SSO information in token. 8. Check for use well-vetted SSO mechanisms. 9. Check that any server-side process using session IDs to maintain state also use synchronizing tokens to recognize replay attacks and forceful browsing. 10. Do not store any credentials 11. Use authentication mechanisms that do not require clear text credentials to be passed over the network 12. Encrypt communication channels to secure authentication tokens 13. Separate anonymous from authenticated resources 14. Examine where every password is acquired during and/or after initial install and anticipated patch processes 15. Are there any default accounts? 16. Is application using a separate authentication service? If so, is the application still taking adequate measures to defend that resource from denial of service attacks? |

|  |
| --- |
| **Category: Authorization** |
| **Vulnerabilities/Checks:**   1. Ensure have proper segmentation of resources, functionality, and data in terms of varying levels of privileges 2. Ensure database access is as limited as possible 3. Appropriate process and user permission levels in runtime environment 4. Use least privilege accounts |

|  |
| --- |
| **Category: Exception Handling** |
| **Vulnerabilities/Checks:**   1. For each exception:    1. Examine information being returned to user from sensitivity and information leakage perspective    2. Examine resources required to generate response being returned to error source. Ensure properly managed from denial of service perspective    3. Examine information being captured on server side relative to error to ensure content is not sensitive, properly handled since it is not validated, and provides usable information for forensics and diagnostics. Consider using admin controlled graduated information capture.    4. Examine resources required to generate response being captured on server side. Ensure properly managed from denial of service perspective 2. Examine all third party software being used to ensure error response is appropriate 3. In code:    1. Use structured exception handling (by using **try/catch** blocks)    2. Catch and wrap exceptions only if the operation adds value/information |

|  |
| --- |
| **Category: Configuration Management** |
| **Vulnerabilities/Checks:**   1. Review for segmented and additional defenses for administration interfaces 2. Review for access to configuration stores 3. Review for any retrieval of clear text configuration secrets 4. Review for over-privileged process and service accounts 5. Having too many administrators 6. Avoid storing sensitive information in the Web space 7. For each software component:    1. Examine initial installation for default accounts or installer-provided overrides (overrides should be replaced with installer-provided values)    2. Identify that debugging, error handling, diagnostic, and auditing functions are properly configured in default state. |