Secure Coding Practices Checklist

Input validation

☐Conduct all input validation on a trusted system (server side not client side)
☐Identify all data sources and classify them into trusted and untrusted
☐Validate all data from untrusted sources (databases, file streams, etc)
☐Use a centralized input validation routine for the whole application
☐Specify character sets, such as UTF-8, for all input sources (canonicalization)
☐Encode input to a common character set before validating
☐All validation failures should result in input rejection
☐If the system supports UTF-8 extended character sets and validate after UTF-8 decoding is completed
☐Validate all client provided data before processing
☐Verify that protocol header values in both requests and responses contain only ASCII characters
☐Validate data from redirects
☐Validate for expected data types using an "allow" list rather than a "deny" list
☐Validate data range
☐Validate data length
☐If any potentially hazardous input must be allowed then implement additional controls
☐If the standard validation routine cannot address some inputs then use extra discrete checks
☐Utilize canonicalization to address obfuscation attacks

Output encoding

☐Conduct all output encoding on a trusted system (server side not client side)
☐Utilize a standard, tested routine for each type of outbound encoding
☐Specify character sets, such as UTF-8, for all outputs
☐Contextually output encode all data returned to the client from untrusted sources
☐Ensure the output encoding is safe for all target systems
☐Contextually sanitize all output of un-trusted data to queries for SQL, XML, and LDAP
☐Sanitize all output of untrusted data to operating system commands

Authentication and password management

☐Require authentication for all pages and resources, except those specifically intended to be public
☐All authentication controls must be enforced on a trusted system
☐Establish and utilize standard, tested, authentication services whenever possible
☐Use a centralized implementation for all authentication controls, including libraries that call external authentication services
☐Segregate authentication logic from the resource being requested and use redirection to and from the centralized authentication control
☐All authentication controls should fail securely
☐All administrative and account management functions must be at least as secure as the primary authentication mechanism

☐If your application manages a credential store, use cryptographically strong one-way salted hashes
☐Password hashing must be implemented on a trusted system server side not client side)
☐Validate the authentication data only on completion of all data input
☐Authentication failure responses should not indicate which part of the authentication data was incorrect
☐Utilize authentication for connections to external systems that involve sensitive information or functions
☐Authentication credentials for accessing services external to the application should be stored in a secure store
☐Use only HTTP POST requests to transmit authentication credentials
☐Only send non-temporary passwords over an encrypted connection or as encrypted data
☐Enforce password complexity requirements established by policy or regulation
☐Enforce password length requirements established by policy or regulation
☐Password entry should be obscured on the user's screen
☐Enforce account disabling after an established number of invalid login attempts
☐Password reset and changing operations require the same level of controls as account creation and authentication
☐Password reset questions should support sufficiently random answers
☐If using email based resets, only send email to a pre-registered address with a temporary link/password
☐Temporary passwords and links should have a short expiration time
☐Enforce the changing of temporary passwords on the next use
☐Notify users when a password reset occurs
☐Prevent password re-use
☐Passwords should be at least one day old before they can be changed, to prevent attacks on password re-use
☐Enforce password changes based on requirements established in policy or regulation, with the time between resets administratively controlled
☐Disable "remember me" functionality for password fields
☐The last use (successful or unsuccessful) of a user account should be reported to the user at their next successful login
☐Implement monitoring to identify attacks against multiple user accounts, utilizing the same password
☐Change all vendor-supplied default passwords and user IDs or disable the associated accounts
☐Re-authenticate users prior to performing critical operations
☐Use Multi-Factor Authentication for highly sensitive or high value transactional accounts
☐If using third party code for authentication, inspect the code carefully to ensure it is not affected by any malicious code

Session management

☐Use the server or framework’s session management controls. The application should recognize only these session identifiers as valid
☐Session identifier creation must always be done on a trusted system (server side not client side)
☐Session management controls should use well vetted algorithms that ensure sufficiently random session identifiers

☐Set the domain and path for cookies containing authenticated session identifiers to an appropriately restricted value for the site
☐Logout functionality should fully terminate the associated session or connection
☐Logout functionality should be available from all pages protected by authorization
☐Establish a session inactivity timeout that is as short as possible, based on balancing risk and business functional requirements
☐Disallow persistent logins and enforce periodic session terminations, even when the session is active
☐If a session was established before login, close that session and establish a new session after a successful login
☐Generate a new session identifier on any re-authentication
☐Do not allow concurrent logins with the same user ID
☐Do not expose session identifiers in URLs, error messages or logs
☐Implement appropriate access controls to protect server side session data from unauthorized access from other users of the server
☐Generate a new session identifier and deactivate the old one periodically
☐Generate a new session identifier if the connection security changes from HTTP to HTTPS, as can occur during authentication
☐Consistently utilize HTTPS rather than switching between HTTP to HTTPS
☐Supplement standard session management for sensitive server-side operations, like account management, by utilizing per-session strong random tokens or parameters
☐Supplement standard session management for highly sensitive or critical operations by utilizing per-request, as opposed to per-session, strong random tokens or parameters
☐Set the "secure" attribute for cookies transmitted over an TLS connection
☐Set cookies with the HttpOnly attribute, unless you specifically require client-side scripts within your application to read or set a cookie's value

Access control

☐Use only trusted system objects, e.g. server side session objects, for making access authorization decisions
☐Use a single site-wide component to check access authorization. This includes libraries that call external authorization services
☐Access controls should fail securely
☐Deny all access if the application cannot access its security configuration information
☐Enforce authorization controls on every request, including those made by server side scripts
☐Segregate privileged logic from other application code
☐Restrict access to files or other resources, including those outside the application's direct control, to only authorized users
☐Restrict access to protected URLs to only authorized users
☐Restrict access to protected functions to only authorized users
☐Restrict direct object references to only authorized users
☐Restrict access to services to only authorized users
☐Restrict access to application data to only authorized users
☐Restrict access to user and data attributes and policy information used by access controls
☐Restrict access security-relevant configuration information to only authorized users
☐Server side implementation and presentation layer representations of access control rules must match
☐If state data must be stored on the client, use encryption and integrity checking on the server side to detect state tampering

☐Enforce application logic flows to comply with business rules
☐Limit the number of transactions a single user or device can perform in a given period of time, low enough to deter automated attacks but above the actual business requirement
☐Use the "referer" header as a supplemental check only, it should never be the sole authorization check as it is can be spoofed
☐If long authenticated sessions are allowed, periodically re-validate a user's authorization to ensure that their privileges have not changed and if they have, log the user out and force them to re-authenticate
☐Implement account auditing and enforce the disabling of unused accounts
☐The application must support disabling of accounts and terminating sessions when authorization ceases
☐Service accounts or accounts supporting connections to or from external systems should have the least privilege possible
☐Create an Access Control Policy to document an application's business rules, data types and access authorization criteria and/or processes so that access can be properly provisioned and controlled. This includes identifying access requirements for both the data and system resources

Cryptographic practices

☐All cryptographic functions used to protect secrets from the application user must be implemented on a trusted system
☐Protect secrets from unauthorized access
☐Cryptographic modules should fail securely
☐All random numbers, random file names, random GUIDs, and random strings should be generated using the cryptographic module's approved random number generator
☐Cryptographic modules used by the application should be compliant to FIPS 140-2 or an equivalent standard
☐Establish and utilize a policy and process for how cryptographic keys will be managed

Error handling and logging

☐Do not disclose sensitive information in error responses, including system details, session identifiers or account information
☐Use error handlers that do not display debugging or stack trace information
☐Implement generic error messages and use custom error pages
☐The application should handle application errors and not rely on the server configuration
☐Properly free allocated memory when error conditions occur
☐Error handling logic associated with security controls should deny access by default
☐All logging controls should be implemented on a trusted system
☐Logging controls should support both success and failure of specified security events
☐Ensure logs contain important log event data
☐Ensure log entries that include un-trusted data will not execute as code in the intended log viewing interface or software
☐Restrict access to logs to only authorized individuals
☐Utilize a central routine for all logging operations
☐Do not store sensitive information in logs, including unnecessary system details, session identifiers or passwords
☐Ensure that a mechanism exists to conduct log analysis
☐Log all input validation failures

☐Log all authentication attempts, especially failures
☐Log all access control failures
☐Log all apparent tampering events, including unexpected changes to state data
☐Log attempts to connect with invalid or expired session tokens
☐Log all system exceptions
☐Log all administrative functions, including changes to the security configuration settings
☐Log all backend TLS connection failures
☐Log cryptographic module failures
☐Use a cryptographic hash function to validate log entry integrity

Data protection

☐Implement least privilege, restrict users to only the functionality, data and system information that is required to perform their tasks
☐Protect all cached or temporary copies of sensitive data stored on the server from unauthorized access and purge those temporary working files a soon as they are no longer required.
☐Encrypt highly sensitive stored information, such as authentication verification data, even if on the server side
☐Protect server-side source-code from being downloaded by a user
☐Do not store passwords, connection strings or other sensitive information in clear text or in any non-cryptographically secure manner on the client side
☐Remove comments in user accessible production code that may reveal backend system or other sensitive information
☐Remove unnecessary application and system documentation as this can reveal useful information to attackers
☐Do not include sensitive information in HTTP GET request parameters
☐Disable auto complete features on forms expected to contain sensitive information, including authentication
☐Disable client side caching on pages containing sensitive information
☐The application should support the removal of sensitive data when that data is no longer required
☐Implement appropriate access controls for sensitive data stored on the server. This includes cached data, temporary files and data that should be accessible only by specific system users

Communication security

☐Implement encryption for the transmission of all sensitive information. This should include TLS for protecting the connection and may be supplemented by discrete encryption of sensitive files or non-HTTP based connections
☐TLS certificates should be valid and have the correct domain name, not be expired, and be installed with intermediate certificates when required
☐Failed TLS connections should not fall back to an insecure connection
☐Utilize TLS connections for all content requiring authenticated access and for all other sensitive information
☐Utilize TLS for connections to external systems that involve sensitive information or functions
☐Utilize a single standard TLS implementation that is configured appropriately
☐Specify character encodings for all connections

☐Filter parameters containing sensitive information from the HTTP referer, when linking to external sites

System configuration

☐Ensure servers, frameworks and system components are running the latest approved version
☐Ensure servers, frameworks and system components have all patches issued for the version in use
☐Turn off directory listings
☐Restrict the web server, process and service accounts to the least privileges possible
☐When exceptions occur, fail securely
☐Remove all unnecessary functionality and files
☐Remove test code or any functionality not intended for production, prior to deployment
☐Prevent disclosure of your directory structure in the robots.txt file by placing directories not intended for public indexing into an isolated parent directory
☐Define which HTTP methods, Get or Post, the application will support and whether it will be handled differently in different pages in the application
☐Disable unnecessary HTTP methods
☐If the web server handles different versions of HTTP ensure that they are configured in a similar manner and ensure any differences are understood
☐Remove unnecessary information from HTTP response headers related to the OS, web-server version and application frameworks
☐The security configuration store for the application should be able to be output in human readable form to support auditing
☐Implement an asset management system and register system components and software in it
☐Isolate development environments from the production network and provide access only to authorized development and test groups
☐Implement a software change control system to manage and record changes to the code both in development and production

Database security

☐Use strongly typed parameterized queries
☐Utilize input validation and output encoding and be sure to address meta characters. If these fail, do not run the database command
☐Ensure that variables are strongly typed
☐The application should use the lowest possible level of privilege when accessing the database
☐Use secure credentials for database access
☐Connection strings should not be hard coded within the application. Connection strings should be stored in a separate configuration file on a trusted system and they should be encrypted.
☐Use stored procedures to abstract data access and allow for the removal of permissions to the base tables in the database
☐Close the connection as soon as possible
☐Remove or change all default database administrative passwords
☐Turn off all unnecessary database functionality
☐Remove unnecessary default vendor content (for example sample schemas)

☐Disable any default accounts that are not required to support business requirements
☐The application should connect to the database with different credentials for every trust distinction (for example user, read-only user, guest, administrators)

File management

☐Do not pass user supplied data directly to any dynamic include function
☐Require authentication before allowing a file to be uploaded
☐Limit the type of files that can be uploaded to only those types that are needed for business purposes
☐Validate uploaded files are the expected type by checking file headers rather than by file extension
☐Do not save files in the same web context as the application
☐Prevent or restrict the uploading of any file that may be interpreted by the web server.
☐Turn off execution privileges on file upload directories
☐Implement safe uploading in UNIX by mounting the targeted file directory as a logical drive using the associated path or the chrooted environment
☐When referencing existing files, use an allow-list of allowed file names and types
☐Do not pass user supplied data into a dynamic redirect
☐Do not pass directory or file paths, use index values mapped to pre-defined list of paths
☐Never send the absolute file path to the client
☐Ensure application files and resources are read-only
☐Scan user uploaded files for viruses and malware

Memory management

☐Utilize input and output controls for untrusted data
☐Check that the buffer is as large as specified
☐When using functions that accept a number of bytes ensure that NULL terminatation is handled correctly
☐Check buffer boundaries if calling the function in a loop and protect against overflow
☐Truncate all input strings to a reasonable length before passing them to other functions
☐Specifically close resources, don't rely on garbage collection
☐Use non-executable stacks when available
☐Avoid the use of known vulnerable functions
☐Properly free allocated memory upon the completion of functions and at all exit points
☐Overwrite any sensitive information stored in allocated memory at all exit points from the function

General coding practices

☐Use tested and approved managed code rather than creating new unmanaged code for common tasks
☐Utilize task specific built-in APIs to conduct operating system tasks. Do not allow the application to issue commands directly to the Operating System, especially through the use of application initiated command shells
☐Use checksums or hashes to verify the integrity of interpreted code, libraries, executables, and configuration files
☐Utilize locking to prevent multiple simultaneous requests or use a synchronization mechanism to prevent race conditions

☐Protect shared variables and resources from inappropriate concurrent access
☐Explicitly initialize all your variables and other data stores, either during declaration or just before the first usage
☐In cases where the application must run with elevated privileges, raise privileges as late as possible, and drop them as soon as possible
☐Avoid calculation errors by understanding your programming language's underlying representation
☐Do not pass user supplied data to any dynamic execution function
☐Restrict users from generating new code or altering existing code
☐Review all secondary applications, third party code and libraries to determine business necessity and validate safe functionality
☐Implement safe updating using encrypted channels