IRIS BUSINESS SERVICES LIMITED



Secure Coding Checklist

Secure coding checklist				
Document Name	**Secure coding checklist**			
Document Number	**1**			
Current Version	Ver 1.0			
Classification	**Internal**			
Secure coding checklist				
Document Details				
Owner	PKX Thomas			
Created by	Akshay Rane			
Reviewed by	Mahesh Tilave			
Approved by	PKX Thomas			
Distribution				
Revision History				
Version	**Changes**	**Date**	**Reviewed by**	**Approved by**
1	Version Control Implementation	04-10-2023	Mahesh Tilave	PKX Thomas

Table of Contents:-

Authentication and credentials management: <<	4
Authorization and access control: <<	5
Session management: <<	6
Cryptography: <<	6
Input validation: <<	7
Output encoding: <<	7
Logging and error handling: <<	8
Data protection: <<	9
Miscellaneous: <<	10
File management: <<	11
Third-party libraries: <<	11

Authentication and credentials management: <<

1. Authentication credentials must use TLS and not HTTP clear text.
2. Authentication must be enforced on all pages, except the ones intended to be public.
3. The error messages (in the login page, reset password page, and registration page) should not lead to information-gathering disclosure (for example, in the case of an invalid username).
4. Authentication logic must be validated on the server side.
5. Authentication passwords must be saved under secure hashing algorithms (mot MD5 or SHA1), and salting is preferable.
6. The password's hashing logic must be on the server side.
7. Validate the authentication data after the completion of all the data entry by the end user.
8. If the application is interacting with third-party web services, you will need to ensure the authentication as well as these endpoints.
9. The authentication credentials to interact with third-party web services should be encrypted and not be in clear text (check the config file; developers will leave it there).
10. Enforce password complexity/length requirements established by policy or regulation.
11. Use the following common best practices:
    - Minimum length of 10 characters
    - Minimum of one capital letter
    - Minimum of one special character
    - Minimum of one number
12. Ensure that all password fields do not echo the user's password when it is entered, and that the password fields have autocomplete disabled.
13. Password reset questions should support sufficiently random answers (for example, What is your favorite color is a bad question because Red is a very common answer).
14. If using email-based resets, only send email to a preregistered address with a temporary random link/password (short expiration time).
15. The temporary passwords must be changed for the next usage.
16. Alert users by email or SMS when a user changes or resets their password.
17. Enforce account disabling after a number of login failures (five attempts is a commonly used limit). The account must be disabled for a period of time sufficient to discourage the brute-force guessing of credentials, but not so long as to allow for a denial-of-service attack to be performed.
18. Re-authenticate users prior to performing critical operations.
19. Use multi-factor authentication for highly sensitive or high-value transactional accounts.
20. Disable remember me functionality for password fields.

Authorization and access control: <<

1. Authorization must be developed on the server side.
2. Deny all access if the application cannot access its security configuration information (for example, if the application cannot connect to the database).
3. Authorization must exist on every web request (for example, the Web API endpoint).
4. Access to files (for example, source code, configuration files) and resources (including protected URLs and web services) must be restricted to admins; only they should be allowed to access those resources.
5. If authorization data must be stored on the client side, then you must encrypt it.
6. Use the Referrer header as an additional check, but be careful not to depend on it because it can be spoofed.
7. OS/application service accounts should have the least privilege.
8. Authorize only HTTP methods: GET, POST, PUT, and DELETE.
9. Make sure that you apply authorization changes right away after submitting them to the server by forcing the user to log out from the application.

Session management: <<

1. Sessions must be managed on the server side.
2. Session identifier (session ID) must be random (hackers should not be able to predict it).
3. Logout functionality should totally terminate your session and should be available on all the authenticated pages.
4. Establish a session timeout after inactivity. To calculate the timeout period properly, you need to calculate the security risk of that resource.
5. Do not put session IDs in URLs, logs, and error messages (the session ID is located in the cookie header).
6. Set the secure attribute for cookies.
7. Set the HttpOnly attribute for cookies.

Cryptography: <<

1. Any cryptographic functionality to protect data should be implemented on the server side.
2. Critical data (for example, database-connection strings, passwords, keys, and so on) must be encrypted and should not be in clear text.
3. Cryptographic keys must be protected from unauthorized users (only super admins should have access to them).
4. All generated random items—such as numbers, file names, and strings—must use highly cryptographic random generators.
5. All cryptographic algorithms must use the latest and greatest secure algorithms.

Input validation: <<

1. All data validation must be performed on the server side.
2. Encode data before validation.
3. All validation failures should be rejected in a custom error message.
4. The validation should happen on anything that is processed in the backend, including hidden form values, URLs, and header contents (it should not be limited to form inputs).
5. Hazardous characters, such as <>” ‘ % () & + \ /, should be validated. You should also validate the following:
   - Null bytes (%00)
   - New line (\r,\n,%0d,%0a)
   - dot dot slash (../ or ..\)
6. Confirm that no hard coded SQL queries exist in the source code.
7. Truncate all input strings to a reasonable length before passing them to the copy and concatenation functions.

Output encoding: <<

1. Conduct all the output encoding logic on the server side.
2. Sanitize all the output of untrusted data for SQL, XML, LDAP, and operating system commands.

Logging and error handling: <<

1. Do not disclose sensitive information in error messages, including debugging information, such as a stack trace.
2. Use custom error messages and error pages.
3. Logging controls must be executed on the server side.
4. Logging events must be raised on both success and failure actions.
5. Log data must be clear enough to be able to understand what happened.
6. Log data must be sanitized if it's dependent on an input.
7. Log functions must be centralized and managed in the same class or module.
8. Make sure that you log the following events:
   - Validation failures
   - Authentication attempts
   - Authorization failures
   - Tampering events (for example, URL manipulation for SQL injection)
   - Using invalid or expired sessions
   - All the administrative functions
   - Cryptographic module failures
   - Access from certain countries
   - High frequency of web requests
9. When exceptions occur, you need to be able to exit that function securely.
10. Error or monitoring logs should not be saved on the same server to avoid DOS attacks (by filling the disk drive with random data).

Data protection: <<

1. Temporary sensitive data (for example, caches, or transferred files) must be stored in a secure location, and those items must be purged as soon as possible.
2. Remove comments in the source code that may reveal critical information about the application.
3. Make sure that you protect files on the web server, and that only the intended files are called by clients. Protect config files, backup files, deployment scripts (or any script), documentation that is not intended for the public, temporary files, and any file that contains confidential information.
4. Sensitive information should not be used in the URL query string.
5. Disable caching for pages that handle confidential information. Use Cache-Control:no-store and Pragma:no-cache for this.
6. Data in transit must be encrypted with the latest and greatest TLS algorithms.
7. Carefully use the HTTP referrer when dealing with external domains.

Miscellaneous: <<

1. Make sure that you remove test codes (not intended for production) before deployment.
2. Avoid disclosing your unwanted directory structure in the robots.txt file. Instead, create a parent directory and put all the hidden directories and files within it rather than disallowing each directory/file in robots.txt.
3. Remove any unnecessary information from the HTTP header (for example, the OS version, web server version, and programming frameworks).
4. If, for any reason, the application must elevate its privileges, make sure that you drop them as soon as possible.
5. When designing a REST web API, you have so many options for error codes other than 200 for success and 404 for errors. Proper error codes may help to validate the incoming requests properly. Here are some best practices to consider for each REST API status return code:
   - 200 OK: The action is successful.
   - 202 Accepted: The request to create a resource is accepted.
   - 204 No Content: The POST request did not include a client-generated ID.
   - 400 Bad Request: The request is malformed.
   - 401 Unauthorized: Wrong authentication ID or credentials.
   - 403 Forbidden: An authenticated user does not have the permission to access the resource.
   - 404 Not Found: Requesting a non-existant resource.
   - 405 Method Not Allowed: Unexpected HTTP method in the request.
   - 429 Too Many Requests: This error may occur when a DOS attack is detected.
6. Make sure that the following headers exist:
   - X-frame-options
   - X-content-type-options
   - Strict-transport-security
   - Content-security-policy
   - X-permitted-cross-domain-policies
   - X-XSS-protection:1;mode=block
   - X-content-type-options:nosniff
7. Properly free allocated memory upon the completion of functions and at all exit points.

File management: <<

1. The user must be authenticated before uploading any files into the application.
2. Limit the type of files that can be uploaded into the application.
3. Validate uploaded files by checking the file headers. Checking the extension by itself is not sufficient.
4. Uploaded files should be saved on a separate server rather than the web server.
5. Carefully check and validate (or remove if necessary) the uploaded files that will be executed and interpreted by the web server.
6. Execution privileges must be turned off on the file upload server.
7. Antiviruses and endpoint security must exist on the upload file server.
8. Do not pass directory or file paths; instead use index values mapped to a predefined list of paths. Never send the full absolute path in the response to the client.
9. The web application files and resources must be in read-only format.

Third-party libraries: <<

1. Use checksums to verify the integrity of files (such as libraries and scripts) downloaded from the internet.
2. Ensure that the library that is downloaded and used in the application is the latest stable version.
3. Use a third-party libraries scanner (for example, Sonatype, Blackduck).