Classification: Public

National Data Center

Guidelines document

Secure Coding Guidelines

Bangladesh Computer Council

Information and Communication Technology Division

Document Change Control

Version	Date	Reason for issue	Issued By	Reviewed By
1.0	01/08/2015	Confirming secure coding practice for NDC's environment	BCC Management	NDC Team

Document Approval

Name	Role	Date	Signature
Mr. Tarique M Barkatullah	Director, National Data Center	31/08/2016	Approved

Distribution List

Name	Role
Not Applicable	Public

Document Control

Classification	**Public**
Document Location	
Maintenance Owner	National Data Center Team
Approval owner	Director, National Data Center
Release Date	31/08/2016
Next Review Date	28/09/2017

Table of Contents

1. Introduction 4
2. Objective 4
3. Scope 4
4. Roles and Responsibilities 5
   - 4.1 Roles and Responsibilities for Customers: 5
   - 4.2 Roles and responsibilities of BCC: 6
5. Identification of security related requirements 7
   - 5.1 Assess the Risk 7
   - 5.2 Security Requirements 7
   - 5.3 System Design 7
   - 5.4 Security Testing 8
6. General Coding Practices 9
7. Secure Coding 10
   - 7.1 Input validation 10
   - 7.2 Output encoding 11
   - 7.3 Access Control 11
   - 7.4 Authentication & Password Management 12
   - 7.5 Session Management 14
   - 7.6 Encryption 15
   - 7.7 Error Handling & Logging 15
   - 7.8 Data Protection 16
   - 7.9 Communication Security 16
   - 7.10 System Configuration 17
   - 7.11 Database Security Configuration 18
   - 7.12 File Management 18
   - 7.13 Memory Management 19
   - 7.14 HTTP Security 19
   - 7.15 Business logic 20
8. References 21

1. Introduction

Security must be incorporated in the design of applications and in the application development process. It is a necessary consideration at the time of development of applications. Secure coding practices must take into consideration the Industry best practices such as OWASP, NIST, ISO etc. to guide the personnel involved in application development and to ensure security is embedded in the design and development of the applications as per best practices.

2. Objective

The objective of this document is to:

- Provide guidelines for secure coding practices.
- Provide best practices for development of secure applications such as OWASP, NIST, ISO etc.
- Encourage the adoption of enterprise processes and tools to reduce or eliminate vulnerabilities before deployment.

This guideline is to be used by the application developers to write secure code, make code changes and follow security principles for applications.

3. Scope

This guideline is applicable to

- All the applications, API, web services or scripts hostedin and used from/toNational Data Center at Bangladesh Computer Council (BCC).
- Commercial-off-the-shelf (COTS) applications hosted in the National Data center.
- All customers who have developed and hosted their applications in the National Data Center.
- The applications (if any) developed within the National Data Center.

4. Roles and Responsibilities

Roles and responsibilities required are as follows:

4.1 Roles and Responsibilities for Customers:

Customers are the organizations/departments which hosts applications in National Data Centre at BCC. The roles are described as below. The roles mentioned are indicative and equivalent roles in the customer organization shall apply as applicable.

Role	Responsibilities
Department Owner	Responsible for overseeing the functioning of a particular function/department and also ensures that applications meets requirements mentioned in this guideline.
Information Owner	Accountable for and has authority to control the management and use of confidential information of applications hosted in National Data Center.
Information Security Team	Responsible for enforcing, implementing and executing portions of these guidelines to ensure that security has been incorporated into the development of applications.
Customer Manager	• Ensure security requirements as mentioned in this guidelines are integrated into all phases of the system and software development process and managing the effects of changes or differences in configurations on an information system, network or application. • Responsible to adhere to this guideline into all phases of application development and patching. • Inform BCC for any major change in applications hosted in National Data Center.
Test Engineer	Verify security controls are functioning properly for the developed system, prior to its hosting in National Data Center.
Software Engineer	Utilize secure coding guidelines and implement security requirements during system development.
System Engineer	Utilize secure configuration during system development.

4.2 Roles and responsibilities of BCC:

During the course of application hosting BCC roles and responsibilitiesare as follows**:**

Role	Responsibilities
Division Owner	Responsible for overseeing the functioning of a particular function/division and applications hosted in that division.
Information Custodian	In consultation with Information Owner, accountable for management and use of information of applications hosted in National Data Center.
Information Security Team	Responsible for assessing and managing risks from hosted application in National Data Center.
System Owner	Accountable for and has authority to control the management and use of NDC information systems.
CIRT	Responsible for testing and highlighting security flaws in the applications hosted and to be hosted in the National Data Center.
Test Engineer	Tests and verify proper functioning of security controls for the developed application, prior to its hosting in National Data Center.
System Engineer	Utilize secure configuration during system development.

5. Identification of security related requirements

Security must be incorporated from planning phase of system development and acquisition to ensure that threats, requirements and potential constraints in functionality & Integration are considered. Following should be adhered to:

5.1 **Assess the Risk**:Customer should assess the risk of the application in initial development phase. Following points should be considered to assess the risk associated with the application:

a) System specific threats should be identified and classified according to their impact on the system.
b) Sensitive system functions should be identified.
c) If third party component is involved, associated risks should be evaluated.
d) Compensatory controls should be identified where risks cannot be mitigated through the identified security controls.

5.2 **Security Requirements**:Security requirements for the system and department functionalities should be identified based on the risks. Some of the recommended security related requirements are as follows:

a) Authentication, authorization and accounting (AAA) – Based upon the information stored/ processed by the system, requirements of Authentication, Authorization and Accounting should be identified in consultation with the Information Owner. The following should be identified:
   - Authentication method (one factor or multi-factor) and Authentication type (Remote access or direct access).
   - Authorization requirement should adhere to users' roles.
   - Audit logging should be enabled to establish accountability.
b) Transmission Protocol – Transmission protocols to be used by the system should be identified in consultation with the Information Owner. This will include, but not limited to:
   - Requirements of TCP protocols.
   - Application level protocols such as HTTP and HTTPS.
c) Requirements for encryption, secure configuration and system monitoring and reporting should be identified.
d) Information to be stored on the system should be classified.
e) Availability requirements of system should be identified in consultation with department owner. This includes, but not limited to:
   - System uptime requirements (Recovery Time objective and Recovery Point Objective).
   - System capacity requirements.
   - Disaster recovery requirements.

5.3 **System Design**:Output of security risk assessment and security requirements should be documented and used to design and develop the applications. The system Access control matrix should be prepared for resources utilizing the system. If system is being acquired, due diligence should be performed.

a) All key stakeholders should have a common understanding of security implications, considerations, and requirements. Key security milestones and time frames should be discussed.

b) Plans for development phase security training, quality assurance techniques, deliverables, and milestones should be discussed.

c) Development team should carry out the review of the initial system design based on the following aspects:
   - Whether the design mitigates the risks identified in the risk assessment.
   - Topology, data flow, access control, host interfaces, network and device configuration, contingency planning, system validation, VPN/remote access.
   - Whether compensatory controls have been considered wherever required.

d) Results of the preliminary security review should be documented and shared with the stakeholders.

e) Security-related requirements identified in the Planning phase should be shared with the development team to be considered in the construction phase.

f) Development team should incorporate compensatory controls where identified controls cannot be applied due to technical, commercial or legal limitations.

5.4 **Security Testing**:Before the application is deployed in production, it should be thoroughly tested. Security testing of the application must include:

a) Secure Code Review
b) Black-box and Grey-box application security assessment
c) Vulnerability assessment and penetration testing of supporting infrastructure

Third party security testing reports should be shared with BCC prior to hosting the application in National Data Center (NDC). Severity for identified issues should be categorized and remediated according to the risk. Applications with critical vulnerabilities and significant risk shall not be deployed in the National Data Center.BCC shall assess the security testing report or may perform additional test in order to approve the application to be hosted in NDC.

6. General Coding Practices

a) Tested and approved code developed securely should be used rather than creating new code for common tasks.
b) Task specific built-in APIs should be utilized to conduct operating system tasks. Application should not issue commands directly to the Operating System, especially through the command shells initiated by the application.
c) Checksums or hashes to verify the integrity of interpreted code, libraries, executables, and configuration files should be used.
d) To prevent multiple simultaneous requests or use a synchronization mechanism to prevent race conditions locking should be utilized.
e) Shared variables and resources should be protected from inappropriate concurrent access.
f) All variables and other data stores should be explicitly initialized, either during declaration or just before the first usage.
g) In cases where the application must run with elevated privileges, privileges should be raised as late as possible, and drop them as soon as possible.
h) Calculation errors should be avoided by understanding the programming language's underlying representation and how it interacts with numeric calculation. Byte size discrepancies, precision, signed/unsigned distinctions, truncation, conversion and casting between types, "not-a-number" calculations, and how the language handles numbers that are too large or too small for its underlying representation should be closely monitored.
i) User supplied data should not be passed to any dynamic execution function.
j) Users should be restricted from generating new code or altering existing code.
k) All secondary applications, third party code and libraries to determine the necessity and validate safe functionality should be reviewed, as these can introduce new vulnerabilities.
l) Safe updating should be implemented e.g., if the application will utilize automatic updates, then cryptographic signatures for the code should be used and it should be ensured that download client verifies those signatures. Encrypted channels to transfer the code from the host server should be used.
m) Latest development framework should be used to develop the applications.
n) The existing framework should be updated to latest version. Compensatory control should be applied where this cannot be achieved due to operational issues.

7. Secure Coding

Secure coding should be imparted to minimize the risks of vulnerabilities during development. A training plan should be developed to train developers on secure coding practices and how to apply these practices. Trainings should be conducted periodically for the development team to communicate the importance of secure coding practices. Secure coding guidelines should be made available to the developers.

Systems during development should have following secure coding practices as recommended by OWASP (Open web application security project):

7.1 **Input validation**:

A malicious input provided by the user may compromise the application or underlying systems.Input validation is used to verify all properties of the data entered into an application are as expected. All inputs to the system/software should be checked for the characters/phrases that may compromise the system. Input validation should happen at the server side and the client side. Following should be followed for input validation:

a) All data should be validated on a trusted system (e.g., the server) before processing.
b) All data sources should be identified and classified into trusted and untrusted.
c) All data from untrusted sources (e.g., Databases, file streams, etc.) should be validated.
d) There should be a centralized input validation routine for the application.
e) Proper character sets, such as UTF-8, should be specified for all sources of input.
f) Data should be encoded to a common character set before validating.
g) All validation failures should result in input rejection.
h) Determine if the system supports UTF-8 extended character sets and if so, validate after UTF-8 decoding is completed.
i) All clients provided data should be validated before processing, including all parameters, URLs and HTTP header content (e.g. Cookie names and values). Include automated post backs from JavaScript, Flash or other embedded code.
j) Verify that header values in both requests and responses contain only ASCII characters.
k) Data from redirects, data range, data length and expected data types should be validated.
l) All input should be validated against a list of allowed characters, whenever possible.
m) If any potentially hazardous characters are allowed as input, additional controls like output encoding, secure task specific APIs and accounting for the utilization of that data should be implemented throughout the application. Examples of common hazardous characters include: <> " ' % ( ) & + \ \' \"*#;--.
n) If the validation routine cannot address the following inputs, then they should be checked discretely for:
   - Null bytes (%00).
   - New line characters (%0d, %0a, \r, \n).
   - "Dot-dot-slash" (../ or ..\) path alterations characters. In cases where UTF-8 extended character set encoding is supported, address alternate representation like: %c0%ae%c0%ae/ (Utilize canonicalization to address double encoding or other forms of obfuscation attacks).
   - All non-printable characters from (in hexadecimal values) 0H to 19H.
o) Application should check and prevent against HTTP Parameter Pollution attacks, if application makes no distinction about source of request.

7.2 Output encoding: A properly validated output to user prevents user from being compromised to session hijacking and defaced website. These controls verify that output data is clearly distinguished and cannot be mistaken as executable code such as for Cross Site Scripting. All output should be encoded according to guidelines mentioned as below:

a) All encoding should be conducted on a trusted system (e.g., the server).
b) A standard, tested routine for each type of outbound encoding should be implemented.
c) All data returned to the client that originated outside the application's trust boundary should be contextually output encoded. (HTML entity encoding is one example, but does not work in all cases).
d) All characters should be encoded unless they are known to be safe for the intended interpreter.
e) All outputs of un-trusted data to queries for SQL, XML, and LDAP should be contextually sanitized.
f) All output of un-trusted data to operating system commands should be sanitized.
g) All untrusted data that are output to HTML (including HTML elements, JavaScript data values, CSS blocks and URI attributes) should be properly escaped for the applicable context.

7.3 Access Control: Access controls such as RSA tokens, usernames and passwords prevent unauthorized access to application. It should be implemented to authenticate and authorize users for system access in accordance with their roles and responsibilities. Requirement of single factor or multifactor authentication should be assessed to provide access to users. Following guidelines should be incorporated while implementing access controls:

a) Only trusted system objects, e.g. server side session objects, should be used for making access authorization decisions.
b) A single site-wide component should be used to check access authorization. This includes libraries that call external authorization services.
c) Access controls should fail securely.
d) All access should be denied if the application cannot access its security configuration information.
e) Authorization controls should be enforced on every request, including those made by server side scripts, "includes" and requests from rich client-side technologies like AJAX and Flash.
f) Privileged logic should be segregated from other application code.
g) Access to files or other resources, including those outside the application's direct control, should be restricted to only authorize users.
h) Access to protected URLs, protected functions, direct object references, services, and application data should be restricted only to authorized users.
i) Access to user and data attributes, and policy information should be restricted.
j) Access to security-relevant configuration information should be restricted only to authorized users.
k) Server side implementation and presentation layer representations of access control rules should match.
l) If state data must be stored on the client, encryption and integrity checking should be used on the server side to catch state tampering.
m) Application logic flows should be enforced to comply with rules.

n) Number of transactions a single user or device can perform in a given period of time should be limited. The transactions time should be above the actual requirement, but low enough to deter automated attacks.
o) The "referrer" header should be used as a supplemental check only, it should never be the sole authorization check, as it is can be spoofed.
p) If long authenticated sessions are allowed, re-validate a user's authorization periodically to ensure that their privileges have not changed and if they have, log the user out and force them to re-authenticate.
q) Account auditing should be implemented and enforce the disabling of unused accounts (e.g., After no more than 30 days from the expiration of an account's password).
r) The application should support disabling of accounts and terminating sessions when authorization ceases (e.g., Changes to role, employment status, etc.)
s) Service accounts or accounts supporting connections to or from external systems should have the least privilege possible.
t) Create an Access Control Policy to document an application's 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.
u) Directory browsing should be disabled unless desired.
v) Access controls should be enforced on server.
w) All access attempts to the application including failed attempt should be logged. Logs should provide sufficient information (timestamp, source and destination information etc.) to regenerate the scenario.

7.4 Authentication & Password Management: A strong password will prevent the application from being compromised by an attacker.Password policies should be implemented and maintained for the system. The system policy should be designed to enforce strong passwords. Following guidelines should be adhered to:

a) Authentication for all pages and resources should be required, except those specifically intended to be public.
b) All authentication controls should be enforced on a trusted system (e.g., the server).
c) Standard, tested, authentication services should be established and utilized whenever possible.
d) A centralized implementation for all authentication controls should be used, including libraries that call external authentication services.
e) There should be segregation between the authentication logic and the resource being requested, and use redirection to and from the centralized authentication control.
f) All authentication controls should fail securely.
g) All administrative and account management functions should be at least as secure as the primary authentication mechanism.
h) If application manages a credential store, it should ensure that only cryptographically strong one-way salted hashes of passwords are stored and that the table/file that stores the passwords and keys is write-able only by the application. (Do not use the MD5 algorithm because it could be broken easily).
i) Password hashing should be implemented on a trusted system (e.g., the server).
j) The authentication data should be validated only on completion of all data input, especially for sequential authentication implementations.

k) Authentication failure responses should not indicate which part of the authentication data was incorrect. For example, instead of "Invalid username" or "Invalid password", just use "Invalid username and/or password" for both. Error responses should be truly identical in both display and source code.
l) Authentication for connections to external systems should be utilized that involve sensitive information or functions.
m) Authentication credentials for accessing services external to the application should be encrypted and stored in a protected location on a trusted system (e.g., the server). The source code is NOT a secure location.
n) Only HTTP POST requests should be used to transmit authentication credentials.
o) Non-temporary passwords should only be transmitted over an encrypted connection or as encrypted data, such as in an encrypted email.
p) Password complexity requirements should be enforced, established by policy or regulation. Authentication credentials should be sufficient to withstand attacks that are typical of the threats in the deployed environment. (e.g., requiring the use of alphabetic as well as numeric and/or special characters.)
q) Password length requirements should be enforced, established by policy or regulation.
r) Password entry should be obscured on the user's screen. (e.g., on web forms use the input type "password".)
s) Account disabling should be enforced after an established number of invalid login attempts (e.g., five attempts are common). The account should be disabled for a period of time sufficient to discourage brute force guessing of credentials, but not so long as to allow for a denial-of-service attack to be performed.
t) Password reset and changing operations should require the same level of controls as account creation and authentication.
u) Password reset questions should support sufficiently random answers. Questions like pet’s name, mother’s maiden name etc. should be avoided.
v) If using email based resets, only email to a pre-registered address with a temporary link/password should be sent.
w) Temporary passwords and links should have a short expiration time.
x) Changing temporary passwords should be enforced on the next use.
y) Users should be notified when a password reset occurs.
z) Password re-use should be prevented.
aa) Passwords should be at least one day old before they can be changed, to prevent attacks on password re-use.
bb) Enforce password changes based on requirements established in policy or regulation. Critical systems may require more frequent changes. The time between resets must be administratively controlled.
cc) The "remember me" functionality should be disabled for password fields.
dd) The last use (successful or unsuccessful) of a user account should be reported to the user at their next successful login.
ee) Monitoring should be done to identify attacks against multiple user accounts, utilizing the same password. This attack pattern is used to bypass standard lockouts, when user IDs can be harvested or guessed.
ff) All vendor-supplied default passwords and user IDs or disable the associated accounts should be changed.
gg) Users should be re-authenticated prior to performing critical operations.

hh) Multi-Factor Authentication for highly sensitive or high value transactional accounts should be used.
ii) If using third party code for authentication, inspect the code carefully to ensure it is not affected by any malicious code.
jj) While using HTML, for password field, add attribute “Autocomplete” and set its value to “off” to prevent password cached in web-browsers.

7.5 Session Management: Session data when compromised by an attacker could be used to mimic user’s session; henceUser sessions should be handled securely. Session data should not be transmitted in clear-text. Number of concurrent sessions for each system account should be defined by account type. Sessions should be locked after inactivity or upon receiving an invalid input from a user. Following session management practices should be adhered:

a) Session identifier creation should always be done on a trusted system. (e.g., the server)
b) Session management controls should use well vetted algorithms that ensure sufficiently random session identifiers.
c) The domain and path for cookies containing authenticated session identifiers should be used to an appropriately restricted value for the site.
d) Logout functionality should fully terminate the associated session or connection and available from all pages protected by authorization.
e) Session inactivity timeout should be established that is as short as possible, based on balancing risk and functional requirements.
f) Persistent logins should not be allowed and enforce periodic session terminations, even when the session is active. Especially for applications supporting rich network connections or connecting to critical systems. Termination times should support access requirements and the user should receive sufficient notification to mitigate negative impacts.
g) If a session was established before login, that session should be closed and new session should be after a successful login.
h) A new session identifier should be generated on any re-authentication.
i) Concurrent logins with the same user ID should not be allowed.
j) Session identifiers should not be exposed in URLs, error messages or logs. Session identifiers should only be located in the HTTP cookie header. For example, do not pass session identifiers as GET parameters.
k) Server side session data should be protected from unauthorized access, by other users of the server, by implementing appropriate access controls on the server.
l) A new session identifier should be generated and deactivate the old one periodically. (This can mitigate certain session hijacking scenarios where the original identifier was compromised.)
m) A new session identifier should be generated if the connection security changes from HTTP to HTTPS, as can occur during authentication. Within an application, it is recommended to consistently utilize HTTPS rather than switching between HTTP to HTTPS.
n) For sensitive server side operations, such as account management, strong random tokens or parameters should be used per-session. These can help in preventing CSRF attacks.
o) For highly sensitive or critical applications, strong random tokens or parameters should be used per-request.
p) For cookies transmitted over TLS connection, the “secure” attribute should be set.
q) The cookies should be set only with HttpOnly attribute, unless there is a specific requirement for client side scripts within the application to read or set cookie value.

r) All pages that require authenticated access should have logout links.
s) Session tokens generated should be sufficiently long and random.
t) The "domain" cookie attribute restriction should not be set unless required (such as during single sign-on)
u) Session tokens sent in cookies should be protected by "HTTPOnly" cookie attribute.
v) Authenticated session tokens in cookies should be protected with "secure" attribute.

7.6 Encryption: Protecting sensitive information during transmission and storage is necessary to prevent misuse or unwanted modification in information. Following guidelines should be adhered for encryption practices:

a) All cryptographic functions used to protect secrets from the application user should be implemented on a trusted system. (e.g., the server)
b) Master secrets should be protected from unauthorized access.
c) Cryptographic modules should fail securely.
d) All random numbers, random file names, random GUIDs, and random strings should be generated using the cryptographic module's approved random number generator when these random values are intended to be un-guessable.
e) Cryptographic modules used by the application should be compliant to FIPS 140-2 (http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf) or an equivalent standard.
f) The policy and process for cryptographic key management should be established.

7.7 Error Handling & Logging: Application should handle errors appropriately and errors should not disclose any sensitive information about system such as installed version, system uptime, number of connected users and internal IP addresses. This information could be used by an attacker to plan further and sophisticated attacks. Web application should be robust and fault-tolerant regarding intended operation; it should invalidate a session, if an obvious or assumed abuse attempt is present. Information disclosure can further be prevented by adhering to following:

a) Sensitive information should not be disclosed in error responses, including system details, session identifiers or account information.
b) Error handlers that do not display debugging or stack trace information should be used.
c) Generic error messages and use custom error pages should be used and implemented.
d) The application should handle application errors and not rely on the server configuration.
e) Allocated memory should be securely released when error conditions occur.
f) Error handling logic associated with security controls should deny access by default.
g) All logging controls should be implemented on a trusted system. (e.g., the server)
h) Logging controls should support both success and failure of specified security events.
i) Logs should contain important log event data.
j) Log entries that include un-trusted data should not execute as code in the intended log viewing interface or software.
k) Access to logs should be restricted only to authorized individuals.
l) A master routine for all logging operations should be utilized.
m) Sensitive information, including unnecessary system details, session identifiers or passwords should not be stored in logs.
n) A mechanism should exist to conduct log analysis.
o) All input validation failures, authentication attempts (including failures) and access control failures should be logged.

p) All apparent tampering events, including unexpected changes to state data should be logged.
q) Attempts to connect with invalid or expired session tokens should be logged.
r) All system exceptions, administrative functions, including changes to the security configuration settings, backend TLS connection failures, and cryptographic module failures should be logged.
s) A cryptographic hash function should be used to validate log entry integrity.
t) All non-printable characters should be encoded in log entries.
u) All events including untrusted data should not execute as code in log viewing software.

7.8 **Data Protection:** Data should be protected from unauthorized access or modification and available to authorized users when required. Sensitive information should be encrypted during storage. Additionally, following guidelines should be adhered:

a) Least privilege should be implemented, restrict users to only the functionality, data and system information that is required to perform their tasks.
b) List of all sensitive data should be identified and a policy should be established to control the access of the data.
c) All cached or temporary copies of sensitive data stored on the server should be protected from unauthorized access and purge those temporary working files a soon as they are no longer required.
d) Highly sensitive stored information, like authentication verification data, should be encrypted even on the server side. Well vetted algorithms should be used.
e) Server-side source-code should be protected from being downloaded by a user.
f) Passwords, connection strings or other sensitive information should not be stored in clear text or in any non-cryptographically secure manner on the client side. This includes embedding in insecure formats like: MS viewstate, Adobe flash or compiled code.
g) Comments in user accessible production code that may reveal backend system or other sensitive information should be removed.
h) Unnecessary application and system documentation as this can reveal useful information should be removed.
i) Sensitive information should not be included in HTTP GET request parameters.
j) Auto complete features on forms expected to contain sensitive information, including authentication should be disabled.
k) Client side caching on pages containing sensitive information should be disabled. Cache-Control: no-store, may be used in conjunction with the HTTP header control "Pragma: no-cache", which is less effective, but is HTTP/1.0 backward compatible.
l) The application should support the removal of sensitive data when that data is no longer required. (e.g. personal information or certain financial data.)
m) Appropriate access controls for sensitive data stored on the server should be implemented. This includes cached data, temporary files and data that should be accessible only by specific system users.

7.9 **Communication Security:** Information should be protected against unauthorized modification or access during transmission. TLS protocol encrypts the data that is exchanged between browser and server and ensures that confidential information is protected against spying during transmission. Following guidelines should be adhered while developing communication security:

a) Encryption for the transmission of all sensitive information should be implemented. This should include TLS for protecting the connection and may be supplemented by discrete encryption of sensitive files or non-HTTP based connections.
b) TLS certificates should be valid and have the correct domain name, not be expired, and be installed with intermediate certificates when required.
c) Failed TLS connections should not fall back to an insecure connection.
d) Failed TLS connection should be logged.
e) TLS connections for all content requiring authenticated access and for all other sensitive information should be used.
f) TLS for connections to external systems that involve sensitive information or functions should be used.
g) A single standard TLS implementation that is configured appropriately should be utilized.
h) Character encodings for all connections should be specified.
i) Parameters containing sensitive information from the HTTP referrer, when linking to external sites should be filtered.
j) At minimum following operations should be encrypted:
   - Registration
   - Login
   - Access to personal data
   - Change of password
   - "Password reminder" function

7.10 System Configuration: Improperly configured systems could be easily compromised.System should be configured in accordance Minimum Baseline Security Standards for respective systems. Additionally, following guidelines should be adhered:

a) Servers, frameworks and system components should be running the latest approved version.
b) Servers, frameworks and system components should have all patches issued for the version in use.
c) Directory listings should be turned off.
d) The web server, process and service accounts should be restricted to the least privileges possible.
e) Exceptions when occur, should fail securely.
f) All unnecessary functionality, files test code or any functionality not intended for production, should be removed prior to deployment.
g) Disclosure of directory structure in the robots.txt file should be disabled by placing directories not intended for public indexing into an isolated parent directory. Then "Disallow" that entire parent directory in the robots.txt file rather than disallowing each individual directory.
h) Define which HTTP methods the application will support and whether it will be handled differently in different pages in the application.
i) Unnecessary HTTP methods, such as WebDAV extensions should be disabled. If an extended HTTP method that supports file handling is required, utilize a well-vetted authentication mechanism.
j) If the web server handles both HTTP 1.0 and 1.1, both should be configured in a similar manor.
k) Unnecessary information from HTTP response headers related to the OS, web-server version and application frameworks should be removed.

l) The security configuration store for the application should be able to be output in human readable form to support auditing.
m) An asset management system should be implemented and register system components and software in it.
n) Isolate development environments from the production network and access should be provided only to authorized development and test groups.
o) A software change control system should be implemented to manage and record changes to the code both in development and production.

7.11 Database Security Configuration: Insecure databases could result in compromise of the database or underlying system via application flaws.Databases should be hardened in accordance with CIS benchmarks. Application should interact with database in a secure manner and following controls should be implemented:

a) Strongly typed parameterized queries should be used.
b) Input validation and output encoding should be utilized and meta-characters should be addressed. If these fail, do not run the database command.
c) Variables should be strongly typed.
d) The application should use the lowest possible level of privilege when accessing the database.
e) Secure credentials should be used for database access.
f) 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.
g) Stored procedures should be used to abstract data access and allow for the removal of permissions to the base tables in the database.
h) Connection should be closed as soon as possible.
i) All default database administrative passwords should be removed or changed. Utilize strong passwords/phrases or implement multi-factor authentication.
j) All unnecessary database functionality (e.g., unnecessary stored procedures or services, utility packages, install only the minimum set of features and options required (surface area reduction)) should be turned off.
k) Unnecessary default vendor content (e.g., sample schemas) should be removed.
l) Any default accounts that are not required should be disabled.
m) The application should connect to the database with different credentials for every trust distinction (e.g., user, read-only user, guest, administrators).

7.12 File Management: Files stored on server including source code should be protected from unauthorized access or modification to prevent defacing of the application and ensure that proper functionality of application modules. Additionally, unauthorized access to files by system user (for example: by exploiting a vulnerability) should be prevented. Files which are required to be uploaded should be restricted to certain file types only and should be validated irrespective of file extension. Adhere to following guidelines to securely develop file management controls:

a) User supplied data should not be passed directly to any dynamic include function.
b) Authentication should be required before allowing a file to be uploaded.
c) Limit the type of files that can be uploaded to only those types that are needed for application's purpose.

d) Uploaded files should be validated against the expected type by checking file headers. Checking for file type by extension alone is not sufficient.
e) Files should not be saved in the same web context as the application. Files should either go to the content server or in the database.
f) Prevent or restrict the uploading of any file that may be interpreted by the web server.
g) Execution privileges should be turned off on file upload directories.
h) Implement safe uploading in UNIX by mounting the targeted file directory as a logical drive using the associated path or the chrooted environment.
i) When referencing existing files, a white list of allowed file names and types should be used. Validate the value of the parameter being passed and if it does not match one of the expected values, either reject it or use a hard coded default file value for the content instead.
j) Do not pass user supplied data into a dynamic redirect. If this should be allowed, then the redirect should accept only validated, relative path URLs.
k) Directory or file paths should not be passed, use index values mapped to pre-defined list of paths.
l) Absolute file path should never be sent to the client.
m) Application files and resources should be read-only.
n) User uploaded files should be scanned for viruses and malware.

7.13 Memory Management: Insecure system memory might result in compromise of application such as buffer or stack overflow.System memory (primary memory and secondary memory) should be secured by implementing following controls:
a) Input and output control for un-trusted data should be utilized.
b) Double check that the buffer should be as large as specified.
c) When using functions that accept a number of bytes to copy, such as strncpy(), ensure that if the destination buffer size is equal to the source buffer size, it may not NULL-terminate the string.
d) Buffer boundaries should be checked if calling the function in a loop and make sure there is no danger of writing past the allocated space.
e) All input strings should be truncated to a reasonable length before passing them to the copy and concatenation functions.
f) Specifically close resources, don't rely on garbage collection. (e.g., connection objects, file handles, etc.)
g) Non-executable stacks should be used when available.
h) Use of known vulnerable functions (e.g., printf, strcat, strcpy etc.) should be avoided.
i) Allocated memory should be securely freed upon the completion of functions and at all exit points.

7.14 HTTP Security: Application data transmitted over HTTP should be protected to prevent application or underlying system being compromised. Following guidelines should be adhered to:
a) Application should accept only defined set of HTTP request (such as POST or GET). All other methods should be explicitly blocked.
b) HTTP response should contain "content-type" header specifying character set such as "UTF-8".
c) HTTP headers should not contain non-printable ASCII characters.

d) HTTP header or other mechanism should be included prevent clickjacking attack on old browsers.
e) User should not be able to spoof HTTP headers included by framework or at frontend.
f) X-frame option should be used where content is not allowed to be displayed in third party X-frame.
g) HTTP headers should not disclose detailed version information of system components.

7.15 Business logic: Uncontrolled business logic might result in exploitation of application flaw. An attacker may exploit the flaw to compromise underlying system such as inserting long strings in database. To prevent business logic following guidelines should be adhered to:

a) Verify all high level business logic flow in trusted system (e.g. server).
b) Application should not allow spoofed transaction i.e. an attacker perform a transaction on behalf of a user.
c) Application should not allow business logic parameters to be tampered such as modification of price during payment.
d) Application should have defensive measures against repudiation attacks such as transactional logs should be available and monitoring should be done for transactional anomalies.
e) Application should be protected against information disclosure attacks such as tampering, session brute-force etc.
f) Elevation of privileges should be prevented.
g) Business logic flows should be processed in sequential order.
h) Application should have additional authorization for lower valued system and segregation of duties for higher valued systems.

8. References

[1] Security Considerations in the System Development Life Cycle, NIST Special Publication 800-64 Revision 2
[2] OWASP – A Guide to Building Secure Web Applications Version 2.0
[3] OWASP Secure Coding Practices – Quick Reference Guide
[4] OWASP Application Security Verification Standard (2014)
[5] ISO/IEC 27034-6 - Security guidance for specific applications

Note: Future changes to this document shall take into consideration the above references, superseding documents and other Industry best practices.