Web Application Secure Coding Standards

Version 2.0

Reference: 6.100 **-** Information Technology and Security Policy
6.101 **-** Use of County Information Technology Resources

Developed by: Security Engineering Team - Applications Security

Table of Contents

1.0.1 Release Notes and History Log

The content in this document will be periodically updated to reflect changes in two environments: 1) the County's; and 2) vendor hosted. Both environments have the requirement to capture the industry best practices as technology and standards continue to evolve.

DATE	VERSION NUMBER	MODIFIED BY:	DESCRIPTION of CHANGE
5/22/2006	1.0	R. Pittman (CIO)	1) Document initial creation.
2/15/2007	1.0	R. Pittman (CIO)	1) Draft provided to E-Commerce Readiness Committee for review and comment 2) Draft provided to the County's Security Engineering Team (SET) Applications Security for their review and approval
5/4/2008	1.0	R. Pittman (CIO)	1) Numerous revisions made based on comments from E-Commerce Readiness Group Committee
7/13/2010	1.0	S. Foong (DHS)	1) Document Title changed from Application Development Security Handbook 2) Added a new section for Web Application Architecture
7/27/2010	1.0	S. Foong (DHS)	1) Numerous revisions made on the new section Web Application Architectures and Link-to-Gov
8/10/2010	1.0	S. Foong (DHS)	1) Revisions made up to 5.0.3
8/24/2010	1.0	S. Foong (DHS)	1) Additional revisions made on Section 3
9/14/2010	1.0	S. Foong (DHS)	1) Completed revision on Section 3& 4
9/28/2010	1.0	S. Foong (DHS)	1) Started on Section 5 – revisions made up to 5.03
10/12/2010	1.0	S. Foong (DHS)	1) Reviewed up to 5.03; revisions made up to 5.04
10/26/2010	1.0	S. Foong (DHS)	1) Reviewed and revised 5.04 (Error) – 5.05 (Database)
11/09/2010	1.0	S. Foong (DHS)	1) Completed Section 5
12/14/2010	1.0	S. Foong (DHS)	1) Completed Section 6
01/11/2011	1.0	S. Foong (DHS)	1) Completed the entire document
03/15/2011	1.0	SET - Policies	Reviewed document
04/26/2011	1.0	S. Foong (DHS)	Revise document per SET Policies review
05/10/2011	1.0	S. Foong (DHS)	Revised coding checklist
05/24/2011	1.0	S. Foong (DHS)	Revised coding checklist
06/14/2011	1.0	S. Foong (DHS)	Revised coding checklist
07/12/2011	1.0	S. Foong (DHS)	Completed the coding checklist
07/26/2011	1.0	S. Foong (DHS)	Completed the review of the entire document
08/02/2011	1.0	S. Foong (DHS)	Revised document per SET Policies Team review
01/24/2012	2.0	S. Foong (DHS)	Added mobile application security controls
02/28/2012	2.0	S. Foong (DHS)	Additional mobile application security controls added
01/22/2013	2.0	S. Foong (DPH)	Completed mobile web application secure coding requirements
02/12/2013	2.0	App Sec SET Team	Document Review
06/25/2013	2.0	App Sec SET Team	Final Draft

1.0 Purpose

This technology agnostic document defines the required secure coding standards. These standards implement the security requirements that are within the County Application Security Development Lifecycle Standards. Writing secure code reduces the risks of introducing software defects responsible for causing vulnerabilities and improves the quality of the software.

For E-Commerce applications, the County has a master agreement with a third-party vendor (vendor), which provides a hosted environment. They are compliant with the Payment Card Industry (PCI) data security standard (See intranet site at: http://web.co.la.ca.us/lacounty/ecommerce/cms1_029102.pdf).

2.0 Common Vulnerabilities In Web Applications

The term "vulnerability" implies a weakness or a flaw in design, development, or configuration that if exploited, can adversely affect an organization's security posture. Knowledge of software vulnerabilities can help developers to identify and manage risks in existing code.

The following is the OWASP Top 10 Application Security Risk list. The Top 10 provides basic techniques to protect against these high risks problem areas.
https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

1: Injection

2: Broken Authentication and Session Management

3: Cross-Site Scripting (XSS)

4: Insecure Direct Object References

5: Security Misconfiguration

6: Sensitive Data Exposure

7: Missing Function Level Access Control

8: Cross-Site Request Forgery (CSRF)

9: Using Known Vulnerable Components

10: Unvalidated Redirects and Forwards

3.0 Application Secure Coding Requirements and Checklist

The best place to defend against web application attacks is within the source code itself.

Regardless of the programming language or techniques used for development, the following checklist must be implemented by the developer at the time the application is being developed.

This checklist must also be used to verify software security after an update in the application to accept additional functionality or to store/access data of greater sensitivity.

Completed checklist(s) with deviations from the Standards are to be documented, reviewed and approved by the appropriate Manager and/or the Departmental Information Security Officer (DISO).

The following defines the settings used in the checklist:

Mandatory (M) - All Mandatory settings must be applied.

Recommended (R) - All Recommended settings should be applied unless the business operation is severely impacted.

Sensitive/Confidential Applications – Applications that store, process, or communicate sensitive/confidential information (e.g., electronic protected health information (ePHI), personally identifiable information (PII) or other classified data defined by the business owner).

General Applications – Applications that do not store, process, or communicate sensitive/confidential information.

	Secure Coding Requirements Checklist	General Applications	Sensitive /Confidential Applications
3.0	**GENERAL CODING PRACTICES**		
3.0.1	Use tested and approved managed code for common tasks rather than creating new code.	M	M
3.0.2	Utilize task specific built-in APIs (Application Programming Interface) 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.	M	M
3.0.3	Use checksums or hashes to verify the integrity of interpreted code, libraries, executables, and configuration files.	R	R
3.0.4	Utilize locking to prevent multiple simultaneous updates and use a queuing mechanism to prevent race condition.	M	M
3.0.5	Avoid using global shared variables whenever possible.	M	M
3.0.6	Explicitly initialize all your variables and other data stores, either during declaration or just before the first usage.	M	M
3.0.7	In cases where the application must run with elevated privileges, raise privileges as late as possible, and drop them as soon as possible.	M	M
3.0.8	Do not pass user supplied data to any dynamic execution function.	M	M
3.0.9	Review all secondary applications, third party code and libraries (e.g., anti-cross site scripting library) to determine business necessity and validate safe functionality, as these can introduce new vulnerabilities.	R	R
3.0.10	Test the application code with a web application scanner to detect vulnerabilities.	M	M
3.0.11	All queries must be parameterized to prevent SQL injection.	M	M
3.1	**INPUT VALIDATION**		
3.1.1	Data is checked on both the client and server side.	M	M
3.1.2	Conduct all data validation on a trusted system (e.g., The server)	M	M
3.1.3	Validate all incoming data regardless of the source (e.g., Databases, file streams, etc.)	M	M
3.1.4	There should be a centralized routine for server-side input validation for the application.	M	M
3.1.5	Specify proper character sets, such as UTF-8, for all sources of input.	M	M
3.1.6	Encode data to a common character set before validating (canonicalize). All URL and HTML encoded characters are interpreted (%3c becomes <). All Unicode or alternate encoded characters are placed in their expected character representation.	M	M
3.1.7	If the system supports UTF-8 extended character set, validate after UTF-8 decoding is completed.	M	M
3.1.8	All validation failures should result in input rejection.	M	M
3.1.9	Validate all client provided data before processing, including all parameters, URLs and HTTP header content (e.g., Cookie names and value). Be sure to include automated post backs from JavaScript, flash or other embedded code.	M	M

	Secure Coding Requirements Checklist	General Applications	Sensitive /Confidential Applications
3.1.10	Verify that header values in both requests and responses contain only ASCII characters.	R	M
3.1.11	Validate data from redirects (an attacker may submit malicious content directly to the target of the redirect, thus circumventing application logic and any validation performed before the redirect)	M	M
3.1.12	Data is strongly typed. Expected input is matched to a data type such as: varchar, integer, string, Boolean, or a custom type.	M	M
3.1.13	Validate data range.	M	M
3.1.14	Validate data length. For example, if usernames can be no longer than 12 characters, then a 15-character username is not accepted. This also applies to columns in which the data is stored in the database.	M	M
3.1.15	Validation filters are applied to the entire input string. In regular expressions, this means the caret (^) and dollar sign ($) are placed at the beginning and end of the regular expression.	M	M
3.1.16	Validate all input against a "white" list of allowed characters, whenever possible.	M	M
3.1.17	Data is checked for valid content. The value of a parameter is checked for correctness. For example, a U.S. state abbreviation is a string, but can only be one of 51 possible combinations.	M	M
3.1.18	If any potentially hazardous characters must be allowed as input, additional controls must be implemented, like output encoding, secure task specific APIs and accounting for the utilization of that data throughout the application. Examples of common hazardous characters include: < > " ' % ( ) & + \ \' \"	M	M
3.2	**SESSION MANAGEMENT**		
3.2.1	Use the server or framework's session management controls. The application should only recognize these session identifiers as valid.	R	R
3.2.2	When the session ID must be created, it must be created securely (e.g., on a trusted system such as the server). Session ID are derived from a sufficiently pseudo-random pool to prevent spoofing attacks. Use well vetted algorithms that ensure sufficiently random session identifiers. Use high entropy (unpredictable) long session identifiers.	M	M
3.2.3	Disallow persistent logins (e.g., "remember me" feature using cookies)	R	M
3.2.4	Enforce idle session termination. Termination time should support business requirements and the user should receive sufficient notification to mitigate negative impacts.	M	M
3.2.5	A user may have only one active session at any given time. Any pre-existing session for the user must be terminated after successful login.	M	M
3.2.6	The application must track and prevent concurrent logins. This can stop session hijacking and session replay attacks.	M	M
3.2.7	Generate a new session identifier on any re-authentication.	M	M
3.2.8	Generate a new session identifier and delete the old one periodically during an active session.	R	M

	Secure Coding Requirements Checklist	General Applications	Sensitive /Confidential Applications
3.2.9	Generate a new session identifier if a user's privileges or role changes.	M	M
3.2.10	Generate a new session identifier if the connection security changes from HTTP to HTTPS as can occur during authentication.	M	M
3.2.11	Within an application, it is recommended to consistently utilize HTTPS rather than switching between HTTP to HTTPS.	R	M
3.2.12	Do not expose session identifiers in URLs, error messages or logs. Session identifiers should only be located in the HTTP cookie header (e.g., do not pass session identifiers as GET parameters).	M	M
3.2.13	Only utilize the system generated session identifiers for client side session management. Avoid using parameters or other client data for state management.	M	M
3.2.14	Supplement standard session management for highly sensitive or critical operations by utilizing per-request, as opposed to per-session, strong random identifier or parameters.		M
3.2.15	Authorization permissions are tied to the session object, not tracked by separate identifier. This can prevent privilege escalation attacks.	M	M
3.2.16	Set the domain and path for cookies containing authenticated session identifiers to an appropriately restricted value for the site. This restrictions allows browsers to only send cookie information back to the server for the given domain and path.	R	R
3.2.17	Set cookies with the HttpOnly attribute, unless you specifically require client-side scripts within your application to read or set a cookie's value.	M	M
3.2.18	Logout functionality must be available from all pages.	M	M
3.2.19	Logout functionality should fully terminate the associated session or connection.	M	M
3.2.1	Enforce the changing of temporary passwords on the next use.	M	M
3.3	**ACCESS CONTROL**		
3.3.1	Use only trusted system objects, e.g., server side session objects, for making access authorization decisions.	M	M
3.3.2	Use a single site-wide component to check access authorization. This includes libraries that call external authorization services.	M	M
3.3.3	Access controls should fail securely such that if they are not working properly, access to the resources are denied by default.	M	M
3.3.4	Enforce authorization controls on every request, including those made by server side scripts, "includes" and requests from rich client-side technologies like AJAX and Flash.	R	M
3.3.5	Segregate privileged access control code from other application code.	M	M
3.3.6	Restrict access to files or other resources, including those outside the application's direct control, to only authorized users.	M	M
3.3.7	Restrict access to protected URLs to only authorized users.	M	M
3.3.8	Restrict access to protected functions to only authorized users.	M	M
3.3.9	Restrict direct object references to only authorized users.	M	M

	Secure Coding Requirements Checklist	General Applications	Sensitive /Confidential Applications
3.3.10	Restrict access to services to only authorized users.	M	M
3.3.11	Restrict access to application data to only authorized users.	M	M
3.3.12	Restrict access to user and data attributes and policy information used by access controls.	M	M
3.3.13	Server side implementation and presentation layer representations of access control rules must match.	M	M
3.3.14	If state data must be stored on the client, use encryption and integrity checking on the server side to catch state tampering.	M	M
3.3.15	Limit the number of transactions a single user or device can perform in a given period of time. The transactions/time should be above the actual business requirement, but low enough to deter automated attacks.	M	M
3.3.16	Use the "referer" header as a supplemental check only. It should never be the sole authorization check as it can be spoofed.	M	M
3.3.17	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.	R	M
3.3.18	Implement account auditing and enforce the disabling of unused accounts (e.g., 30 days from the expiration of an account's password)	M	M
3.3.19	The application must support disabling of accounts and terminating sessions when authorization ceases (e.g., employment status)	M	M
3.3.20	Service accounts or accounts supporting connections to or from external systems should have the least privilege possible.	M	M
3.4	**DATABASE SECURITY**		
3.4.1	The application must use the lowest possible level (least) of privilege when accessing the database.	M	M
3.4.2	The application with sensitive data must communicate with the database over an encrypted channel.	R	M
3.4.3	Use secure credentials that meet the County's password complexity requirements for database access.	M	M
3.4.4	Remove or change all default database administrative passwords. Utilize strong passwords/phrases or implement multi-factor authentication.	M	M
3.4.5	Make use of the Data Access Layer (DAL) to interface with the database, (e.g., stored procedures, parameterized queries, etc.) to prevent direct database access.	M	M
3.4.6	Remove unnecessary default vendor content (e.g., sample schemas)	M	M
3.4.7	Disable any default accounts that are not required to support business requirements.	M	M
3.4.8	Utilize input validation and output encoding and be sure to address meta characters. If these fail, do not run the database command.	M	M
3.4.9	Ensure that variables are strongly typed.	M	M
3.4.10	Use strongly typed parameterized queries.	M	M
3.4.11	If the application uses functional error handling for database connections (e.g.,	M	M

	Secure Coding Requirements Checklist	General Applications	Sensitive /Confidential Applications
	TRY CATCH block), its use must be comprehensive and thorough.		
3.4.12	Do not hard code connection credentials. They must be stored in an encrypted manner. Access to files containing connection credential must be restricted. Whenever possible, the file should be stored outside the web document root.	M	M
3.4.13	Do not hard code connection strings within the application. Connection strings should be encrypted and stored in a separate configuration file on a trusted system.	M	M
3.4.14	The application must connect to the database with different credentials for every trust distinction (e.g., user, read-only user, guest, administrators).	R	M
3.4.15	Use stored procedures to abstract data access and allow for the removal of permissions to the tables in the database.	M	M
3.4.16	Use parameterized SQL statements. Do not use string concatenation or string replacement to build SQL statements.	M	M
3.4.17	Terminate the connection as soon as possible.	M	M
3.5	**AUTHENTICATION and PASSWORD MANAGEMENT**		
3.5.2	Temporary passwords and verification links should have a short expiration time (e.g., 1 day)	M	M
3.5.3	All passwords must ~~be unique and~~ meet the County password standards.	M	M
3.5.4	Provide feedback on the strength of passwords when it is being created for the first time.	R	R
3.5.5	Authentication uses a challenge-response mechanism to reduce (but not block) the effectiveness of sniffing attacks.	R	R
3.5.6	Username and/or Password should not be based on full or partial Social Security Number (SSN). Current applications using SSN as a form of identification should only use the last four digits.	M	M
3.5.7	For external public facing applications, the initial user enrollment process must require the input of a minimum of two pieces of private information to be used for user identification and password reset.	M	M
3.5.8	When obtaining answers to the security questions, a message should be posted warning the user not to use publicly known information (e.g., pet's name on a social network page).	R	R
3.5.9	After authentication, the application should track the hashed value based on the session instead of the user's password.	R	M
3.5.10	Use only POST requests to transmit authentication credentials.	M	M
3.5.11	Segregate authentication login and use redirection after login.	R	M
3.5.12	All authentication controls should fail securely (e.g., session is terminated)	M	M
3.5.13	Sensitive user actions must require a two-factor authentication process.	R	M
3.5.14	Send password over an encrypted connection.	M	M
3.5.15	Password entry must be obscured on the user's screen. (e.g., on web forms use the input type "Password")	M	M

	Secure Coding Requirements Checklist	General Applications	Sensitive /Confidential Applications
3.5.16	Passwords must be stored using non-reversible encryption.	M	M
3.5.17	If a user forgets their password, then a password reset function will generate a new temporary password delivered to the user in a secure manner.	M	M
3.5.18	An administrative function should be provided to improve monitoring of password reset activity.	R	R
3.5.19	Use a salted version of a cryptographically strong one-way hash algorithm for password credential hashes, such as SHA-256. Do not use the MD5 algorithm if it can be avoided.	R	M
3.5.20	Enforce account lockout policy (e.g., disable account after an established number of invalid login attempts within a period of time).	M	M
3.5.21	The application should employ a secure protocol (e.g., https, TSL) for transmission of sensitive data such as user authentication credentials.	M	M
3.5.22	Error conditions 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.	M	M
3.5.23	If using third party code for authentication, inspect the code carefully to ensure it is not affected by any malicious code.	M	M
3.5.24	Use context to add security to authentication (e.g., device ID, IP location).	R	M
3.6	**OUTPUT ENCODING**		
3.6.1	Implement all output encoding on the server side.	M	M
3.6.2	Utilize a standard, tested routine for each type of outbound encoding.	M	M
3.6.3	Verify that all output data are properly sanitized and escaped/encoded (including HTML elements, attributes, JavaScript data values, CSS Blocks, and URI attributes, SQL. XML, LDAP, and operating systems commands) for the application context.	M	M
3.7	**ERROR HANDLING and LOGGING**		
3.7.1	Do not disclose sensitive information in error responses, including system details, session identifiers or account information.	M	M
3.7.2	Do not store sensitive information in logs, including unnecessary system details, session identifiers or passwords.	M	M
3.7.3	Use error handlers that do not display debugging or stack trace information.	M	M
3.7.4	Implement generic error messages and use custom error pages	M	M
3.7.5	The application should handle application errors and not rely on the server configuration	M	M
3.7.6	Properly free allocated memory when error conditions occur.	M	M
3.7.7	Error handling logic associated with security controls should deny access by default.	M	M
3.7.8	All logging controls should be implemented on a trusted system (e.g., The server).	M	M
3.7.9	Any logs that are generated must be sent to a centralized location for storage. This central log server must employ the strictest level of security, restricting	M	M

	Secure Coding Requirements Checklist	General Applications	Sensitive /Confidential Applications
	access to logs to only authorized individuals.		
3.7.10	The application must write to a custom error log which will provide useful information for debugging the application as well as identifying malicious activity.	M	M
3.7.11	Centralized logs from 3.7.8 must be sent to the central log server over an encrypted channel.	M	M
3.7.12	Logging controls should support both success and failure of specified security events.	M	M
3.7.13	Ensure log entries that include un-trusted data will not execute as code in the intended log viewing interface or software.	M	M
3.7.14	Ensure logs contain important log event data:		
	a. Log all input validation failures.	R	M
	b. Log all authentication attempts.	M	M
	c. Log all access control failures.	M	M
	d. Log all apparent tampering events, including unexpected changes to state data	M	M
	e. Log attempts to connect with invalid or expired session ID.	M	M
	f. Log all system exceptions.	M	M
	g. Log all administrative functions, including changes to the security configuration settings.	M	M
	h. Log all backend TLS (Transport Layer Security) connection failures.	M	M
	i. Log cryptographic module failures	M	M
3.7.15	Use a cryptographic hash function to validate log entry integrity	R	M
3.8	**APPLICATION AUDIT EVENTS**		
3.8.1	The user ID and source IP address is recorded for authentication success and failure.	M	M
3.8.2	The user ID and source IP address is recorded for each modification of sensitive profile information (i.e., SSN, home address, etc.).	M	M
3.8.3	The user ID and source IP address is recorded for each access to financial information.	M	M
3.8.4	Log for malicious input (i.e., for SQL injection, input validation, and buffer overflow attacks).	M	M
3.8.5	Log attempts to connect with invalid or expired session identifiers.	M	M
3.8.6	The information recorded for each event should be enough to identify a user and the associated activities.	M	M
3.9	**FILE MANAGEMENT**		
3.9.1	Do not pass user supplied data directly to any dynamic include function.	M	M
3.9.2	Do not pass user supplied data into a dynamic redirect. If this must be allowed, then the redirect should accept only validated relative path URLs.	M	M
3.9.3	Do not pass directory or file paths. Use index values mapped to pre-defined list of paths.	M	M

	Secure Coding Requirements Checklist	General Applications	Sensitive /Confidential Applications
3.9.4	File references should remove all directory traversal characters. Never send the absolute file path to the client.	M	M
3.9.5	Limit the type of files that can be uploaded to only those types that are needed for business purposes to prevent malicious embedded code.	M	M
3.9.6	Require verification such as authentication or CAPTCHA before allowing a file to be uploaded.	M	M
3.9.7	Validate that the uploaded files are the expected type by checking file headers. Checking for file type by extension alone is not sufficient.	M	M
3.9.8	Implement file size upload restriction to prevent rogue uploads from filling up the drive space.	M	M
3.9.9	Do not save files in the same web context as the application. Files should either go to the content server or in the database.	M	M
3.9.10	Prevent or restrict the uploading of any file that may be interpreted and executed by the web server.	M	M
3.9.11	Turn off execution privileges on file upload directories. Ensure application files and resources are read-only.	M	M
3.9.12	When referencing existing files, use a "whitelist" of allowed file names and types. Validate the value of the parameter being passed and reject it if it does not match one of the expected values.	M	M
3.9.13	Files are only retrieved from a specific directory which does not contain application code.	M	M
3.10	**MEMORY MANAGEMENT (If Applicable)**		
3.10.1	Ensure that the buffer size is as large as specified.	M	M
3.10.2	Check buffer boundaries to prevent writing beyond the allocated buffer size.	M	M
3.10.3	Check buffer boundaries if calling the function in a loop and make sure there is no danger of writing past the allocated buffer size.	M	M
3.10.4	Truncate all input strings to a reasonable length before passing them to the copy and concatenation functions.	M	M
3.10.5	Specifically close resources, don't rely on garbage collection. (e.g., connection objects, file handles, etc.)	M	M
3.10.6	Use non-executable stacks when available.	M	M
3.10.7	Avoid the use of known vulnerable functions (e.g., printf, strcat, strcpy, etc.)	M	M
3.10.8	Properly free allocated memory upon the completion of functions and at all exit points.	M	M
3.11	**DATA PROTECTION**		
3.11.1	Implement least privilege; restrict users to only the functionality, data and system information that is required to perform their tasks.–	M	M
3.11.2	Protect all cached or temporary copies of sensitive data stored on the server from unauthorized access and purge those temporary working files as soon as they are no longer required.	R	M
3.11.3	Encrypt highly sensitive stored information, such as authentication verification data, even on the server side.	M	M

	Secure Coding Requirements Checklist	General Applications	Sensitive /Confidential Applications
3.11.4	Encryption and MAC (message authentication code) must be applied to cookies or fields with sensitive data.	M	M
3.11.5	Protect server-side source code from being downloaded by a user.	M	M
3.11.6	Remove comments in user accessible production code that may reveal backend system or other sensitive information.	M	M
3.11.7	Remove unnecessary application and system documentation as this can reveal useful information to attackers.	M	M
3.11.8	Do not include sensitive information in HTTP GET request parameters.	M	M
3.11.9	Disable auto complete features on forms expected to contain sensitive information, including authentication.	M	M
3.11.10	Disable client side caching on pages containing sensitive information.	M	M
3.11.11	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 users.	M	M
3.12	**COMMUNICATION SECURITY (For System Administrators)**		
3.12.1	TLS certificates should be valid and have the correct domain name, not be expired, and be installed with intermediate certificates when required.	R	M
3.12.2	Failed TLS connections should not fall back to an insecure connection.	M	M
3.12.3	Utilize TLS connections for all content requiring authenticated access and for all other sensitive information.	M	M
3.12.4	Utilize TLS for connections to external systems that involve sensitive information or functions.	M	M
3.12.5	Utilize a single standard TLS implementation that is configured appropriately.	M	M
3.12.6	Specify character encodings for all connections.	R	M
3.12.7	Filter parameters containing sensitive information from the HTTP referrer, when linking to external sites.	M	M
3.13	**WEB SERVER CONFIGURATION (For System Administrators)**		
3.13.1	The security configuration for the application should be able to be output in human readable form to support auditing.	M	M
3.13.2	Disable unnecessary HTTP methods, such as WebDAV extensions.	M	M
3.13.3	Remove all unnecessary functionality and files.	M	M
3.13.4	Remove test code or any functionality not intended for production, prior to deployment.	M	M
3.13.5	Turn off directory listings.	M	M
3.13.6	Configure server to prevent leaking of information such as version, patch revision, etc. to clients. Ex: on Apache, set ServerTokens Prod and ServerSignature Off in httpd.conf	M	M

4.0 Mobile Application Security Requirements

A mobile application is an application that resides on a mobile device and leverages server-side resources. In addition to the security requirements stated in this document, mobile applications ***must adhere*** to the following controls and design principles to eliminate the most common mobile application vulnerabilities.

	Secure Coding Requirements Checklist	General Applications	Sensitive /Confidential Applications
4.0	**PROTECT SENSITIVE DATA AT REST** Reference Section 3.11		
4.0.1	Sensitive data should be stored on the server instead of the client-side/mobile device. When storing sensitive data on the device is unavoidable, the application specific storage on the device must be encrypted.	R	M
4.0.2	Do not store information locally on the device beyond the period required by the application and business owner. Reference the Board Of Supervisors (BOS) Policy 3.040.	M	M
4.0.3	Do not store temporary/cached data in a world readable directory. Applications should delete temporary/cached data automatically when it is no longer in use to prevent sensitive data from remaining in cache indefinitely.	M	M
4.0.4	Use a randomly-generated number instead of the device ID number as an identifier (e.g., session ID). Apply the same data minimization principles to application sessions as you would to http sessions and cookies.	M	M
4.0.5	Applications on managed devices should leverage remote wipe and kill switch APIs (OS-level or purpose-built) to remove sensitive information from the device in the event of theft or loss.	M	M
4.0.6	SMS, MMS or notifications should not be used to send sensitive data to or from mobile end-points.	R	R
4.2	**HANDLE PASSWORD CREDENTIALS SECURELY** Reference Section 3.1		
4.2.1	Do not embed any passwords in the application. Do not use a generic shared password for integration with the backend (i.e., password embedded in code). Mobile application binaries can be easily downloaded and reverse engineered.	M	M
4.2.2	In case passwords need to be stored on the device, leverage the encryption and key-store mechanisms provided by the mobile OS to securely store passwords, password equivalents and authentication tokens. Passwords must be stored in a salted hash format .	M	M
4.2.3	Ensure passwords and keys are not visible in cache or logs.	M	M
4.2.4	Provide the ability for the mobile user to change/remove passwords on the device.	M	M
4.2.5	Consider using authentication tokens (e.g., County RSA SecureID Tokens) instead of passwords.	R	M
4.3	**IMPLEMENT USER AUTHENTICATION AND SESSION MANAGEMENT** Reference Section 3.2		
4.3.1	Use context to add security to authentication (e.g., device ID, geo-location).	R	M
4.3.2	Implement two factor authentications for applications giving access to sensitive data or interfaces where possible.	R	M
4.3.3	Use authentication that ties back to the end user's identity, rather than the device identity. Users maintain their own credentials and applications transmit the user's credentials to the server for authentication.	M	M

	Secure Coding Requirements Checklist	General Applications	Sensitive /Confidential Applications
4.4	**KEEP THE BACKEND API (Services) AND THE PLATFORM (Server) SECURE**		
4.4.1	Carry out a specific check of your code for sensitive information (i.e., metadata, comments, file names) that may be unintentionally disclosed, such as when information is transferred between the mobile device and web-server back-end and other external interfaces.	M	M
4.4.2	All back-end services (e.g., REST/Web Services) for mobile applications should be tested for vulnerabilities periodically using static code analyzer tools and fuzzing tools for testing and finding security flaws.	M	M
4.4.3	Ensure that the back-end platform (server) is running with a hardened configuration with the latest security patches regularly applied to the OS, web server and other application components.	M	M
4.4.4	Ensure adequate and secured logs are retained on the back-end in order to detect and respond to incidents and perform forensics (e.g., within the limits of data protection law).	M	M
	Ensure that input is entered by a person and not a script by employing techniques such as image validation and CAPTCHA .	M	M
4.4.5	Reduce the risk from DDoS attacks by employing rate limiting and throttling on a per-user/IP basis.	M	M
4.4.6	Test for DoS vulnerabilities where the server may become overwhelmed by certain resource intensive application calls.	M	M
	Plan for post deployment optimization with the deployment team to ensure the connection string timeouts are optimized to strike a balance between allowing legitimate application calls and timing out on possible malicious application calls.	R	M
4.4.7	Perform testing of the backend Web Service, REST or API to determine whether vulnerabilities exist. Perform abuse case testing, in addition to use case testing.	M	M
4.5	**SECURE DATA INTEGRATION WITH THIRD PARTY SERVICES AND APPLICATIONS**		
4.5.1	Test and verify the security/authenticity of any third party code/libraries used in your mobile application prior to incorporation within an application (e.g., reliable source, vendor supported, no backend trojans, licensing terms/agreement).	M	M
4.5.2	Track all third party frameworks/APIs used in the mobile application for security patches (e.g., subscribe to vendor listserv). A corresponding security update/release must be done for the mobile applications using these third party APIs/frameworks.	M	M
4.5.3	All data received from and sent to third party applications must be validated before processing in an application.	M	M
4.6	**IMPLEMENT CONTROLS TO PREVENT UNAUTHORIZED ACCESS TO PAID-FOR RESOURCES (e.g., wallet, SMS, NFC Payments, Phone calls)**		
4.6.1	Maintain logs of access to paid resources in a non-reputable format and make them available to the end-user for review (e.g., signed receipt sent to server back-end). Logs should be protected from unauthorized access and manipulation.	M	M

	Secure Coding Requirements Checklist	General Applications	Sensitive /Confidential Applications
4.6.2	Check for abnormal usage patterns in paid resource usage and require re-authentication (e.g., significant changes in location, user language change).	M	M
4.6.3	Consider using a white-list model by default for paid resource addressing (e.g., address book only unless specifically authorized for phone calls).	M	M
4.6.4	Authenticate all API calls to paid resources (e.g. using an application developer certificate, API keys).	M	M
4.6.5	Ensure that wallet API callbacks do not pass clear text account/pricing/ billing/item information.	M	M
4.6.6	Warn user and obtain consent for any cost implications for application behavior.	M	M
4.7	**ENSURE SECURE DISTRIBUTION/PROVISIONING OF MOBILE APPLICATIONS**		
4.7.1	Applications must be designed and provisioned to allow updates for security patches, taking into account the requirements for approval by app-stores and the extra delay this may incur.	M	M
4.7.2	Distributing applications through official app-stores provides a safety net in case issues or vulnerabilities are discovered in your application. Most app-stores are able to remove malicious applications from their store front at short notice in case of an incident.	M	M
4.7.3	Provide feedback channels for users to report security problems with applications (e.g., a security@ email address).	M	M
4.7.4	Developers should digitally sign their mobile applications to help end-users better distinguish between trusted versus potentially questionable code.	M	M
4.8	**CAREFULLY CHECK ANY RUNTIME INTERPRETATION OF CODE FOR ERRORS**		
4.8.1	Run interpreters at minimal privilege levels.	M	M
4.8.2	Define comprehensive escape syntax (e.g., try-catch routine) as appropriate.	M	M
4.8.3	Fuzz test (i.e., abuse case test) interpreters.	M	M
4.8.4	Sandbox interpreters.	M	M

4.1 Relevant general coding best practices:

The following are some of the most important general coding best practices, **particularly relevant to mobile coding**:

- To prevent SQL Injection, *parameterized queries* must be implemented. https://www.owasp.org/index.php/Query_Parameterization_Cheat_Sheet
- Validate all inputs and outputs.
- Minimize lines and complexity of code where possible.
- Use safe, high-level development languages to avoid issues such as buffer-overflow.
- Use safe string functions to avoid buffer and integer overflow.
- Implement a security report handling point (e.g., security@example.com).
- Use static code analyzers and dynamic application scanners to find security flaws.
- Run applications with the minimum privilege required for the application on the operating system. Be aware of privileges granted by default by APIs and disable them.
- Never authorize code/application to execute with root/system administrator privilege.
- Remove all test code before releasing the application.
- Remove all comments from text-based code (e.g., HTML, CGI, scripting) prior to releasing the application.
- Perform abuse case testing, in addition to use case testing.
- Always perform testing as a standard user as well as a privileged user.
- Avoid opening application-specific listener ports on the client device. Use the communication mechanisms provided by the OS.
- Ensure logging is done appropriately but do not record excessive logs, especially those including sensitive user information.

4.2 Enterprise-specific Guidelines:

- If a business-sensitive application needs to be provisioned on a device, applications should enforce a higher security posture on the device, such as PIN, remote management/wipe, application monitoring.

- Device certificates can be used for stronger device authentication.

- Ensure that application testing properly simulates the enterprise's diverse user base and site locations.

5.0 References

a) Countywide Information Security Website
http://ciointranet.lacounty.gov/Pages/Security.aspx/

The County's Information Security intranet website has a wealth of information regarding how to code secure applications as well as using a development methodology from Microsoft called the Security Development Lifecycle (SDL). SDL is similar in scope with the Systems Development Lifecycle (SDLC); however, it incorporates security protections at the beginning of the application lifecycle.

b) OWASP Secure Coding Practices – Quick Reference Guide
http://www.owasp.org/index.php/OWASP_Secure_Coding_Practices_-_Quick_Reference_Guide

The guide is a technology agnostic set of general software security coding practices, in a comprehensive checklist format, that can be integrated into the development lifecycle. The focus is on secure coding requirements, rather than on vulnerabilities and exploits.

c) OWASP Mobile Security Project
https://www.owasp.org/index.php/OWASP_Mobile_Security_Project

The OWASP Mobile Security Project is a centralized resource intended to give developers and security teams the resources they need to build and maintain secure mobile applications. The goal is to classify mobile security risks and provide developmental controls to reduce their impact or likelihood of exploitation.

d) OWASP Web Application Security Testing Cheat Sheet
https://www.owasp.org/index.php/Web_Application_Security_Testing_Cheat_Sheet

This cheat sheet provides a checklist of tasks to be performed when performing a blackbox security test of a web application.

e) The World Wide Web Security FAQ
www.w3.org/Security/Faq

The World Wide Web Consortium (W3C) hosts this document as a service to the Web Community; however, it does not endorse its contents. However, it has a wealth of valuable information that includes FAQs for application security and general security information as well as client-side, server-side, CGI-Scripts, protecting confidential documents, and denial-of-service attacks.

f) CERT Software Engineering Institute
http://www.cert.org/advisories/CA-1997-25.html

The CERT Coordination Center has received reports and seen mailing list discussions of a problem with some CGI scripts, which allow an attacker to execute arbitrary commands on a WWW server under the effective user-id of the server process. The problem lies in how the scripts are written, NOT in the scripting languages themselves.

g) http://www.webopedia.com

This web site is very useful as a reference guide (e.g., encyclopedia) in identifying and gaining definitions of information technology related terms and acronyms. In other words, this site is dedicated to computer technology.

h) http://msdn.microsoft.com/en-us/magazine/gg309184.aspx

MSDN Security briefs – web application configuration security.

i) Hack Notes: Web Security Portable Reference, Mike Shema; 174 pages, 2003, McGraw-Hill Companies. This book is an excellent resource that comprises information on how hackers break into web applications with a tool as fundamental as a web browser, guard against simple to complex web application attacks, strengthen web application security using a detailed methodology for testing and secure coding, as well as eliminate susceptibility to E-commerce, SQL injection, and input validation hacks.

j) Writing Secure Code, Microsoft Second Edition, Michael Howard and David LeBlanc; 768 pages, 2003, Microsoft Press. This book is required reading at Microsoft, as indicated by Bill Gates. It covers practical strategies and techniques for secure application coding in a networked world.