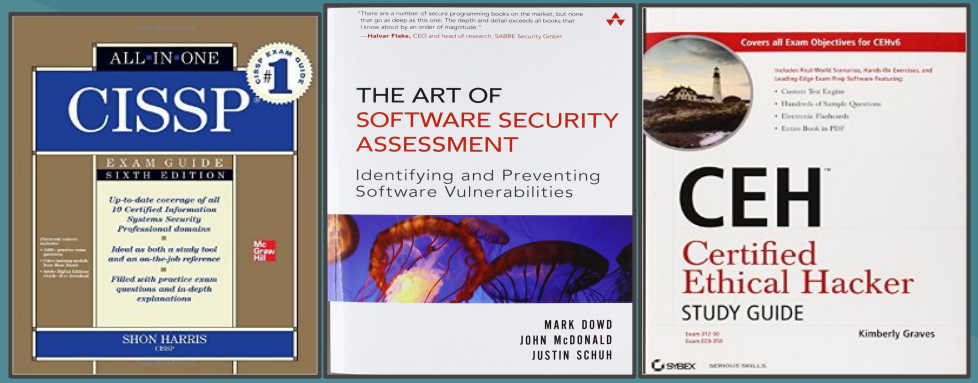
OWASP

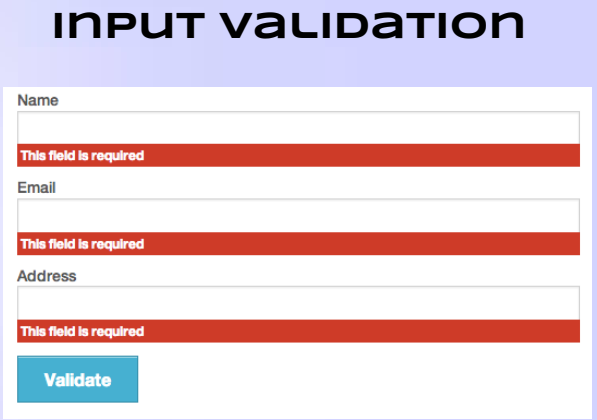
The Open Web Application Security Project



Secure coding practices Checklist

Secure coding practices Checklist

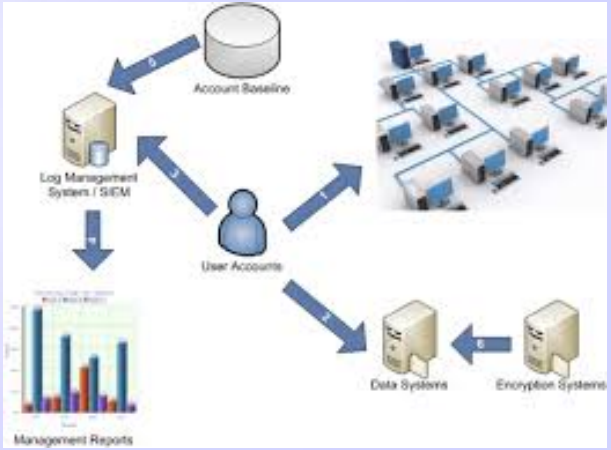
Conduct all data validation on a trusted system (e.g., The server)
Identify all data sources and classify them into **trusted and untrusted.** Validate all data from untrusted sources (e.g., Databases, file streams, etc.)
Specify proper character sets, such as UTF-8, for all sources of input.
Encode data to a common character set before validating.
All validation failures should result in input rejection
Verify that header values in both requests and responses contain only ASCII characters
Validate data length
Validate all input against a "white" list of allowed characters, whenever possible



Secure coding practices Checklist

Require authentication for all pages and resources, except those specifically intended to be public.
All authentication controls must be enforced on a trusted system (e.g., The server).
Establish and utilize standard, tested, authentication services whenever possible.
Use a centralized implementation for all authentication controls, including libraries that call external authentication services.
Segregate authentication logic from the resource being requested and use redirection to and from the centralized authentication control.
All authentication controls should fail securely.

AUTHENTICATION AND PASSWORD MANAGEMENT



Secure coding practices Checklist

If your 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 writeable only by the application.

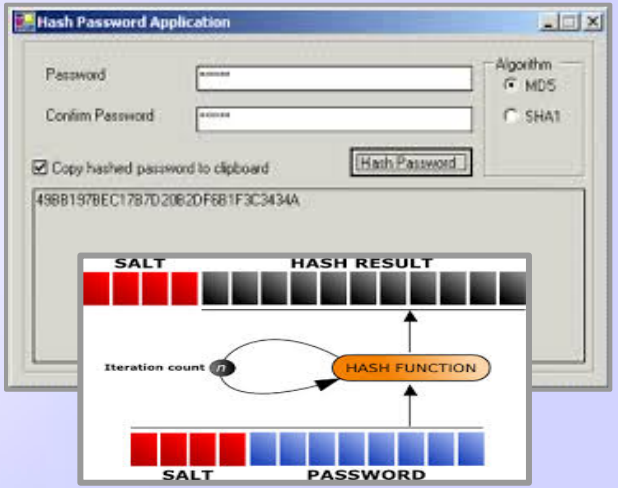
Password hashing must be implemented on a trusted system (e.g., The server).

Validate the authentication data only on completion of all data input.

Authentication failure responses should not indicate which part of the authentication data was incorrect.

For example, instead of "Invalid username" or "Invalid password", just use "Invalid username and/or password" for both.

AUTHENTICATION AND PASSWORD MANAGEMENT



Secure coding practices Checklist

Utilize **authentication for connections to external systems** that involve sensitive information or functions.

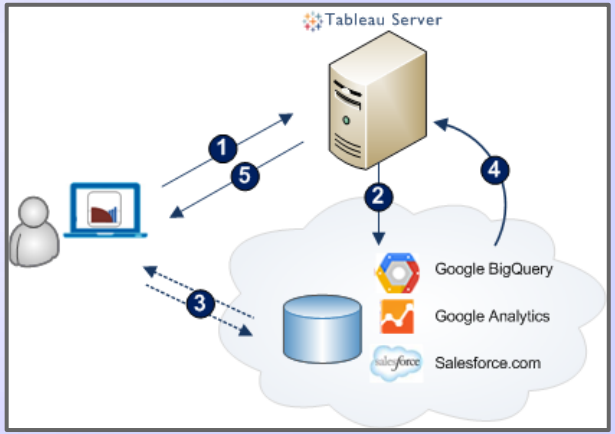
Authentication credentials for accessing services external to the application should be encrypted and stored in a protected location on a trusted system (e.g., The server). **The source code is NOT a secure location.**

Use only HTTP POST requests to transmit authentication credentials.

Only send non temporary passwords over an encrypted connection or as encrypted data, such as in an encrypted email. Temporary passwords associated with email resets may be an exception.

Enforce password complexity requirements established by policy or regulation.

Authentication and Password Management



Secure coding practices Checklist

Password entry should be obscured on the user's screen. (e.g.,on web forms use the input type "password")

Enforce account disabling or delaying after an established number of invalid login attempts(e.g.,five attempts is common).

Password reset questions should support sufficiently random answers. (e.g., "favorite book" is a bad question because “The Bible” is a very common answer)

If using email based resets, **only send email to a pre-registered address** with a temporary link/password.

Temporary passwords and links should have a short expiration time.

Authentication and Password Management

Secure coding practices Checklist

Use the server or framework's session management controls. **The application should only recognize these session identifiers as valid.**
Session identifier creation must always be done on a trusted system (e.g., The server)
Logout functionality should fully terminate the associated session or connection.
Logout functionality should be available from all pages protected by authorization.
Establish a session inactivity timeout that is as short as possible, based on balancing risk and business functional requirements.
Disallow persistent logins and enforce periodic session terminations, even when the session is active.

SESSION MANAGEMENT

Secure coding practices Checklist

If a session was established before login, **close that session and establish a new session after a successful login**.

Generate a new session identifier on any re-authentication.

Do not allow concurrent logins with the same user ID.

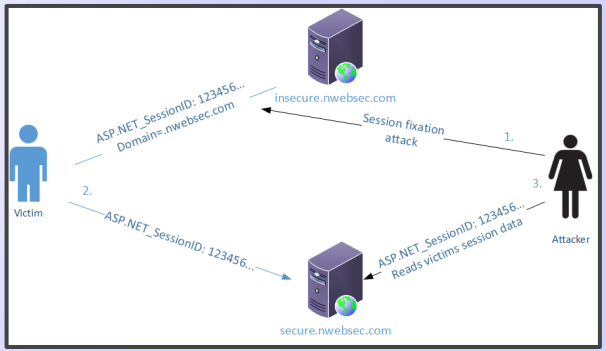
Do not expose session identifiers in URLs, error messages or logs. Session identifiers should only be located in the HTTP cookie header.

Protect server side session data from unauthorized access, by other users of the server, by implementing appropriate access controls on the server.

Set the "secure" attribute for **cookies transmitted over an TLS connection.**

Generate a new session identifier if the connection security changes from HTTP to HTTPS.

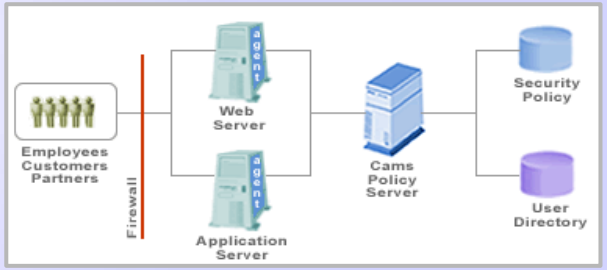
SESSION MANAGEMENT



Secure coding practices Checklist

Use only trusted system objects, e.g.server side session objects, for making access authorization decisions.
Access controls should fail securely.
Deny all access if the application cannot access its security configuration information.
Enforce authorization controls on every request, including those made by server side scripts.
Restrict access to protected URLs to only authorized users.
Restrict access to protected functions to only authorized users.
Restrict direct object references to only authorized users.
Restrict access to application data to only authorized users.

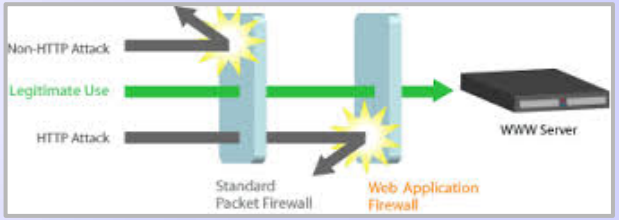
ACCESS CONTROL



Secure coding practices Checklist

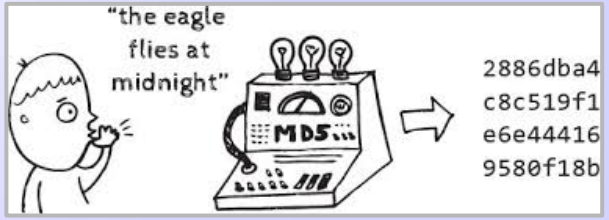
Restrict access to user and data attributes and policy information used by access controls.
If state data must be stored on the client, **use encryption and integrity checking on the server side to catch state tampering.**
Enforce application logic flows to comply with business rules.
Limit the number of transactions a single user or device can perform in a given period of time. The transactions/time should be above the actual business requirement, but low enough to deter automated attacks.
If long authenticated sessions are allowed, periodically re-validate a user's authorization to ensure that their privileges have not changed and if they have, log the user out and force them to re-authenticate.
Implement account auditing and enforce the disabling of unused accounts.

ACCESS CONTROL



Secure coding practices Checklist

All cryptographic functions used to protect secrets from the application user **must be implemented on a trusted system** (e.g., The server).
Protect master secrets from unauthorized access.
Cryptographic modules should fail securely.
Cryptographic modules used by the application should be compliant to FIPS 140-2 or an equivalent standard.
Establish and utilize a policy and process for how cryptographic keys will be managed.



Secure coding practices Checklist

Error Handling and Logging	Do not disclose sensitive information in error responses, including system details, session identifiers or account information.
	Implement generic error messages and use custom error pages.
	The application should handle application errors and not rely on the server configuration.
	Properly free allocated memory when error conditions occur.
	Error handling logic associated with security controls should deny access by default.
	All logging controls should be implemented on a trusted system (e.g., The server)
	Restrict access to logs to only authorized individuals.
	Do not store sensitive information in logs, including unnecessary system details, session identifiers or passwords.

Secure coding practices Checklist

Error Handling and Logging	Ensure that a mechanism exists to conduct log analysis.
	Log all input validation failures.
	Log all authentication attempts, especially failures.
	Log all access control failures.
	Log all apparent tampering events, including unexpected changes to state data.
	Log attempts to connect with invalid or expired session tokens.
	Log all system exceptions.
	Log all backend TLS connection failures

Secure coding practices Checklist

Data Protection	Implement least privilege, restrict users to only the functionality.
	Encrypt highly sensitive stored information, like authentication verification data, even on the server side.
	Protect server side source code from being downloaded by a user.
	Remove comments in user accessible production code that may reveal backend system or other sensitive information.
	Remove unnecessary application and system documentation.
	Protect all cached or temporary copies of sensitive data stored on the server from unauthorized access and purge those temporary working files a soon as they are no longer required.

Secure coding practices Checklist

Communication Security	Implement encryption for the transmission of all sensitive information.
	TLS certificates should be valid and have the correct domain name, not be expired.
	Failed TLS connections should not fall back to an insecure connection.
	Utilize TLS connections for all content requiring authenticated access and for all other sensitive information.
	Specify character encodings for all connections.
	Filter parameters containing sensitive information when linking to external sites.

Secure coding practices Checklist

System Configuration	Ensure servers, frameworks and system components are running the latest approved version.
	Ensure servers, frameworks and system components have all patches issued for the version in use.
	Turn off directory listings.
	Restrict the web server, process and service accounts to the least privileges possible.
	When exceptions occur, fail securely.
	Remove all unnecessary functionality and files.
	Implement a software change control system to ma nage and record changes to the code.

Secure coding practices Checklist

Database Security	Utilize input validation and output encoding and be sure to address meta characters.
	Ensure that variables are strongly typed.
	The application should use the lowest possible level of privilege when accessing the database.
	Use secure credentials for database access.
	Close the connection as soon as possible.
	Remove or change all default database administrative passwords.
	Remove unnecessary default vendor content.