Secure Development Standard

1. Introduction

This document is intended as a guideline for the secure use of web technologies in line with OWASP Top 10 and therefore to be used to raise awareness, assist understanding and provide guidance on:

* Ensuring the application and security design is tightly integrated
* Exploiting areas of the best practice for development teams
* Developing test plans and methods
* Defining maintenance methodologies that will increase security and minimise risk.
  1. Context and Conventions

This standard supports associated policies or processes by stating detailed controls that need to be implemented in a technical security context. What must be done. How These individual controls are implemented and maintained needs to be evidenced in documentation managed at the operational level.

In this standard, controls are written using 'MUST', 'SHOULD' and 'MAY'. Where these are present, they are defined as follows:

- Controls that use 'MUST' require mandatory compliance

- Controls that use 'SHOULD' are considered recommendations that will further enhance security and should be followed where feasible

- Controls that use 'MAY' give permission to perform a specific action

1. Scope

This standard states secure development control requirements and where applicable links to other policies, standards and processes. Software owners or technical/operational delegations are responsible for ensuring the controls in this standard are implemented.

This standard does not replace any local or global laws, government regulations, or other legislative requirements.

This standard is prepared for those accountable or responsible for requesting, designing, developing, testing, implementing, maintaining and disposing softwares.

* 1. In scope

This standard covers:

- softwares developed in-house (by employees, contractors or third parties) that are accessible internally or externally

- changes to existing softwares developed in-house or acquired but modified (e.g. open source softwares).

- softwares that run on the internet, intranet or used by employees, customers and contractors.

- cloud services

* 1. Out of scope

This standard does not cover:

- softwares from government or regulatory body

- softwares that are exempt from the requirements due to pre-existing contractual restrictions.

1. Requirements

Ensure that initial risk assessments are conducted, security control and design documentation is created.

* 1. General Controls
     1. A risk based approach MUST be utilised to identify security requirements that will be embedded into the security design for new softwares and changes to existing softwares.
     2. Recorded open security risks MUST be considered.
     3. Cyber security MUST be engaged in the process to validate that security requirements are met.
     4. Developers (including contractors) MUST take part in secure software development training.
  2. Security Artefacts
     1. Software information MUST be registered and maintained within an approved asset inventory.
     2. The security design of softwares including all required security controls MUST be documented and maintained.

Note: Typical system security documentation will include identified security controls, data classification, data flows, solution/infrastructure diagrams, and legal/regulatory requirements.

* + 1. In conjunction with the software owner, newly developed software MUST be determined and documented in BIA.
    2. BIA MUST be recorded on jira environment.

1. Secure Design and Development
   1. General Design Requirements
      1. Multiple layers of security MUST be considered in the design suitable with the risk of the functionality or data being protected.
      2. Where applicable, software entities MUST be designed to enable data segregation.
      3. Web Browsers
         1. As web browsers reside on a client’s machine, it is impossible to force the latest patches or updates to be installed. However, if a particular version of a browser is identified to be a security risk, it MUST be blocked from accessing the application. The customer SHOULD be able to check which browsers are supported and which will be blocked.
         2. Browsers maintain a cache both in memory and in long term store on disk. Cached data is available for retrieval at a later time and it is intended to help speed up rendering of pages. However this can be pose a security risk if a page displays sensitive information or accepts sensitive information via forms. To stop these pages from being retrieved by a different user or at a later date, pages MUST NOT be cached. Avoid web page cache in memory by disabling caching for HTTP 1.0/1.1
      4. Application Servers
         1. Transport level encryption for connection to other systems (e.g. database connections) MUST be applied before data transmission.
         2. Ensure connectivity credentials (such as database user password) are not stored in configuration files or transmitted in plain text.
         3. All accesses to remote systems (such as databases) MUST have the minimum required level of access and privilege. To support this, applications MUST use specific and dedicated service accounts to access other systems.
         4. For database queries, prepared statements or stored procedures MUST be used where possible. This serves to allow programmers to hide the SQL call details limiting risk exposure to SQL injection threats.
         5. Private keys MUST never be held in clear text in configuration files, or exposed in logs and server dumps. Where applicable HSMs MUST be used as preferred key management systems.
      5. Browser Based Applications
         1. Standard secure coding practices, including input validation, output encoding and error handling MUST be applied by design. Data validations MUST be clearly identified in the design requirements.
         2. No open redirects: Application design MUST support a configurable list or pattern of authorised destinations for any property that automatically redirects to a different page based on a field or parameter sent from the browser.
      6. Mobile Based Applications
         1. Applications MUST implement certificate pinning: Ensures the application only trusts legitimate back-end certificates.
         2. Application MUST implement root/jailbreak detection: Ensures the application is running on a trusted environment.
         3. Code obfuscation MUST be applied on the code: Ensures the confidentiality of the code when the application binaries are decompiled.
         4. Verify hostname of servers: Ensures the application is connected to the correct server (e.g. target URL matches certificate hostname)
         5. Do not ignore TLS errors
         6. RASP tooling MUST be considered for ruther controls.
   2. Sensitive Information
      1. All system information MUST be classified. processed, stored and transferred in accordance with related documents.
      2. Where feasible, systems MUST be designed to avoid exposing the names of internal resources to users. Where such resources have to be used, the application MUST be designed to use positive validation and ensure authorisation at the time of access.
      3. Restricted or Highly Restricted data MUST NOT be included in source code as clear text, including hard-coded credentials (e.g. passwords, tokens).
      4. Highly restricted data SHOULD NOT be used as a unique identifier/key for data handling.
      5. Any Restricted or Highly Restricted data that has been cached or written to temporary files MUST be cleared immediately upon a system error, or before a program terminates.
      6. Data masking requirements MUST be identified during software design and managed in accordance with related standards.
      7. Browsers allow data entered into fields to be cached in a different manner. Such caching is known as auto-completion. Where sensitive information is being requested within a form auto-complete functionality MUST bu disabled.
   3. Authentication and Authorisation Requirements
      1. A risk based approach MUST be used to identify authentication mechanisms for internal/customer facing systems.
      2. Application and system access controls MUST be designed in accordance with Access control standards.
      3. Internal or internet facing applications MUST be assessed according to the risk based approach.
      4. Systems MUST NOT use the platform default administrator privilege for performing routine tasks.
      5. Security design MUST address business, legal or regulatory requirements to separate the data accessible to different groups of users (role based, record level access control).
      6. Automated file transfers and batch processes MUST have unique application and platform credentials assigned for system and database access.
      7. Automated file transfers and batch processes MUST ONLY be granted the minimum necessary authorisations required to execute the tasks required.
      8. All system components MUST authenticate and authorise all requests from other system components.
      9. Systems MUST be designed so that the password of all accounts used in automated processing can be changed.
      10. In the event of a failed logon attempt, the application MUST NOT reveal which credential was incorrect.
      11. Applications MUST NOT use HTTP Basic Authentication to meet secure design requirements.
   4. Session Management
      1. When concurrent logon is not allowed, if a new login request is submitted with an authenticated session identified, the currently authenticated session MUST be terminated, and a new session will be created with the credentials from the login request.
      2. The storage capabilities or repository used by the session management mechanism to temporarily save the session tokens MUST be secure, protecting the session tokens against local or remote accidental disclosure or unauthorised access.
      3. Inactive sessions MUST be timed out after a configurable period of time.
      4. Session tokens/IDs MUST be considered a security credential, classed as Highly Restricted information and MUST be protected as such.
      5. Session tokens MUST be unique for each client logon.
      6. Session token MUST be unpredictable to prevent guessing attacks through statistical analysis techniques.
      7. It MUST be infeasible to anticipate or calculate a session ID.
      8. Sufficient entropy for session management tokens SHOULD be created from a secure Random Number Generator function.
      9. Tokens used to identify authenticated sessions MUST ONLY be transmitted over encrypted communication channels.
      10. A new session token MUST always be set after authentication when users transition from anonymous to authenticated sessions.
      11. Session data MUST be removed and the session nullified after a reasonable period of inactivity.
      12. If hidden fields are used to maintain a session, the page MUST expire and MUST NOT be cached.
      13. Where possible, session data MUST be removed and the session nullified during a log off or any other exit point.
      14. Persistent cookies MUST NOT be used to maintain sessions.
      15. Session token MUST be protected during transmission, processing and storage.
      16. Where Highly Restricted data is collected by the system, this MUST be performed in a secure session, protected by strong encryption in accordance with related standards. This also includes user credentials.
      17. Any sensitive data MUST be encrypted if stored in either cookie or token.
      18. Links to 3rd party sites MUST NOT include the session ID/Token.
      19. Application MUST have a logout link, unless specifically mandated by business requirements.
   5. HTTP Security Headers
      1. CORS Policy MUST be applied: Cross Origin Resource Sharing (CORS) is a mechanism that enables a web browser to perform cross-domain requests using the XMLHttpRequest (XHR) Level 2 (L2) API in a controlled manner. In the past, the XHR L1 API only allowed requests to be sent within the same origin as it was restricted by the Same Origin Policy (SOP).
      2. CSP MUST be applied: Content Security Policy (CSP) is a security feature that is used to specify the origin of content that is allowed to be loaded on a website or in a web application. It is an added layer of security that helps to detect and mitigate certain types of attacks, including Cross-Site Scripting (XSS) and data injection attacks. These attacks are used for everything from data theft to site defacement to distribution of malware.
      3. X-Frame-Options MUST be applied: The X-Frame-Options HTTP response header can be used to indicate whether a browser should be allowed to render a page in a , <iframe>, or . Sites can use this to avoid click-jacking attacks, by ensuring that their content is not embedded into other sites.
      4. HSTS MUST be applied: HTTP Strict Transport Security (also named HSTS) is a web security policy mechanism which helps to protect websites against protocol downgrade attacks and cookie hijacking. It allows web servers to declare that web browsers (or other complying user agents) should only interact with it using secure HTTPS connections, and never via the insecure HTTP protocol.
      5. Cache-Control Policy MUST be applied: This header holds directives (instructions) for caching in both requests and responses. If a given directive is in a request, it does not mean this directive is in the response (source Mozilla MDN). Specifying the capability of a resource to be cached is important to prevent exposure of information via the cache.
      6. Referrer Policy SHOULD be considered: The Referrer-Policy HTTP header governs which referrer information, sent in the Referer header, should be included with requests made.
      7. Content Type Options Should be considered: Setting this header will prevent the browser from interpreting files as a different MIME type to what is specified in the Content-Type HTTP header (e.g. treating text/plain as text/css).
   6. API Security
      1. APIs MUST be protected using relevant security policies for:

* Authentication and Authorisation: The client/user has been authenticated and they are authorised to access the API endpoint.
* Cross Origin Resource Sharing (CORS): Ensures the request is from trusted origin
* Rate limiting: defined maximum number of requests per interval to protect availability of downstream systems from denial of service
* Anti Cross Site Request Forgery: Ensures the actions based on input from trusted and authenticated users
* Replay attack protection: Ensures the security tokens can not be replayed against APIs for different user sessions
* Input validation: Protect from XSS, SQL/LDAP/XML injection and buffer overload threats
  + 1. Care MUST be taken to prevent leakage of sensitive information which can be captured in web server logs.
    2. Security information (e.g. passwords, tokens, etc) and toxic restricted data (such as customer details) MUST NOT appear in the URL.
    3. Consider transferring data in the request body or headers for POST/PUT requests and as HTTP header for GET requests. For additional details, see: OWASP Rest Security Cheat Sheet
  1. Cryptography and Key Management
     1. Application key management MUST comply with requirements of the related key management standards.
     2. Softwares which allocates memory for encryption keys MUST overwrite the encryption keys before freeing the memory.
     3. Credentials, encryption keys, and data used as key material MUST NOT be persisted or cached in process memory indefinitely, or loaded from configuration files as immutable objects.
     4. Private or symmetric key exchange MUST NOT occur over unencrypted channels.
     5. Different keys and salt values MUST be used in Testing, Staging and Production environments.
     6. The Security Design Document MUSt describe how any encryption keys or credentials used by the application are stored and processed.
     7. In accordance with the BIA, mobile applications with Extreme/Major business impacts MUST implement certificate pinning.
  2. Data Validation

Data validation ensures that the application is robust against all forms of input data, whether obtained from the user, infrastructure, external entities or database systems.

* + 1. Data validation MUST always be performed on the server side and where possible consider centralised input/output validation.
    2. Applications MUST not rely on client-side validation.
    3. All data received from a client MUST be treated as suspect, including headers and other HTTP protocol identifiers sent from the browser. Data MUST be validated against both business rules and technology constraints.
    4. Data from the client, whether from forms, cookies, HTTP headers or any other source potentially malicious and MUST be checked by the back-end server.
    5. Data input by the user may be checked on the client side for usability reasons but MUST NOT be seen as a security control, since any such checks can easily be bypassed/disabled by the user.
    6. Do not place any reliance on HTMl form controls to perform validation. All browser code can be modified. Including removing “disabled” tags and adding any amount of data desired to a control such as a list box, radio button or hidden field.
    7. Free format data to be displayed on the client's browser may contain malicious browser instructions, such as scripts. Therefore it is necessary to make this data safe by replacing the browser instructions with display on characters (e.g. applying HTML encoding).
    8. Where feasible, use “positive” (whitelisting) checks (check those values that are allowed and reject all others.).
    9. Check for expected format where appropriate (e.g. date of birth, email address).
    10. At a transaction/monetary related action, maximum/minimum values MUST be checked in line with business requirements.
    11. Check the length of a submission to prevent buffer overflow problems.
    12. Validation routines SHOULD be reviewed when new vulnerabilities are identified.
    13. All of the HTTP headers MUST be treated as insecure and MUST be sanitised.
    14. Take particular care over characters which could be used in attacks. At first do not allow the following characters or replace with their safe equivalents where applicable:
* High risk characters: <, >, ‘, “,
* Medium risk characters: %, ^, !, &, ;
  + 1. To reduce the risk of a buffer overrun, restrictions on the length of data stored within input fields MUST be enforced. These restrictions can be performed on the client side and MUST be performed on the server side.
    2. Before processing an incoming data field, if its length is greater than that permissible everything thereafter MUST be removed.
  1. File Uploads

File uploading is one of the most critical processes performed in web applications since it is importing unpredictable content to a server. It requires extra precautions to be set while implementing such functionality (e.g. profile photo, CV, sample file, video, ).

* + 1. Following controls MUST be applied during file upload:
* File Name/Type/Extention
* File Content
* File Content Type
  1. Sanitising Application Code for Production Released
     1. All code implemented to support debugging, including temporary code used to interrupt processes for troubleshooting. MUST be removed and code retested prior to release.
     2. Logging code intended for production can be left in but MUST be configurable levels of logging to control the amount of the detail recorded.
     3. When compiling code for a production environment, compiler options that generate structures to support debugging such as symbol tables, break points, etc. MUST be disabled.
     4. Displays or dumps of data variables MUST be removed.
     5. Unused code and partially developed code for future features MUST be removed.
     6. Debug/test facilities MUST be removed.
     7. Error/warning messages MUST be sanitised to ensure technical details are not revealed. Users MUST only be presented with information he/she is entitled to know.
     8. HTML comments are easily viewed from browsers, and can be used for social engineering. Snippets of code or other revealing information MUST be removed.
     9. Code within server-side comments (including in PHP, ASP, JSP, etc.) MUST also be removed.
     10. All comments MUST be reviewed prior to release.
  2. HTTP Request Methods
     1. Unless there is a specific design requirement, an application MUST generate an error if an HTTP method used to access a resource is different from that expected. For example, if a module is designed to process HTTP POST requests, and an HTTP GET request is received, the application MUST treat this as an error.
     2. GET URLs and query string parameters MUST NOT be used for any sensitive information such as session identifiers, security tokens, username or passwords.
     3. GET requests are commonly used as part of XmlHttpRequests (XHR). XHR GET requests don’t expose URLs in browser history. Risk exposing the full URL in server logs however remains.
  3. HTML Hidden Fields
     1. Restricted or Highly restricted data to which a user is not entitled MUST NOT be included in hidden form fields or HTML comments.
     2. Hidden form fields MUST be treated as changed and all data validation and sanitisation rules MUST be applied on hidden form fields.
  4. Error and Exception Handling
     1. Where feasible, applications MUST be designed such that when errors occur, the operation does not complete, the event is logged, and the user is returned to the same secure state as before the error ensuring that only minimal information is provided in the error notification.
     2. Errors MUST NOT increase the level of access or the authorisation available to the user (e.g. errors during authentication MUST NOT result in increased access to the application).
     3. Where applicable the exception handling capabilities of programming languages MUST be used.
     4. Where exception handling capabilities are not available, appropriate code to generate exceptions from functional error messages MUST be implemented to ensure that a consistent approach to error handling is adopted within the application.
     5. In the event of a total system failure all current sessions MUST be terminated.
     6. The try/catch construct MUST be used to handle all possible errors to prevent stack traces being displayed to the user.
     7. User-entered data or internal information MUST NOT be included in error messages.
     8. Implement error pages to support application errors.
  5. Thread Safety, Sharing and Reuse of Objects
     1. When a programming language allows sharing and re-use of objects, ensure that this does not permit data from one session to be accessed in another session. If a shared object is used, ensure that only one thread can process it. Ensure variables and buffers are cleared up after use.
     2. Consider carefully the use of static class variables when designing the contents of HTTP sessions to avoid unexpected sharing of information between sessions.
  6. Security Event Logging and Monitoring
     1. System logging MUST comply with the requirements of related logging standards and the legal or regulatory.
  7. Secure Development Methodology
     1. Applications MUST NOT be released to production or internet facing test environments, if critical or high risk vulnerabilities have been identified and not addressed, and communicated and accepted by the business owner.
     2. Changes within the development lifecycle MUST be controlled using approved change control procedures, ensuring the security of version control.
  8. Secure Development Environments
     1. Development activities MUST be performed in dedicated development environments, which are isolated from production and testing environments, regardless of whether they are performed by internal, contractor or third party resources.
     2. Access to system development environments MUST be controlled and strictly limited to development groups.
     3. Access to development documentation MUST be granted on the principle of least privilege.
  9. Coding
     1. Access to source code MUST be strictly limited and controlled based on the principle of least privilege.
     2. Approved code version control tool SHOULD be used.
     3. Approved source code software repositories MUST be used.
     4. All changes to source code in the source code repository MUST be applied by individual accountables (e.g. no use of shared accounts).
     5. The source code repository MUST hold a full version of all development activity, especially every version of systems released to testing or production environments.
     6. It MUST be possible to demonstrate that all compiled executables deployed to test or production environments were generated from source code held within the repository.
     7. The repository MUST contain all executable script files required for the operation of the production environment.
     8. Where source code is not owned for application or library used in development, third party escrow arrangements SHOULD be considered.
     9. Security controls identified during the assessment of design requirements MUST be integrated into the code.
     10. Security testing toolset MUST be used by developers to identify security flaws during development.
     11. Code reviews MUST NOT be performed solely by the developer of the code (apply separation of duties for code reviews).
     12. Maintenance hooks or backdoor logins MUST NOT be included in the code.
     13. All code implemented to support debugging, including temporary code used to interrupt processes for troubleshooting, MUST be removed and all code MUST be retested prior to release.
     14. All partially developed code for future features MUST be removed prior to release.
     15. All externally hosted third party code embedded in applications at runtime (e.g. JavaScript), MUST:
* have controls which protect and isolate the application
* be managed as part of the change control process.
  + 1. Approved libraries SHOULD be used where a specific security capability (e.g. encryption, authentication, secure number generation, logging, database access) is to be added to an application.
  1. Mobile Development
     1. Third party SDKs SHOULD be subject to a risk assessment prior to use.
     2. Industry standard tools SHOULD be used to obfuscate code to prevent unintentional information disclosure that could aid an attacker. Where possible 100% obfuscation should be achieved; if this is not possible all sensitive code should be sufficiently protected to prevent code enumeration.
     3. Mobile apps SHOULD detect whether the mobile device is jail-brokne or rooted, upon detection, the mobile apps SHOULD inform the end user.
     4. If required, mobile apps MUST store highly restricted information (e.g. password, encryption keys) with encryption in a secure location on the device.

1. Security Assessment
   1. New application or changes to existing applications MUST be subject to following assessments:

* In conjunction with application owner, critical assets associated with the application MUST be identified for inclusion in the scope of assessment.- Where applicable, existing security controls MUST be validated during assessment. If existing security controls are assessed as inadequate, then new or upgraded security controls MUST be determined.
* Mitigating security controls MUST be communicated with where applicable.

1. Security Testing

During the security testing, the developed code and identified security controls are tested to ensure they function as intended.

* 1. General Controls
     1. Security testing MUST be undertaken in accordance with the Security Testing Standard.
     2. When testing is performed, evidence MUST be retained.
     3. Security testing model, scope, plan and approach MUST be determined in accordance with the risk, availability and technical complexity of the newly developed/changes application. Security test decision matrix SHOULD be used.
     4. Based on the decision, the security testing approach MUST include various types of testing such as automated scanning, manual testing and code scanning.
     5. Security test findings MUST be communicated to relevant parties (e.g. Business owners, service owners, development teams).
     6. Security issues MUST be addressed with a risk rating.
     7. Corrective actions MUST be verified to ensure the identified issues are remediated successfully.
     8. Risk acceptance process MUST be performed regarding the issues not remediated successfully before releasing to production.
  2. Testing Environments
     1. Security testing MUST be performed in the testing environments which are isolated from the production and development environments, regardless of whether this is performed by internal or third party resources.
     2. Testing environments MUST be configured as close to production as possible.
     3. Access to testing environments MUST be controlled regarding unauthorised accesses.

1. System Implementation
   1. Release to production MUST follow approved change control and release procedures, ensuring potential impacts to other systems and interdependencies have been identified and addressed.
   2. Approved automated release tools SHOULD be used to deploy applications to production.
2. System Operation
   1. Where possible, a post implementation review SHOULD be performed by a Development/Project Manager.
   2. BIA and risk assessment SHOULD be reviewed at least annually or when a major change.
3. System Decommissioning
   1. Approved secure disposal procedures for data, media etc. MUST be followed taking into account data retention policies.