Information Technology Services Office

Secure Web Application Development Guideline

Revision History

Date	Author	Version	Change Reference
28 Jan 2016	ISO Team	1.0	Initial version
7 Mar 2025	Cybersecurity Team	2.0	Updated content on standards and programming practices

Table of Contents

1. Introduction

To meet University needs, applications are often developed either in-house or outsourced to vendors. It is important that proper design and programming practices are adopted to keep the application systems secure to avoid any loss of data, especially confidential materials.

This guideline covers security considerations and actions to be taken at different application development stages to mitigate the occurrence of common software vulnerabilities. While the primary focus is on web applications and their supporting infrastructure, most of the guidelines can be applied to general software deployment platform with which system availability, reliability and confidentiality can be strengthened to an acceptable risk level.

2. Security Considerations in Application Design and Development

- ***General Secure Web Code Standards***
  - Keep it simple and structured
  - Enforce Input data validation
  - White-List instead of Black-List
  - Check parameters on suspicious input and calling dangerous functions
  - Perform boundary check
  - Use temporary files properly
  - Don't use dynamic global/public variables
  - Establish structured debugging/checkpoint method

- ***Secure design***

  Security must be considered in application design and development. It is very difficult to implement security measures properly and successfully after an application system has been developed. It is of paramount importance to know the value of what is being protected, the threats and vulnerabilities [Appendix A], and the consequences of being compromised.

- ***Secure the weakest link***

  It is always easier for attackers to go against a weak spot in an application than parts that look the strongest. Adopting proper security measures and having airtight code throughout the application leaving no holes is required. Otherwise, the application will just be as secure as the weakest link.

- ***Secure configuration***

  Applications should be designed to have the least system privileged processes and accounts. Critical administrative functions need to be divided into sub-tasks and assigned to individual administrators. Unnecessary and unused services, shares, protocols and ports should be disabled to reduce the potential areas of attack. Server must be configured to use SSL transmission for all type of data between the client and server.

- ***Secure application platform***

  Server platforms may contain some redundant information (e.g. online manuals, help databases, sample files and system defaults) which may cause the leakage of system information to hackers. Such unused or seldom used information should be removed from production servers to secure the application platforms.

- ***Secure sensitive data***

  Sensitive or personal data should be encrypted when stored in database and during transmission. When sensitive information is displayed, printed or used for demonstration or testing, it should be masked wherever possible.

- ***Secure public web services***

  Web services require stronger security than Web sites. Web services expose functionality and/or data in an open standardized way which implies that they are more vulnerable than those exposed in propriety ways. Measures include:
  - Authorize web service clients the same way web applications authorize users
  - Validate input before consuming it
  - Ensure that output sent to client is encoded to be consumed as data and not as scripts
  - Apply encryption on sensitive data to be sent to the client to ensure integrity on data exchanged
  - Data-in-transit protection - HTTPS
  - Ensure attachments are scanned for viruses before being saved
  - Limiting message size to an appropriate size to reduce the risk of DoS (Denial of Service) attacks

- ***Secure deployment***

  Security of a web application depends on the security of the application infrastructure. Deployment review should be conducted to assess the implementation of all application security measures formulated.
  - Infrastructure hardening
  - Secure configurations
  - Disable unnecessary services
  - Network Segmentation

3. Secure Application Programming Practices

Adopt the following practices to the possible extent:

3.1. Input validation

- Never trust inputs safety:
  - ❖ Assume all inputs are malicious notwithstanding from a trusted site.
- Consistent validation strategies:
  - ❖ Define minimum and maximum length for the data (e.g. {1,25})
  - ❖ Handle boundary conditions (Out of range values)
  - ❖ Validate all data before processing starts.
  - ❖ Centralize the validation codes in shared libraries/modules.
  - ❖ Validate inputs in both client and server sides and to be done in same logic.
- Minimize errors by confining input data:
  - ❖ Specify proper character sets, such as UTF-8.
  - ❖ Use Checkbox, Radio Button, List/Menu if possible.
  - ❖ Define the types of characters that can be accepted
    - often U+0020 to U+007E, though most special characters could be removed and control characters are almost never needed
    - Filter the following from the input or replace with escape sequence
      ~ ! # $ % ^ & * [ ] < > ' \r \n
- Preventing SQL Injection:
  - ❖ Do not pass the HTML forms parameters directly to system call or database query.
  - ❖ Verify Multiple-Query Injection
  - ❖ Demarcate Every Value in Your Queries
  - ❖ Check the Types of Users' submitted Values
  - ❖ Escape Every Questionable character in your queries
  - ❖ Localized the code (Avoid depending on other remote code if possible)
- For RESTful Web Service
  - ❖ Programming
    - Assist the user on inputs > Reject input > Sanitize (filtering) > No input validation
    - Make sure output encoding is very strong if no input validation or only sanitization adopted
    - Log input validation failures, particularly if you assume that client-side code you wrote is going to call your web services.
    - Avoid using Hidden interfaces and Hidden Input
  - ❖ Secure Parsing
    - Use a secure parser for parsing incoming messages
  - ❖ Strong typing
    - It's difficult to perform most attacks if the only allowed values are true or false, or a number, or one of a small number of acceptable values.
  - ❖ Validate incoming content-types
    - When POSTing or PUTting new data, the client will specify the Content-Type (e.g. application/xml or application/json) of the incoming data.

- Preventing Cross-Site Scripting
  - ❖ Do not display the HTML forms parameters directly in the processing response.
  - ❖ Verify input on
    - HTML and CSS Markup Attacks
    - JavaScript Attacks
    - Forged Action URIs
    - Forged Image Source URIs
    - Extra Form Baggage

3.2. Output Encoding

- Send security headers
  - ❖ Server should always send the Content-Type header with the correct Content-Type, and preferably the Content-Type header should include a charset to make sure the content of a given resource is interpreted correctly by the browser
- JSON encoding
  - ❖ Use a proper JSON serializer to encode user-supplied data properly to prevent the execution of user-supplied input on the browser
- XML encoding
  - ❖ Ensures that the XML content sent to the browser is parseable and does not contain XML injection
- XSS and Content Security Policy
  - ❖ Prevent the execution of malicious data and encode into a non-executable format

Character	Entity Name	Description
“	&quot;	quotation mark
‘	&apos;	apostrophe
&	&amp;	ampersand
<	&lt;	less-than
>	&gt;	greater-than

- XML injection
  - ❖ Prevent the interpretation of xml, <> “ ‘ &
- SQL injection
  - ❖ Ensure the inputs should not be immediately or directly used as part of the SQL statement. Input should be verified and parameterized before used
- OS injection
  - ❖ Avoid sending user-controlled data to OS and preventing user from adding additional characters that can be executed by the OS

3.3. Sanitize Application Response

- Encrypting sensitive information in Cookies
- Don’t reveal sensitive information to the HTML with encryption: credit card number, HKID, personal telephone/mobile number, etc.
- Don’t include comments about application logic in the HTML response
- Don’t include unnecessary internal system information like internal IP address, internal host name, internal directory structure, etc. in the response
- Don’t include verbose error messages in case of internal server errors; most application/web servers allow you to custom an error page in case of internal server

error.
- Session ID should not be predictable.
- Store Session ID in Cookie (encrypted when needed)

3.4. Authentication

- Applications should have password policies implemented including
  - ❖ Password complexity requirement
  - ❖ Password rotation
  - ❖ Online password guessing attempt limits
- Account lockout, anti-automation mechanisms
- Password reset functions
- Email verification functions
- Secure password storage (password hash and one-way encryption scheme)

3.5. Session Management

- Creation:
  - ❖ Session ID must be created on a trusted server only.
  - ❖ Where appropriate, make session ID long, complex, randomized and untraceable.
  - ❖ Session ID length should be greater than 128-bit
  - ❖ Session ID creation should be generated via a secure random number generator
  - ❖ Inactivity timeout should be included
- Lifetime:
  - ❖ Lifetime should be bound closely to session/connection termination.
  - ❖ Enforce a short session inactivity timeout.
  - ❖ Establish a minimum session termination scheme while sufficient for normal business activity to complete.
- Storage:
  - ❖ Do not store session ID in hidden variables like hidden fields, HTTP headers, URL, persistent cookies.
  - ❖ Session ID stored in client browsers should have an expiry time and be encrypted.
  - ❖ When user logout, the session ID should be invalidated on server side.
  - ❖ Use session token and API key to maintain client state in a server-side cache
  - ❖ Use time-limited encryption key
  - ❖ Use some form of encryption or encoding
- Transmission:
  - ❖ Use SSL to send the session ID to server.
  - ❖ Encrypt the session ID before sending it to server.
- Prevent Session Attack
  - ❖ Define based on session management from application server
  - ❖ Protect your sessions with SSL or TLS, which will encrypt the entire transaction.
  - ❖ Insist on using cookies rather than variables.

- ❖ Define session timeout.
- ❖ Regenerate session IDs when users change status.
- ❖ Rely on tested code abstraction.
- ❖ Avoid ineffective supposed solutions.
- ❖ Session Identifier ("Session ID") for authentication with classified data shall be protected
- ❖ Don't share session ID for multiple connections
- ❖ Keep sensitive session values at servers

3.6. Authorization

- Anti-farming
- May use CAPTCHA
- Protect HTTP methods
- Permitted HTTP methods should be defined in the RESTful API library
- Whitelist allowable methods
- Needs to define the service properly restrict the allowable verbs
- Protect privileged actions and sensitive resource collections
- Needs to send along as a cookie or body parameter from unauthorized use
- Protect against cross-site request forgery:
- Based on random tokens
- Insecure direct object references
- A data contextual check needs to be done, server side, with each request.
- For RESTful web services
    - ❖ should use session-based authentication, either by establishing a session token via a POST or by using an API key as a POST body argument or as a cookie.
        - Usernames, passwords, session tokens, and API keys should not appear in the URL, as this can be captured in web server logs, which makes them intrinsically valuable.

3.7. Access Control

- Centralize the user access controls in shared libraries/modules for deriving access authorization decisions.
- Access controls must be compelled on each request disregarding the request sources (e.g. from client-side, server-side, from AJAX, or Flash).
- Restrictive functions should be provided for granting minimum access right to individual users for information retrieval/updating.
- Account authorized to directly connect to backend database, or to run SQL, or to run OS commands, must be limited to least execution privileges.
- Restrictive access should be applied to application program files, web server and configuration files.
- Data files, backups, temporary outputs should be kept in different directories.
- Ensure unauthorized function requests cannot be performed
- Ensure requested access has been verified before accessing particular target data.
- Ensure unauthorized access to data should be prohibited

3.8. Cross Domain Request Forgery

- Preventing CSRF by implementing a secure random token to handle state changing operations.
- CSRF token should have:
- Unique per user & per user session
- Tied to a single user session
- Large random value
- Generated by a cryptographically secure random number generator
- Include the CSRF as hidden field for forms or within URL
- Ensure server rejects the requested action if CSRF token fails validation
- Malicious Site Framing (Clickjacking) – Prevention
- Set the x-frame-options header for all responses containing HTML content. The possible values are "DENY" or "SAMEORIGIN".

3.9. File Uploads

- Upload verification should be performed to
  - ensure filename via expected extension type
  - limit maximum file size
- Upload Storage should be controlled with:
  - Do not use any user-controlled text for filename
  - All user uploaded files should be stored on a separate domain and undergo malicious code scanning
- Ensure uploaded content to be served with correct content-type (e.g. image/jpeg)
- “Special” files should be controlled through:
  - Whitelist approach to only allow specific file types and extensions
  - Prohibit use of crossdomain.xml or clientaccesspolicy.xml
  - Per-directory access control should not be enabled
- Perform upload verification through
  - Image rewriting libraries to verify the image is valid and to strip away extraneous content.
  - Setting the extension of stored image to be a valid image extension based on the detected content type of the image
  - Ensure the detected content type of the image within a list of defined image types
- Preventing Remote Execution
  - Remote Execution includes misusing the internal logic of the application in order to:
  - execute arbitrary commands or Scripts
- Strategies for countering Remote Execution:
  - Limit allowable filename extensions for uploads.
  - Allow only trusted, human users to import code.
  - Do not execute function with untrusted input.
  - Do not include untrusted files.
  - Properly escape all shell commands.
  - Beware of input patterns that can be executed as functions.
- Enforcing Security for Temporary Files
  - Temporary files safe used to keep state of using PHP including interim versions, temporary database query caches, temporary storage for files, session

properties, interim storage for data
- ❖ Make locations difficult (i.e. make the location unique and difficult to be guessed)
- ❖ Make permissions restrictive
- ❖ Write to known files only
- ❖ Read from known files only
- ❖ Check uploaded files
- ❖ Test protection against hijacking

3.10. Secure Transmission

- o Use POST only to send requests
- o Ensure login, logout page encrypted by HTTPS
- o Ensure forms should be submitted using POST over HTTPS
- o Ensure authenticated pages should be served over HTTPS including CSS, scripts, images
- o Implement HTTP Strict Transport Security (HSTS) for pages that are excluded from HTTPS

3.11. Secure Transmission

- o Use POST only to send requests
- o Ensure login, logout page encrypted by HTTPS
- o Ensure forms should be submitted using POST over HTTPS
- o Ensure authenticated pages should be served over HTTPS including CSS, scripts, images
- o Implement HTTP Strict Transport Security (HSTS) for pages that are excluded from HTTPS

3.12. Vulnerabilities of hidden parameters

- o Unless the integrity of the HTTP headers is guaranteed, do not trust CGI environment variables for security decisions as they can be spoofed by attackers.
- o Must not pass CGI variables directly to database queries or service/system calls.
- o CGI variables should not be revealed directly in web page responses.
- o Hidden fields, cookies are easily manipulated by attackers, so security control (e.g. cryptographic techniques) should be applied to ensure their trustfulness.
- o Don’t trust HTTP_REFERER, and other HTTP headers from untrusted parties
- o Don’t pass HTTP header’s parameters directly to system call or database query.
- o Don’t display HTTP header’s parameters directly in the processing response.
- o Don’t assume hidden parameters cannot be changed by users

3.13. Error Handling and Logging

- Enforce error handler to be invoked when error is detected.
- Centralize the error handling codes in shared libraries/modules with generic messages being implemented.
- Use custom error message page rather than default system message page.
- Error messages should be meaningful to end users and/or support staff.
- Do not include sensitive information in error messages (e.g. personal data, internal server data, debug trace).
- Exception handling routine must prevent further code execution.
- Enable server side logging controls for failure events.
- Centralize the logging control in shared libraries/modules.
- No sensitive information should be revealed in the log, such as system details, session ID, passwords.
- Access to logs should be restricted to authorized people only.

3.14. Others

- Restrict the types of files to be uploaded to server, such as only gif, jpeg are allowed for images.
- Ensure uploaded files are scanned for viruses before being saved.
- Encrypt sensitive pages and specify no-cache for them
- Don’t put data files, temporary, or backup files in the web directories
- Recommend using Java or .Net as the server side programming platform due to their robust security features, extensive libraries, and strong community support

4. Application Security Testing

Security testing involves the examination on the ability to mitigate vulnerabilities from the security perspective.

- ***Data Security Testing***
  - Ascertain testing should include input of valid, invalid and combination of both types of data.
  - Ascertain data containing sensitive information (e.g. HKID, credit card number) should be protected by masking or modification beyond recognition.
  - Ascertain passwords and sensitive data are not stored in cookies.
  - Ascertain session/cookie data are stored in encrypted format.
  - Ascertain session/cookie data are removed/expired upon logout.
  - Ascertain correct authorization data is used on all access-controlled pages.
  - Ascertain no production data being used for testing. If this is unavoidable, prior approval should be obtained. All this data must be cleared after testing.

- ***Page Security Testing***
  - Ascertain rules for authorization checking are implemented on all access-controlled pages.
  - Ascertain no access to secured web pages without login.
  - Ascertain error messages should not display any sensitive or important information.
  - Ascertain no internal system information, such as application, server, or database information, should be exposed when system malfunction occurs.
  - Ascertain crucial operations are written in log files in which tracking information must be recorded.

5. Application Change Control

In order to maintain integrity of application and to reduce the exposure to fraud and errors, the following change controls should be adopted:

- A proper procedure for requesting and approving application modification must be established.
- Changes can only be made after formal approval has been obtained.
- All changes must be tested and re-accessed to ensure that the application as a whole can be effectively protected from attacks or from being compromised**.**

Appendix A: Common Vulnerabilities in Web Applications

A01:2021 – Broken Access Control	• Access control ensures that policies are enforced to prevent users from exceeding their authorized permissions • Failures in access control can result in unauthorized disclosure, modification, or destruction of data, or enable users to perform actions beyond their permitted scope.
A02:2021 – Cryptographic Failures	• Any failure responsible for the exposure of sensitive and critical data to an unauthorized entity can be considered a cryptographic failure. • The first thing is to determine the protection needs of data in transit and at rest. For example, passwords, credit card numbers, health records, personal information, and business secrets require extra protection, mainly if that data falls under privacy laws,
A03:2021 – Injection	• An application is vulnerable to attack when: • The application does not validate, filter, or sanitize user-provided input. • Dynamic queries or non-parameterized calls are employed without applying context-aware escaping. • Malicious data is incorporated into object-relational mapping (ORM) search parameters to retrieve additional sensitive records. • Untrusted data is directly utilized or concatenated, leading to SQL or command structures that combine legitimate functionality with harmful payloads in dynamic queries, commands, or stored procedures.
A04:2021 – Insecure Design	• Focuses on risks related to design and architectural flaws, expressed as "missing or ineffective control design", e.g. • Lack of input validation controls • Sensitive information disclosure • Missing secure communication layers
A05:2021 – Security Misconfiguration	• Security controls not implemented / wrongly implemented • Security hardening configuration absent • Unnecessary features enabled (ports, services, pages, accounts, roles) • Use of default credentials • Error messages with debugging information returned • The security settings in the application servers, application frameworks (e.g., Struts, Spring, ASP.NET), libraries, databases, etc., are not set to secure values.

A06:2021 – Vulnerable and Outdated Components	• Use of vulnerable / unsupported components • Server-side applications • Client-side libraries • Includes components you directly use as well as nested dependencies • Unaware of vulnerabilities in the OS, database server, runtime environments libraries and related applications • Not keeping an inventory of third-party components used
A07:2021 – Identification and Authentication Failures	Confirmation of the user's identity, authentication, and session management is critical to protect against authentication-related attacks.
A08:2021 – Software and Data Integrity Failures	Software and data integrity failures relate to code and infrastructure that does not protect against integrity violations. An example of this is where an application relies upon plugins, libraries, or modules from untrusted sources, repositories, and content delivery networks (CDNs). An insecure CI/CD pipeline can introduce the potential for unauthorized access, malicious code, or system compromise
A09:2021 – Security Logging and Monitoring Failures	Logging and monitoring can be challenging to test, often involving interviews or asking if attacks were detected during a penetration test. There isn't much CVE/CVSS data for this category, but detecting and responding to breaches is critical. Still, it can be very impactful for accountability, visibility, incident alerting, and forensics.
A10:2021 – Server-Side Request Forgery (SSRF)	SSRF flaws occur whenever a web application is fetching a remote resource without validating the user-supplied URL. It allows an attacker to coerce the application to send a crafted request to an unexpected destination, even when protected by a firewall, VPN, or another type of network access control list (ACL).

More Information: https://owasp.org/www-project-top-ten/