Table of Contents

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 guide 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.

Security Considerations in Application Design and Development

- ***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 consequence 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 being displayed, printed or used for demonstration or testing, it should be masked wherever possible.

- ***Secure authorization***

  Authorization scheme are required for the granting of minimum access rights to individual users for information retrieval/update. System documentation and listings of applications shall be kept to the minimum access.

- ***Secure session protection***

  Web application relies on stateless http protocol with session control using cookies for authentication purposes. Prevalent means should be adopted to protect session authentication cookies, including:
  - Encrypt the cookies before sending to server.
  - Use SSL to send the cookies to server.
  - Limit the cookies lifetime.

- ***Secure input validation***

  There is no single solution to how the input can be securely validated throughout an application. Yet, some basic principles are:
  - Never trust inputs safety
  - Use consistent validation strategies
  - Minimize errors by confined input data
  - Sanitize input by encoding
  - Ensure all validation failures results in input rejection.

- ***Secure error handling***

  Return meaningful message when errors are detected. Message contents should be helpful to end users and/or support staff. But no sensitive data or internal system details should be disclosed. Structured exception handling should be adopted to prevent further code execution if application failure occurs.

- ***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
  - Ensure attachments are scanned for virus 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.

Secure Application Programming Practices

Adopt the following practices to the possible extent:

- **Input validation**
  - Never trust inputs safety:
    - ❖ Assume all inputs are malicious notwithstanding from a trusted site.
  - Consistent validation strategies:
    - ❖ 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.
    - ❖ Data must be strongly typed, length checked.
    - ❖ For numeric data, ranges must be checked and unsigned (unless required).
    - ❖ For data codes, format or syntax should be checked.
    - ❖ Reject known bad inputs, e.g. malicious script text:<script>, Javascript, onLoad, etc.
  - Minimize errors by confining input data:
    - ❖ Specify proper character sets, such as UTF-8.
    - ❖ Use Checkbox, Radio Button, List/Menu if possible.
    - ❖ Encode/escape potentially hazardous characters, such as "~!#$%^&*[]<>'\r\n".
  - Sanitize inputs by encoding:
    - ❖ Inputs will be treated as literal text and thus non-executable.
    - ❖ Do not pass the HTML parameters directly to system call or database query.
  - Ensure all validation failures result in input rejection.
- **Vulnerabilities of hidden parameters**
  - 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.
  - Must not pass CGI variables directly to database queries or service/system calls.
  - CGI variables should not be revealed directly in web page responses.
  - Hidden fields, cookies are easily manipulated by attackers, so security control (e.g. cryptographic techniques) should be applied to ensure their trustfulness.
- **Sanitized application responses**
  - Centralize the outbound encoding in shared libraries/modules.

- Use correct output encode value for the data being used; HTML character entities (e.g. &quot; for "), URL encodes (e.g. %22 for ").
- Outputs, returned codes, error returns from system calls, service calls should be checked for actual calls being invoked.
- Sensitive or personal data should be masked in display, printout or being used for demonstration or testing.
- Sensitive or personal data must not be included in HTTP GET request parameters.
- Disable the auto complete feature on form field of sensitive data inputs.
- Do not include unnecessary application logic comments, system infrastructure information such as internal IP address, internal host name and internal directory structure.

- **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 message should be meaningful to end user and/or support staff.
  - Do not include sensitive information in error message (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 persons only.

- **Session management**
  - Creation:
    - ❖ Session ID must be created on a trusted server only.
    - ❖ Where appropriate, make session ID long, complex, randomized and untraceable.
  - 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 be encrypted.
  - Transmission:
    - ❖ Use SSL to send the session ID to server.
    - ❖ Encrypt the session ID before sending to server.

- **Access Authorization**
  - 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.
- **Others**
  - Restrict the types of files to be uploaded to server, such as only gif, jpeg are allow for images.
  - Ensure uploaded files are scanned for virus before being saved.

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 password and sensitive data are not stored in cookies.
  - Ascertain session/cookie data are stored in encrypted formatted.
  - 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 these 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.

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 proceeded 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

Injection	Injection flaws, such as SQL, OS, and LDAP injection occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.
Broken Authentication and Session Management	Application functions related to authentication and session management are often not implemented correctly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities.
Cross-Site Scripting (XSS)	XSS flaws occur whenever an application takes untrusted data and sends it to a web browser without proper validation or escaping. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.
Insecure Direct Object References	A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, or database key. Without an access control check or other protection, attackers can manipulate these references to access unauthorized data.
Security Misconfiguration	Good security requires having a secure configuration defined and deployed for the application, frameworks, application server, web server, database server, and platform. Secure settings should be defined, implemented, and maintained, as defaults are often insecure. Additionally, software should be kept up to date.
Sensitive Data Exposure	Many web applications do not properly protect sensitive data, such as credit cards, tax IDs, and authentication credentials. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data deserves extra protection such as encryption at rest or in transit, as well as special precautions when exchanged with the browser.
Missing Function Level Access Control	Most web applications verify function level access rights before making that functionality visible in the UI. However, applications need to perform the same access control checks on the server when each function is accessed. If requests are not verified, attackers will be able to forge requests in order to access functionality without proper authorization.
Cross-Site Request Forgery (CSRF)	A CSRF attack forces a logged-on victim's browser to send a forged HTTP request, including the victim's session cookie and any other automatically included authentication information, to a

	vulnerable web application. This allows the attacker to force the victim’s browser to generate requests the vulnerable application thinks are legitimate requests from the victim.
Using Components with Known Vulnerabilities	Components, such as libraries, frameworks, and other software modules, almost always run with full privileges. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications using components with known vulnerabilities may undermine application defenses and enable a range of possible attacks and impacts.
Unvalidated Redirects and Forwards	Web applications frequently redirect and forward users to other pages and websites, and use untrusted data to determine the destination pages. Without proper validation, attackers can redirect victims to phishing or malware sites, or use forwards to access unauthorized pages.

Information Source: https://www.owasp.org/index.php/Top_10_2013-Top_10