MINISTRY OF COMMUNICATIONS
AND MULTIMEDIA MALAYSIA

CyberSecurity
MALAYSIA

First edition
2020-05-05

Guidelines for Secure Software Development Life Cycle (SSDLC)

Reference number:
MyVAC-3-GUI-2-SSDLC-v1

REGISTERED OFFICE:

CyberSecurity Malaysia,
Level 7 Tower 1,
Menara Cyber Axis,
Jalan Impact,
63000 Cyberjaya,
Selangor Darul Ehsan, Malaysia
Email: myvac@cybersecurity.my

COPYRIGHT © 2020 CYBERSECURITY MALAYSIA

NO ENDORSEMENT

Products and manufacturers discussed or referred to in this document, if any, are presented for informational purposes only and do not in any way constitute product approval or endorsement by CyberSecurity Malaysia.

TRADEMARKS

All terms mentioned in this document that are known to be trademarks or service marks have been appropriately capitalised. CyberSecurity Malaysia cannot attest to the accuracy of this information. Use of a term in this document should not be regarded as affecting the validity of any trademark or service mark.

DISCLAIMER

This document is for informational purposes only. It represents the current thinking of CyberSecurity Malaysia on the security aspects of the Secure Software Developemt Life Cycle environment. It does not establish any rights for any person and is not binding on CyberSecurity Malaysia or the public. The information appearing on this guideline is not intended to provide technical advice to any individual or entity. We urge you to consult with your own Secure Software Developemt Life Cycle advisor before taking any action based on information appearing on this guideline or any other documents to which it may be linked.

Contents Page

1 Introduction 1
1.1 Scope 1
1.2 Objective 2
1.3 Intended audience 2
2 Terms, definitions, abbreviated terms and acronyms 2
2.1 Terms and Definitions 2
2.2 Abbreviated terms and acronyms 5
3 Secure Software Development Life Cycle (SSDLC) 6
4 Phase 1: Security requirements 7
4.1 Sources of security requirements 7
4.1.1 Identify core security requirements 7
4.1.2 Identify general requirements 10
4.1.3 Identify operational requirements 10
4.1.4 Identify other requirements 10
4.2 Data classification 10
4.2.1 Types of data 11
4.2.2 Labeling the data 11
4.2.3 Data ownership and roles 11
4.2.4 Data lifecycle management (DLM) 11
4.2.5 Privacy requirements 11
4.3 Use case and misuse case modeling 12
4.3.1 Analyze the use case scenarios 12
4.3.2 Analyze the misuse case scenarios 13
4.3.3 Creating attack model 13
4.3.4 Select mitigation control 13
4.4 Risk management 13
4.4.1 Risk assessment 13
4.4.2 Risk mitigation 15
4.4.3 Evaluation and assessment 16
5 Phase 2: Security design 17
5.1 Core security design considerations 17
5.1.1 Confidentiality design 17
5.1.2 Integrity design 18
5.1.3 Availability design 18
5.1.4 Authentication design 18
5.1.5 Authorization design 18
5.1.6 Accountability design 19
5.2 Additional design considerations 19
5.2.1 Programming languages 19
5.2.2 Data type, format, range, and length 19
5.2.3 Database security 20
5.2.4 Interface design 20
5.2.5 Interconnectivity 21
5.3 Threat modeling 21
5.3.1 Step 1: Decompose the software 21
5.3.2 Step 2: Determine and rank threats 22
5.3.3 Step 3: Determine countermeasures and mitigation. 22

6 Phase 3: Security development 22
6.1 Common software vulnerabilities and controls 22
6.1.1 Vulnerability databases 22
6.1.2 Defensive coding practices 23
6.2 Secure software processes 23
6.2.1 Source code versioning 23
6.2.2 Code analysis 23
6.2.3 Code review 23
6.2.4 Developer testing 23
6.3 Securing build environments 24
6.3.1 Physically securing access to the software's that building code. 24
6.3.2 Using access control lists (ACLs) 24
6.3.3 Using the version control software 24
6.3.4 Build automation 24
6.3.5 Code signing routine 24

7 Phase 4: Security testing 25
7.1 Attack surface validation 25
7.1.1 Post-development testing 25
7.1.2 Perform security testing using security testing methods 25
7.1.3 Perform software security testing for quality assurance 25
7.2 Test data management 26
7.2.1 Identify output test data to confirm software requirements 26
7.2.2 Apply testing with synthetic transactions 26
7.2.3 Test data management solutions 26
7.2.4 Defect reporting and tracking 27

8 Phase 5: Security deployment 27
8.1 Software acceptance considerations 27
8.1.1 Completion criteria 27
8.1.2 Change management 27
8.1.3 Approval to deploy or release 27
8.1.4 Risk acceptance and exception policy 28
8.1.5 Documentation of software 28
8.2 Verification and validation (V&V) 28
8.2.1 Reviews 28
8.2.2 Testing 28
8.3 Certification and accreditation (C&A) 29
8.3.1 Obtain certification 29
8.3.2 Obtain accreditation 29
8.4 Installation 29
8.4.1 Hardening 29
8.4.2 Environment configuration 30
8.4.3 Release management 30
8.4.4 Bootstrapping and secure startup 30

9 Phase 6: Security maintenance 30
9.1 Operations, monitor and maintenance 30
9.1.1 Carry out the operations security 30
9.1.2 Continuous monitoring 31
9.1.3 Audit for monitoring 31
9.2 Incident management 31

9.2.1 Determine events, alerts, and incidents 31
9.2.2 Identify types of incidents 32
9.2.3 Incident response process 32
9.3 Problem management 32
9.3.1 Incident notification 32
9.3.2 Root cause analysis 32
9.3.3 Solution determination 33
9.3.4 Request for change 33
9.3.5 Implement solution 33
9.3.5 Monitor and report 33
9.4 Change management 33
9.4.1 Patch and vulnerability management 33
9.4.2 Backups, recovery and archiving 34
9.5 Disposal 34
9.5.1 End-of-Life policies 34
9.5.2 Sun-setting criteria 34
9.5.3 Sun-setting processes 34
9.5.4 Information disposal and media sanitization 35
10 SSDLC Checklist 36
10.1 Phase 1: Security Requirements (4) 36
10.2 Phase 2: Secure Design (5) 37
10.3 Phase 3: Security Development (6) 38
10.4 Phase 4: Security Testing (7) 39
10.5 Phase 5: Security Deployment (8) 40
10.6 Phase 6: Security Maintenance (9) 40
Annex A 42
Annex B 44
Annex C 47
Annex D 49
Annex E 50
Bibliography 52
Acknowledgements 54

1 Introduction

This document provides a guideline for Secure Software Development Life Cycle (SSDLC) to highlight the security tasks for each phase involves in the development processes. SSDLC consists of six (6) phases; there are security requirement, security design, security development, security testing, security deployment, and security maintenance phases. This guideline describes security information such as security tasks, which incorporate into every phase in producing secure software to ensure the confidentiality, integrity, and availability of their information systems.

The applying of security tasks into the development life cycle are become vital and needed to clarify several problems. The high costs of remediation whenever the vulnerabilities have been identified after the deployment of the software become the major problem to the organization. As consequences, it will be related to a breach and then give effect to an organization. Therefore, the organization needs to ensure the appropriate security controls with security tasks are in place throughout the development life cycle. The organization should plan for security to incorporate security from the beginning of any software development. The organization has assured the appropriate security tasks included in the design phase to meet the requirement phase. The processes continue for the development of software securely and assure the security requirements have been met during implementation. The organization should conduct ongoing reviews to maintain the appropriate level of security in the deployed software.

This guideline will suggest several security tasks of controls to ensure the development of secure software from the earlier processes. Organizations can take this SSDLC guideline to use it as a blueprint to apply the security control in all phases involved in secure software development processes.

1.1 Scope

This document provides guideline for specific security tasks of each phase in Secure Software Development Life Cycle (SSDLC) for the target audience in incorporating the security features in the development of software. The guideline only focuses on the development of secure software for web applications, which assume that the usage of components or codes or frameworks for developments is under a controlled environment. In addition, the secure software also developed not included the cloud-based and external or third-party environments.

This document gives guidance to individuals and organizations with respect to common security tasks throughout the six (6) phases involves in SSDLC. The phases are security requirement, security design, security development, security testing, security deployment, and security maintenance phases. The selection of appropriate security tasks will assist target audiences in minimizing the potential risks in software development. The common example architecture of secure software (Figure A-1) and example of secure software architectures with protection controls to be developed (Figure A-2) can be referred to in Annex A. The provided security tasks describe the importance and relevancy of the steps to be conducted in all phases. Moreover, the guideline describes the knowledge of security control and steps as guidance for implementing the security tasks.

The provided checklist will contain all security tasks, separated into six (6) different phases. Target audience can use the checklist instantly provided that all the security tasks have been followed or applied in the development of secure software.

1.2 Objective

The main objective of this document is to provide guideline of Secure Software Development Life Cycle (SSDLC) for the target audience in ensuring the security features are integrated into the development of secure software. The specific objectives are:

a) To provide security tasks for each phase of SSDLC for the target audience to ensure the development of secure software.
b) To provide the checklist of security tasks as a quick guide for the target audience in applying the SSDLC.

1.3 Intended audience

This guideline is created for the use of a wide range of audiences for information systems and information security professionals including i) individuals with information security management responsibilities (e.g., project managers, information security officers), which usually range from early to end development phases; ii) individuals development responsibilities (e.g., software developers, programmers, software analyst, software architects), which usually range from design to deployment phases; iii) other related stakeholders (software owners, consultant, auditors), which usually involve at early or end development phases.

2 Terms, definitions, abbreviated terms and acronyms

2.1 Terms and Definitions

For the purposes of this document, the terms and definitions as the following apply.

2.1.1
Access control
means to ensure that access to assets is authorized and restricted based on business and security requirements

[ISO/IEC 27000: 2014(E)]

2.1.2
Authentication
provision of assurance that a claimed characteristic of an entity is correct

[ISO/IEC 27000: 2014(E)]

2.1.3
Availability
property of being accessible and usable upon demand by an authorized entity

[ISO/IEC 27000: 2014(E)]

2.1.4
Confidentiality
property that information is not made available or disclosed to unauthorized individuals, entities, or processes

[ISO/IEC 27000: 2014(E)]

2.1.5
Integrity
property of accuracy and completeness

[ISO/IEC 27000: 2014(E)]

2.1.6
Objective
result to be achieved

[ISO/IEC 27000: 2014(E)]

2.1.7
Risk
effect of uncertainty on objectives (5.6)

[ISO/IEC 27000: 2014(E)]

2.1.8
Process
set of interrelated or interacting activities which transforms inputs into outputs

[ISO/IEC 27000: 2014(E)]

2.1.9
Risk treatment
process (5.8) to modify risk (5.7)

[ISO/IEC 27000: 2014(E)]

2.1.10
Residual risk
risk (5.7) remaining after risk treatment (3.72)

[ISO/IEC 27000: 2014(E)]

2.1.11
Risk assessment

process of identifying the risks to system security and determining the probability of occurrence, the resulting impact, and additional safeguards that would mitigate this impact. Part of Risk Management and synonymous with Risk Analysis

[NIST Special Publication 800-30]

2.1.12
Spoofing

"Identity spoofing" is a key risk for applications that have many users but provide a single execution context at the application and database level. Users should not be able to become any other user or assume the attributes of another user

[OWASP Code Review Guide Version 2.0.]

2.1.13
Tampering

users can potentially change data delivered to them, return it, and thereby potentially manipulate client-side validation, GET and POST results, cookies, HTTP headers, and so forth. The application should also carefully check data received from the user and validate that it is sane and applicable before storing or using it

[OWASP Code Review Guide Version 2.0.]

2.1.14
Repudiation

users may dispute transactions if there is insufficient auditing or recordkeeping of their activity

EXAMPLE If a user says they did not make a financial transfer, and the functionality cannot track his/her activities through the application, then it is extremely likely that the transaction will have to be written off as a loss

[OWASP Code Review Guide Version 2.0.]

2.1.15
Information disclosure

users are rightfully wary of submitting private details to a system. It is possible for an attacker to publicly reveal user data at large, whether anonymously or as an authorized user

[OWASP Code Review Guide Version 2.0.]

2.1.16
Denial of Service

application designers should be aware that their applications may be subject to a denial of service attack. The use of expensive resources such as large files, complex calculations, heavy-duty searches, or long queries should be reserved for authenticated and authorized users, and not available to anonymous users

[OWASP Code Review Guide Version 2.0.]

2.1.17
Elevation of privilege

if an application provides distinct user and administrative roles, then it is vital to ensure that the user cannot elevate his/her role to a higher privilege one

[OWASP Code Review Guide Version 2.0.]

2.2 Abbreviated terms and acronyms

CSM	CyberSecurity Malaysia
DoS	Denial of Service
DREAD	Damage, Reproducibility, Exploitability, Affected Users, and Discoverability
EOL	End-of-Life
FIPS	Federal Information Processing Standards
IPL	Initial Program Load
MyCC	Malaysia Common Criteria
NIST	National Institute Standard Technology
NT	New Technology
NTLM	New Technology LAN Manager
OWASP	Open Web Application Security Project
POST	Power-on self-test
SANS	SysAdmin, Audit, Network and Security
SDLC	Software Development Life Cycle
SSDLC	Secure Software Development Life Cycle
STRIDE	Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege
WAF	Web application firewall

3 Secure Software Development Life Cycle (SSDLC)

Secure Software Development Life Cycle (SSDLC) follows an iterative process of phases such as security requirement, security design, security development, security testing, security deployment, and security maintenance. Furthermore, the SSDLC methodology adapts the security controls to the project life cycle to produce secure software.

The main purpose of SSDLC methodology is to confirm the presence of essential components in the environment of the software are secure, starting from the earlier phase of this secure software development processes. SSDLC also necessitates for cost savings from the early integration of security within the SDLC, which could help avoid expensive design flaws and increase the long-term viability of software projects.

This guideline provides several specific security tasks required for each phase of SSDLC methodology. These are the common security controls involved in developing secure software. Figure 1 below presents the security tasks of SSDLC phases.

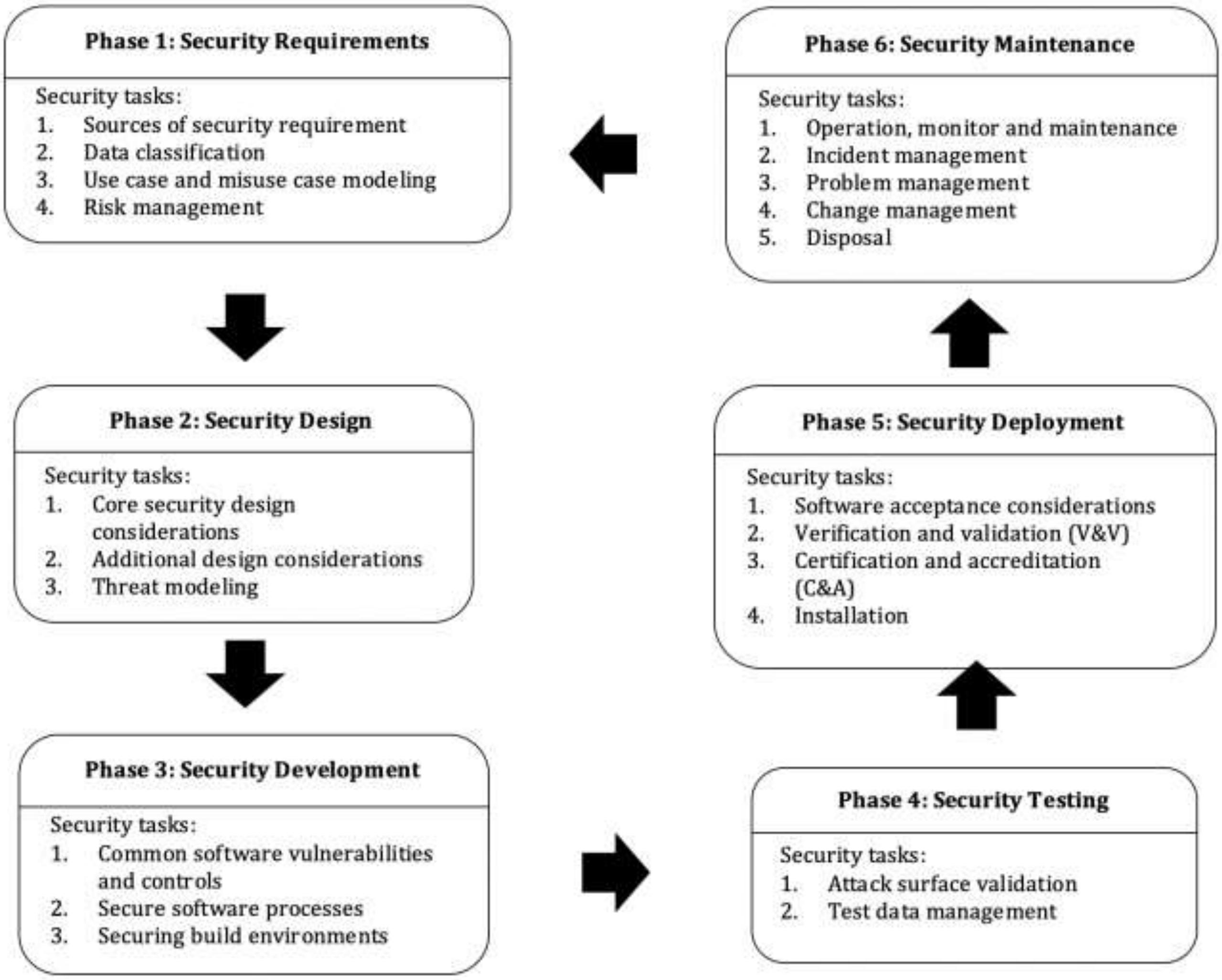
Phase 1: Security Requirements
Security tasks:
1. Sources of security requirement
2. Data classification
3. Use case and misuse case modeling
4. Risk management
Phase 2: Security Design
Security tasks:
1. Core security design considerations
2. Additional design considerations
3. Threat modeling
Phase 3: Security Development
Security tasks:
1. Common software vulnerabilities and controls
2. Secure software processes
3. Securing build environments
Phase 4: Security Testing
Security tasks:
1. Attack surface validation
2. Test data management
Phase 5: Security Deployment
Security tasks:
1. Software acceptance considerations
2. Verification and validation (V&V)
3. Certification and accreditation (C&A)
4. Installation
Phase 6: Security Maintenance
Security tasks:
1. Operation, monitor and maintenance
2. Incident management
3. Problem management
4. Change management
5. Disposal

Figure 1: Security Tasks of SSDLC Phases

Developing secure software with SSDLC is about the processes of handling and managing risks from the early phase, the security requirement phase, which viewed as a non-functional requirement called software requirements specifications documents. The security software specifications will be translated into architectural blueprints, and identifying control based on risks will be conducted in the security design phase. Meanwhile, in the security development phase will use the architectural blueprints to code the software and apply secure coding to protect software and data. In security testing phase will validate and verify the functionality and security of software based on the requirement specification. Followed by security deployment phase, the verification will be conducted to ensure that the software meets the specified functional and assurance requirements. Finally, in security maintenance phase will monitoring ongoing operations. Maintenance includes addressing incidents impacting the software and patching the software to mitigate its chances of being exploited by hackers and malware threats.

4 Phase 1: Security requirements

<u>Objectives</u>

a) To identify and define relevant security requirements for the software developed.
b) To identify security preparation for software development. To elicit protection needs using data classification, use and misuse case modeling, and risk management.

There are four (4) types of security tasks, which are i) sources for security requirement; ii) data classification; iii) use case and misuse case modeling; iv) risk management; should be considered by the audience to accomplish the security requirement phase. The details will be provided in the next sub-section:

4.1 Sources of security requirements

<u>Objectives</u>

a) To explicitly define and address security objectives or goals of the organization.
b) To identify requirements that are applicable to business context and software functionalities serving the context.

<u>Implementation guide</u>

4.1.1 Identify core security requirements

a) Confidential requirements
To address protection against the unauthorized disclosure of data or information that is either private or sensitive in nature. Confidentiality protection mechanisms are such as follows:

   i) **Cryptographic** is a protection mechanism in which the goal is to prevent the disclosure of the information deemed secret. The types of mechanisms include:
      (1) overt[1] mechanisms, such as encryption and hashing. The goal of overt secret writing is to make the information humanly indecipherable or unintelligible even if disclosed.
      (2) covert[2] mechanisms, such as steganography and digital watermarking. The goal of covert secret writing is to hide information within itself or in some other media or form.

[1] Overt- [Over] Over and above all. Out in the open, obvious, for all to see, not hidden.
[2] Covert- [Cover] Hidden, camouflaged, cryptic. UNDER COVER.

ii) **Masking** is a weaker form of confidentiality protection mechanism in which the original information is either asterisked or X'ed out. This is primarily used to protect against shoulder surfing attacks, which are characterized by someone looking over another's shoulder and observing sensitive information. The masking of credit card numbers, except for the last four digits when printed on receipts or displayed on a screen, is an example of masking providing confidentiality protection.

Confidentiality requirements need to be defined throughout the information life cycle from the origin of the data in question to its retirement. It is necessary to explicitly state confidentiality requirements for non-public data:

i) In *Transit*: When the data is transmitted over unprotected networks, i.e., data-in-motion.
ii) In *Processing:* When the data is held in computer memory or media for processing
iii) In *Storage*: When the data is at rest, within transactional systems as well as non-transactional systems, including archives, i.e., data-at-rest.

Confidentiality requirements may also be *time-bound.*

b) Integrity requirements
To address two primary areas of software security, namely, reliability assurance and protection or prevention against unauthorized modifications. Integrity refers not only to the software modification protection (system integrity) but also the data that the software handles (data integrity).

Reliability – essentially refers to ensuring that the software is functioning as it is designed and expected to. It is also meant to provide security controls that will ensure that the *accuracy* of the software and data is maintained. Integrity protection also takes into consideration the *completeness* and *consistency* of the software or data that the software handles.

c) Availability requirements
To ensure that there is no disruption to business operations. Availability requirements are those software requirements that ensure the protection against destruction of the software and/or data, thereby assisting in the prevention against Denial of Service (DoS) to authorized users.

d) Authentication requirements
To verify and assure the legitimacy and validity of the identity that is presenting entity claims for verification. Authentication credentials could be provided by different factors or a combination of factors that include knowledge, ownership, or characteristics.

Two-factor authentication – when two factors are used to validate an entity's claim and/or credentials. *Multi-factor* authentication – when more than two factors are used for authentication.

The most common forms of authentication are as below:

i) **Anonymous authentication** is the means of access to public areas of your software without prompting for credentials such as username and password.
ii) **Basic authentication** is one of the HyperText Transport Protocol (HTTP) 1.0 specifications, which is characterized by the client browser prompting the user to supply their credentials.
iii) **Digest authentication** is a challenge/response mechanism, which unlike Basic authentication, does not send the credentials over the network in cleartext or encoded form, but instead sends a message digest (hash value) of the original credential.
iv) **Integrated authentication** is commonly known as New Technology LAN Manager (NTLM) authentication or New Technology (NT) challenge/response authentication, like Digest authentication, the credentials are sent as a digest.

v) **Client certificate-based authentication** works by validating the identity of the certificate holder. These certificates are issued to organizations or users by a certification authority (CA) that vouches for the validity of the holder.

vi) **Forms authentication** requires the user to supply a username and password for authentication purposes, and these credentials are validated against a directory store, which can be the active directory, a database or configuration file.

vii) **Token-based authentication** is usually used in conjunction with forms for authentication where a username and password are supplied for verification. Upon verification, a token is issued to the user who supplied the credentials. The token is then used to grant access to resources that are requested. This way, the username, and password need not be passed on each call.

viii) **Smart card-based authentication** provides the ownership (something you have). They contain a programmable embedded microchip that is used to store the authentication credentials of the owner.

ix) **Biometric authentication** uses biological characteristics (something you are) for providing the identity's credentials. Biological features such as retinal blood vessel patterns, facial features, and fingerprints are used for identity verification purposes.

e) Authorization requirements
To confirm, an authenticated entity has the needed rights and privileges to access and perform actions on a requested resource.

Identify the *subjects* and *objects*. Subjects are the entities that are requesting access, and Objects are the items that the subject will act upon. A subject can be a human user or a software process. Action on objects needs to be explicitly captured, commonly are Create, Read, Update, or Delete (CRUD) data operations.

The access control models to apply are primarily of the following types:

i) **Discretionary Access Control (DAC)** – restricting access to objects based on the identity of subjects and/or groups to which they belong. DAC is implemented either by using identities or roles. DAC is often observed to be implemented by using access control lists (ACLs), the relationship between the individuals (subjects) and the resources (objects) is direct and the mapping of individuals to resources by the owner.

ii) **Non-Discretionary Access Control (NDAC)** – characterized by the software enforcing the security policies. It does not rely on the subject's compliance with security policies. The non-discretionary aspect is that it is unavoidably imposed on all subjects.

iii) **Mandatory Access Control (MAC)** – access to objects is restricted to subjects based on the sensitivity of the information contained in the objects. The sensitivity is represented by the label. Only subjects that have the appropriate privilege and formal authorization (i.e., clearance) are granted access to the objects.

iv) **Role-Based Access Control (RBAC)** – individuals (subjects) have access to a resource (object) based on their assigned role. Permissions to operate on objects such as Create, Read, Update, or Delete are also defined and determined based on responsibilities and authority (permissions) within the job function.

v) **Resource-Based Access Control** – access be granted based on the resources. Resource-based access control models are useful in architectures that are distributed and multi-tiered, including service-oriented architectures.

vi) Accountability requirements - to assist in building a historical record of user actions. Audit trails can help detect when an unauthorized user makes a change, or an authorized user makes an unauthorized change, both of which are cases of integrity violations.

Examples of Security Requirements can be referred in Annex B.

4.1.2 Identify general requirements

a) Session Management requirements
Upon successful authentication, a session identifier (ID) is issued to the user and that session ID is used to track user behavior and maintain the authenticated state for that user until that session is abandoned or the state changes from authenticated to not authenticated.

b) Errors & Exceptions Management requirements
Errors & exceptions are potential sources of information disclosure.

c) Configuration Parameters Management requirements
Software configuration parameters and code which makeup the software needs protection against hackers. These parameters and codes usually need to be initialized before the software can run.

4.1.3 Identify operational requirements

To identify requirements such as the number of database connections for concurrent access, interdependencies with other applications in the computing ecosystem, and shared and computing resources required. Identify the needed capabilities and dependencies of the software as it serves the business with their intended functionality.

a) Deployment environment requirements
To identify and capture the pertinent requirements about the environment in which the software will be deployed.

b) Archiving requirements
Archives are maintained either as a means for business continuity or as a need to comply with a regulatory requirement or organizational policy; the archiving requirement should be explicitly identified and captured.

c) Anti-piracy requirements
Code obfuscation, code signing, anti-tampering, licensing, and IP protection mechanisms should be included as part of the requirements documentation, especially if in the business of building and selling commercial software.

4.1.4 Identify other requirements

a) Sequencing and timing requirements
Sequencing and timing design flaws in software can lead to what is commonly known as race conditions or Time of Check/Time of Use (TOC/TOU) attacks. Race conditions are, in fact, one of the most common flaws observed in software design.

b) International requirements
International requirements can be of two types – *legal* and *technological*.
Legal requirements – so that are not in violation of any regulations.
Technological requirements – to determine character encoding and display direction

c) Procurement requirements
The identification and determination of software security requirements are no less important when a decision is made to procure the software instead of building it in-house. The brief explanations of the procurement requirements can be referred in Annex C.

4.2 Data classification

Objective

To ensure data or information as the most valuable digital assets will be protected.

Implementation guide

4.2.1Types of data

To classify data into types as follows:

a) **Structured data**
Referred to data which is organized into the identifiable structure. An example of structured data in a database (stored in columns and rows).

b) **Unstructured data**
Referred to data that has no identifiable structure. Examples of unstructured data include images, videos, emails, documents, and text.

4.2.2 Labeling the data

Data classification is an effort to assign *labels* (a level of sensitivity) to information (data) assets, based on potential impact to confidentiality, integrity, and availability (CIA), upon disclosure, alteration or destruction.

4.2.3 Data ownership and roles

Decisions to classify data, which has access and what level of access, etc. are decisions that are to be made by the business/data owner.

4.2.4 Data lifecycle management (DLM)

A policy-based approach, involving procedures and practices to protect data throughout the information life cycle: from the time it is created to the time it is disposed or deleted.

4.2.5 Privacy requirements

Data classification can help in identifying data that will need to have privacy protection requirements applied. Categorizing the data into privacy tiers, based on the impact upon disclosure, alteration and/or destruction, can provide insight into ensuring that appropriate levels of privacy controls are in place.

Best practice guidelines for data privacy that need to be included in software requirements analysis, design, and architecture can be addressed if one complies with the following rules.

a) If you don't need it, don't collect it.
b) If you need to collect it for processing only, collect it only after you have informed the user that you are collecting their information, and they have consented, but don't store it.
c) If you have the need to collect it for processing and storage, then collect it, with user consent, and store it only for an explicit retention period that is compliant with organizational policy and/ or regulatory requirements.
d) If you have the need to collect it and store it, then don't archive it, if the data has outlived its usefulness and there is no retention requirement.

Privacy requirements and controls are as below:

a) **Data anonymization** – By permanently and completely removing personal identifiers from data, anonymity can be assured. Anonymization is the process of removing private information from the data. Anonymized data cannot be linked to any one individual account.
The anonymization techniques are useful to assure data privacy are replacement (also known as substitution), suppression (also known as omission), generalization (specific identifiable information is replaced using a more generalized form), and perturbation (also known as randomization, that involves making random changes to the data).

b) **Disposition** – All software is vulnerable until it and the data it processes, transmits, and stores is disposed in a secure manner. This is particularly of great importance if the data is sensitive and/or personally identifiable.
Most privacy regulations require the implementation of policies and procedures to address the final disposition of private information and/or the sanitization of electronic hardware and media on which it is stored before the hardware is re-provisioned for re-use.

c) **Security models** – a formal abstraction of the security policy which is comprised of the set of security requirements that needs to be part of the software, so that it is resistant to attack, can tolerate the attacks that cannot be resisted and can recover quickly from the undesirable state if compromised.

d) **Pseudonymization** – is a data management and de-identification procedure by which personally identifiable information fields within a data record are replaced by one or more artificial identifiers or pseudonyms. A single pseudonym for each replaced field or collection of replaced fields makes the data record less identifiable while remaining suitable for data analysis and data processing.

An organization should also consider the privacy legislation such as in:

a) Malaysia, which is the Laws of Malaysia, Act 709, Personal Data Protection Act 2010 (PDPA).
b) The United States, which are the U.S. Privacy laws – Privacy Act (5 USC 552a), the consumer credit is addressed in the Fair Credit Reporting Act, healthcare information in the Health Insurance Portability and Accountability Act (HIPAA), financial service organizations in the Gramm–Leach–Bliley Act (GLBA), children's web access in the Children's Online Privacy Protection Act (COPPA), and student records in the Federal Educational Rights and Privacy Act).
c) Non-U.S. Privacy principles, such as for European countries which is the European Union (EU) General Data Protection Regulation (GDPR).

4.3 Use case and misuse case modeling

<u>Objective</u>

To identify possible misbehavior and recommend relevant security requirements based on functionality for developed software.

<u>Implementation guide</u>

4.3.1 Analyze the use case scenarios

To identify the behavior of the software. The use case describes behavior that the software owner intended. Use case modeling and diagramming is very useful for specifying requirements. It can be effective in reducing ambiguous and incompletely articulated business requirements by explicitly specifying exactly when and under what conditions certain behavior occurs.

Use case modeling includes identifying actors, intended software behavior (use cases), and sequences and relationships between the actors and the use cases. Actors may be an individual, a role, or a non-human in nature.

Actors are represented with stick people and use case scenarios by ellipses when the use case is diagrammatically represented. Arrows that represent the interactions or relationships connect the use cases and the actors.

4.3.2 Analyze the misuse case scenarios

To identify weaknesses that to become a threat. Misuse cases model negative scenarios. A negative scenario is an unintended behavior of the software, one that the software owner does not want to occur within the context of the use case. Misuse cases provide insight into the threats that can occur against the software.

In misuse case modeling, mis-actors, and unintended scenarios or behavior are modeled. Misuse cases may be intentional or accidental. Misuse cases can be created through brainstorming negative scenarios like an attacker.

4.3.3 Creating attack model

To create an attack model with explicit consideration of known attacks or attack types. Given a set of requirements and a list of threats, the idea here is to cycle through a list of known attacks one at a time and to think about whether the "same" attack applies to the software. To create an attack model, do the following:

a) Select those attack patterns relevant to the software. Build misuse cases around those attack patterns.
b) Include anyone who can gain access to the software because threats should encompass all potential sources of danger to the software.

4.3.4 Select mitigation control

To propose mitigation controls based on attack identified from the previous step. Highlight the point by proposing the security control; it helps in addressing security requirement.

Figure D-1 depicts a use case, and misuse case examples are shown in Annex D.

4.4 Risk management

<u>Objectives</u>

a) To enable securing the software that store, process, or transmit organizational information;
b) To enable the management to make well-informed risk management decisions to justify the expenditures that are part of a software budget; and
c) To assist the management in authorizing (or accrediting) the software based on the supporting documentation resulting from the performance of risk management.

<u>Implementation guide</u>

4.4.1 Risk assessment

To determine the extent of the potential threat and the risk associated with the software throughout its SSDLC. To identify appropriate controls for reducing or eliminating risk during the risk mitigation process.

The risk assessment methodology encompasses nine (9) primary steps, whereby steps 2, 3, 4, and 6 can be conducted in parallel after Step 1 has been completed.

The steps are as below:

a) Step 1 – System Characterization
To define the scope of the effort. In this step, the boundaries of the software are identified, along with the resources and the information that constitutes the system. Characterizing software establishes the scope of the risk assessment effort, delineates the operational authorization (or accreditation) boundaries and provides information (e.g., hardware, software, system connectivity, and responsible division or support personnel) essential to defining the risk.

b) Step 2 – Threat Identification
To identify the potential threat-sources and compile a threat statement listing potential threat-sources that are applicable to the software being evaluated. A threat-source is defined as any circumstance or event with the potential to cause harm to the software. The common threat sources can be natural, human, or environmental.

c) Step 3 – Vulnerability Identification
The analysis of the threat to the software should include an analysis of the vulnerabilities associated with the software environment. The goal of this step is to develop a list of software vulnerabilities (flaws or weaknesses) that could be exploited by the potential threat-sources.

d) Step 4 – Control Analysis
    i) To analyze the controls that have been implemented or are planned for implementation by the organization to minimize or eliminate the likelihood (or probability) of a threat's exercising software vulnerability.
    ii) To derive an overall likelihood rating that indicates the probability that a potential vulnerability may be exercised within the construct of the associated threat environment (Step 5 below), the implementation of current or planned controls should be considered.

e) Step 5 – Likelihood Determination
To derive an overall likelihood rating that indicates the probability that a potential vulnerability may be exercised within the construct of the associated threat environment, the following governing factors should be considered:
    i) Threat-source motivation and capability.
    ii) Nature of the vulnerability
    iii) Existence and effectiveness of current controls.

f) Step 6 – Impact Analysis
To determine the adverse impact resulting from a successful threat exercise of vulnerability. Before beginning the impact analysis, it is necessary to obtain the following necessary information as below:
    i) System mission (e.g., the processes performed by the software)
    ii) System and data criticality (e.g., the system's value or importance to an organization
    iii) System and data sensitivity.

g) Step 7 – Risk Determination
To assess the level of risk for the software. The determination of risk for a threat/vulnerability pair can be expressed as a function of:
    i) The likelihood of a given threat source's attempting to exercise a given vulnerability
    ii) The magnitude of the impact should a threat-source successfully exercise the vulnerability
    iii) The adequacy of planned or existing security controls for reducing or eliminating risk.

h) Step 8 – Control Recommendations
Controls that could mitigate or eliminate the identified risks, as appropriate to the organization's operations, are provided. The goal of the recommended controls is to reduce the level of risk to the

software and its data to an acceptable level. The following factors should be considered in recommending controls and alternative solutions to minimize or eliminate identified risks:

i) Effectiveness of recommended options (e.g., system compatibility)
ii) Legislation and regulation
iv) Organizational policy
v) Operational impact
vi) Safety and reliability.

i) Step 9 – Results Documentation
Once the risk assessment has been completed (threat-sources and vulnerabilities identified, risks assessed, and recommended controls provided), the results should be documented in an official report or briefing.

A risk assessment report is a management report that helps senior management, the mission owners, make decisions on policy, procedural, budget, and system operation and management changes. Unlike an audit or investigation report, which looks for wrongdoing, a risk assessment report should not be presented in an accusatory manner but as a systematic and analytical approach to assessing risk so that senior management will understand the risks and allocate resources to reduce and correct potential losses.

4.4.2 Risk mitigation

To prioritizing, evaluating, and implementing the appropriate risk-reducing controls recommended from the risk assessment process.

a) Risk mitigation options
Organizations should be considered in selecting any of these risk mitigation options. It may not be practical to address all identified risks, so priority should be given to the threat and vulnerability pairs that have the potential to cause significant mission impact or harm. The risk mitigation options are:

i) Risk Assumption. To accept the potential risk and continue operating the software or to implement controls to lower the risk to an acceptable level.
ii) Risk Avoidance. To avoid the risk by eliminating the risk cause and/or consequence (e.g., skip certain functions of the software or shut down the software when risks are identified).
iii) Risk Limitation. To limit the risk by implementing controls that minimize the adverse impact of a threat's exercising a vulnerability (e.g., use of supporting, preventive, detective controls).
iv) Risk Planning. To manage risk by developing a risk mitigation plan that prioritizes, implements, and maintains controls.
v) Research and Acknowledgment. To lower the risk of loss by acknowledging the vulnerability or flaw and researching controls to correct the vulnerability.
vi) Risk Transference. To transfer the risk by using other options to compensate for the loss, such as purchasing insurance.

b) Risk mitigation strategy
To provide guidance on actions to mitigate risks from intentional human threats:

i) When vulnerability (or flaw, weakness) exists – implement assurance techniques to reduce the likelihood of vulnerability's being exercised.
ii) When a vulnerability can be exercised, apply layered protections, architectural designs, and administrative controls to minimize the risk of or prevent this occurrence.
iii) When the attacker's cost is less than the potential gain – apply protections to decrease an attacker's motivation by increasing the attacker's cost (e.g., use of software controls such as limiting what a software user can access and do can significantly reduce an attacker's gain).
iv) When loss is too great – apply design principles, architectural designs, and technical and non-technical protections to limit the extent of the attack, thereby reducing the potential for loss.

c) Approach for control implementation
To take control action with the following steps:
   i) Step 1 – Prioritize Actions
   Prioritize the implementation actions based on the risk levels.
   ii) Step 2 – Evaluate Recommended Control Options
   To select the most appropriate control option for minimizing risk. Analyze the feasibility (e.g., compatibility, user acceptance) and effectiveness (e.g., degree of protection, and level of risk mitigation) of the recommended control options.
   iii) Step 3 – Conduct Cost-Benefit Analysis
   To aid management in decision making and to identify cost-effective controls, a cost-benefit analysis is conducted.
   iv) Step 4 – Select Control
   To determine the most cost-effective control(s) for reducing risk to the organization's mission.
   v) Step 5 – Assign Responsibility
   To identify and assign appropriate persons (in-house personnel or external contracting staff) who have the appropriate expertise and skillsets to implement the selected control.
   vi) Step 6 – Develop a Safeguard Implementation Plan
   To develop a safeguard implementation plan (or action plan).
   vii) Step 7 – Implement Selected Control(s)
   To implement controls, which may lower the risk level but not eliminate the risk (residual risk[3]).

d) Control categories
To consider technical, management, and operational security controls, or a combination of such controls, to maximize the effectiveness of controls for their software and organization.

Technical security controls for risk mitigation can be configured to protect against given types of threats. These controls may range from simple to complex measures and usually involve software architectures, engineering disciplines, and security packages with a mix of hardware, software, and firmware.Management security controls, in conjunction with technical and operational controls, are implemented to manage and reduce the risk of loss and to protect an organization's mission. Management controls focus on the stipulation of information protection policy, guidelines, and standards, which are carried out through operational procedures to fulfill the organization's goals and missions.

e) Cost-benefit analysis
To allocate resources and implement cost-effective controls, organizations, after identifying all possible controls and evaluating their feasibility and effectiveness, should conduct a cost-benefit analysis for each proposed control to determine which controls are required and appropriate for their circumstances.

f) Residual risk
To analyze the extent of the risk reduction generated by the new or enhanced controls in terms of the reduced threat likelihood or impact.

4.4.3 Evaluation and assessment

To emphasize the good practice and need for ongoing risk evaluation and assessment. There should be a specific schedule for assessing and mitigating mission risks, but the periodically performed process should also be flexible enough to allow changes where warranted, such as major changes to the software and processing environment due to changes resulting from policies and new technologies.

[3] Residual risk is the threat that remains after all efforts to identify and eliminate risk have been made.

5 Phase 2: Security design

Objectives

a) To design a secure architecture by considering security elements
b) To secure the software architecture by understanding and analyze threats against the software before it is built.

There are three (3) types of security tasks, which are i) core security design considerations; ii) additional design considerations; iii) threat modeling; to accomplish the security design phase. The details will be provided in the next sub-section:

5.1 Core security design considerations

Objective

To design the software to address the core security elements of confidentiality, integrity, availability, authentication, authorization, and auditing.

Implementation guide

5.1.1 Confidentiality design

To assure that disclosure protection can be achieved in several ways using cryptographic and masking techniques. Masking is useful for disclosure protection when data is displayed on the screen or on printed forms.

Cryptographic is for assurance of confidentiality, used when the data is transmitted or stored in transactional data stores or offline archives. Work factor involves cryptographic protection is exponentially dependent on:

a) **Key size (length)** - is the length of the key that is used in the algorithm.
b) **Key management** to protecting the secrecy of the key, which includes the generation, exchange, storage, rotation, archiving, and destruction of the key.
c) **Rotation** (swapping) of keys involves the expiration of the current key and the generation, exchange, and storage of a new key.

Encryption algorithms are primarily of two types:

a) **Symmetric algorithms** – using a single key for encryption and decryption operations that are shared between the sender and the receiver.
b) **Asymmetric algorithms** – using two keys is to be held secret and is referred to as the *private* key, while the other key is disclosed to anyone with whom secure communications and transactions need to occur. The key that is publicly displayed to everyone is known as the *public* key.
   i) Digital Certificates – which specifies formats for the public key, used by anyone to verify the authenticity of the certificate itself because it contains the digital certificate of the certificate authority.
   ii) Digital Signatures – to provide identity verification and ensure that the data or message has not tampered since the digital signature that is used to sign the message cannot be easily imitated by someone unless it is compromised. It also provides non-repudiation.

5.1.2 Integrity design

To assures that there is no unauthorized modification of the software or data, using any one of the following techniques or a combination of the techniques:

a) **Hashing (Hash functions**) – to condense variable-length inputs into an irreversible, fixed-sized output known as a message digest or hash value.
b) **Referential Integrity** – Integrity assurance of the data, especially in a relational database management system (RDBMS), which ensures that data is not left in an orphaned state. Uses primary keys and related foreign keys in the database to assure data integrity.
c) **Resource Locking** – When two concurrent operations are not allowed on the same object (say a record in the database), because one of the operations locks that record from allowing any changes to it until it completes its operation.

5.1.3 Availability design

To achieve destruction and DoS protection are by proper coding of the software. Since it's a design phase, configuration requirements such as connection pooling, the use of cursors, and looping constructs can be looked at. Techniques used to design the software for availability as follows:

a) **Replication** – to provide a single point of failure is characterized by having no redundancy capabilities, and this can undesirably affect end-users when a failure occurs.
b) **Failover** – to provide automatic switching from an active transactional software, server, system, hardware component, or network to a standby (or redundant) system. (Switchover is manual)
c) **Scalability** – to handle increasing (or growing) amount of work without degradation in its functionality or performance.

5.1.4 Authentication design

To determine the type of authentication that is required as specified in the requirements documentation, consider using multi-factor authentication and single sign-on (SSO).Recommended using multi-factor or the use of more than one factor to authenticate a principal (user or resource) provides heightened security.If there is a need to implement SSO, wherein the principal's asserted identity is verified once and the verified credentials are passed on to other systems or applications, usually using tokens, then it is crucial to factor into the design of the software both the performance impact and its security.

5.1.5 Authorization design

To give attention to the impact on performance, and to the principles of separation of duties and least privilege. The type of authorization to be implemented according to the requirements should be determined, such as roles or resource-based authorization.If roles are used for authorization, the design should ensure that there are no conflicting roles that circumvent the separation of duties principle. For example, a user cannot be in a teller role and also in an auditor role for a financial transaction. Designing for authorization can be accomplished using *entitlement management*, which is about granular access control.

Access control mechanism appropriate for general objects of unspecified types as listed below:

a) **Directory**
One simple way to protect an object is to use a mechanism that works like a file directory. Imagine we are trying to protect files (the set of objects) from users of a computing system (the set of subjects). Every file has a unique owner who possesses "control" access rights (including the rights to declare who has what access) and to revoke access to any person at any time. Each user has a file directory, which lists all the files to which that user has access.

b) **Access Control List (ACL)**
There is one such list for each object, and the list shows all subjects who should have access to the object and what their access is. This approach differs from the directory list because there is one access control list per object; a directory is created for each subject.

c) **Access control matrix**
A table in which each row represents a subject, each column represents an object, and each entry is the set of access rights for that subject to that object.

d) **Capability**
Unforgeable (not fake) token, that gives the certain rights to an object. In theory, a subject can create new objects and can specify the operations allowed on those objects. For example, users can create objects, such as files, data segments, or sub processes, and can also specify the acceptable kinds of operations, such as read, write, and execute. But a user can also create completely new objects, such as new data structures, and can define types of accesses previously unknown to the software.

e) **Procedure-oriented access control**
Imply the existence of a procedure that controls access to objects (for example, by performing its own user authentication to strengthen the basic protection provided by the basic operating system). The procedure forms a capsule around the object, permitting only certain specified accesses.

5.1.6 Accountability design

To audit the software especially in the event of a breach, primarily for forensic purposes. Log data should include the 'who', 'what', 'where', and 'when' aspects of software operations. As part of the 'who', it is important not to forget the non-human actors such as batch processes and services or daemons.

5.2 Additional design considerations

<u>Objective</u>

To design the software which addresses other design considerations that are needed when building the secure software.

<u>Implementation guide</u>

5.2.1 Programming languages

To determine the programming language that will be used to implement the design, which brings with its inherent risks or security benefits. There are two (2) main types of programming languages:

a) **Unmanages code** – e.g., C/C++, the execution of code is not managed by any runtime execution environment but directly executed by the operating system.

b) **Managed code** – e.g., Java and .NET (include C#, VB.Net), the execution of code is not by the operating system directly, but instead, it is managed by the runtime environment. Security and non-security services such as memory management, exception handling, bound checking, garbage collection, and type safety checking can be leveraged from the runtime environment, and security checks can be asserted before the code executes.

5.2.2 Data type, format, range, and length

To assure integrity with a data type, format, range, and length for design considerations.

a) **Primitive or built-in data types** are such as Character, Integer, Floating-point number, and Boolean.

b) **User-defined data types**, by programmers, are not recommended from a security standpoint because it potentially increases the attack surface, called as strongly typed programming languages.

c) **Set of values and the permissible operations** on that value set, which defined by the data type.
d) **Conversion mismatches and casting or conversion errors** could be detrimental to the secure state of the software. There are such as:
    i) Widening conversion (expansion) – data type is converted from a smaller size range to a larger size range. The potential loss of data is the loss of precision.
    ii) Narrowing conversion (truncation) – data type is converted from a larger size range to a smaller size range. Potentially cause data loss truncation if the value is stored in the new data type is greater than its allowed range.

5.2.3 Database security

To ensure reliability, resiliency, and recoverability of software that depends on the data stored in the database.Impacts of database security design are:

a) **Inference attack** – attacker, obtaining sensitive information about the database from presumably hidden and trivial pieces of information (legitimately obtained by the attacker) using data mining techniques without directly accessing the database.
b) **Aggregation attack** – information at different, security classification levels, which are primarily no sensitive in isolation, end up becoming sensitive information when pieced together as a whole.

Protection design considerations concerning database assets are:

a) **Polyinstantiation** – to provide several instances (or versions) of the database information, so that what is viewed by a user is dependent on the security clearance or classification level attributes of the requesting user.
   It's a database security approach to deal with problems of inference by hiding information using classification labels, and aggregation by labeling different aggregations of data separately.
b) **Database encryption** – to provide data-at-rest encryption is a preventive, control mechanism that can provide strong protection against disclosure and alteration of data. To ensure along with database encryption, proper authentication and access control protection mechanism exist to secure the key that is used for encryption.
c) **Normalization** – is a formal technique used to organize data so that redundancy and inconsistency are eliminated.
d) **Triggers and views** – Triggers are fired to run implicitly by the database when triggering events occurs. Also, very useful for automating and improving security protection mechanisms. The view is the output of a query and is akin to a virtual table or stored query. Views are dynamically constructed that cause the data that are presented can be custom-made for users based on their rights and privileges.

5.2.4 Interface design

To apply interface design considerations when building the software, such as the following:

a) **User interface** – to support the security model and act as the mediating program. User interface design should assure disclosure protection. Masking of sensitive information, such as a password or credit card number by displaying asterisks on the screen, is an example of a secure user interface that assures confidentiality. A database view can also be said to be an example of a restricted user interface.
b) **Application Programming Interfaces (API)** – to communicate from one software component with another or for the software to interact with the underlying operating system. The interfaces are usually made available in a library, and it may include specifications for routines, data structures, object classes, and variables.
c) **Security Management Interfaces (SMI)** – to configure and manage the security of the software itself. These are administrative interfaces with high levels of privilege. SMI is used for user-

provisioning tasks such as adding users, deleting users, enabling or disabling user accounts, as well as granting rights and privileges to roles, changing security settings, configuring audit log settings, and audit trails, exception logging, etc.

d) **Out-of-Band interface** – to allow an administrator to connect to a computer that is in an inactive or shutdown state.

e) **Log interfaces** – to log on or off in different environments (e.g., development, test, production, etc.), which is a crucial component of auditing and when designing software for auditing.

5.2.5 Interconnectivity

To explicitly design the upstream and downstream compatibility of software. This is important when it comes to delegation of trust, single sign-on (SSO), token-based authentication, and cryptographic key sharing between applications. Upstream and downstream compatibility of software should be explicitly designed.

In most mobile applications, the protection of the data on the client is left up to the application itself, and so the applications should be designed to avoid storing any sensitive information on the application's sandboxed environment itself. The publishers and third-party APIs that provide cryptographic services (encryption and decryption) may have to be considered to ensure confidentiality.

When data is stored in a location on the network, the network-attached storage (NAS) device should be protected as well. Only authenticated and authorized connections to the NAS should be designed, and if access is architected to be restricted by an IP range are designed, special considerations to IP spoofing threats need to be given.

5.3 Threat modeling

Objective

To identify, quantify, and address the security risks associated with the software.

Implementation guide

5.3.1 Step 1: Decompose the software

To obtain an understanding of the software and how it interacts with external entities. This involves creating use-cases to understand how the application is used, identifying entry points to see where a potential attacker could interact with the application, identifying assets, i.e., items/areas that the attacker would be interested in, and identifying trust levels which represent the access rights that the software will grant to external entities.Items to consider when decomposing the software includes:

a) external dependencies
b) entry points (or attack vectors)
c) assets
d) determining the attack surface by analyzing the inputs (browser input, cookies, property files, external processes, data feeds, service responses, flat files, command line parameters, environment variables), data flows and transactions
e) trust levels
f) data flow analysis
g) transaction analysis (areas covered are data/input validation of data from all untrusted sources, authentication, session management, authorization, cryptography (data at rest and in transit), error handling /information leakage, logging /auditing)
h) data flow diagrams

5.3.2 Step 2: Determine and rank threats

To identify threats using a threat categorization methodology. The example of threat categorization is such as STRIDE[4] can be used to defines threat categories with regards to the attacker's goals, such as auditing and logging, authentication, authorization, configuration management, data protection in storage, and transit, data validation, and exception management.

To rank the threat can use the DREAD[5] threat-risk ranking model, methodology to rank the risk of the threat is to calculate the average of numeric values assigned to risk ranking categories.

5.3.3 Step 3: Determine countermeasures and mitigation.

To mitigate the vulnerability with the implementation of a countermeasure of protection against a threat. Such countermeasures can be identified using threat-countermeasure mapping lists. Once a risk ranking is assigned to the threats, it is possible to sort threats from the highest to the lowest risk, and prioritize the mitigation effort, such as by responding to such threats by applying the identified countermeasures.

Examples of threat modeling can be referred in Annex E.

6 Phase 3: Security development

Objectives

a) To apply technology and process aspects of writing secure code.
b) To ensure the security control defined in security requirements should be put in place (in the code).
c) To ensure threats identified from threat modeling should be avoided during coding.

There are three (3) types of security tasks, which include i) common software vulnerabilities and controls; ii) secure software processes; iii) securing build environments, to accomplish the security development phase. The details will be provided in the next sub-section:

6.1 Common software vulnerabilities and controls

Objectives

a) To identify the most common vulnerabilities that result from insecure coding.
b) To apply security controls that should be put in place (in the code) to resist and frustrate the actions of threat agents.

Implementation guide

6.1.1 Vulnerability databases

To discover the vulnerability databases or repositories of known vulnerabilities that have been found to be the result of deficiencies and defects in implemented software (e.g., flaws and bugs). The most common software security vulnerabilities and risks have been listed in the SSDLC checklist to ease the developer during development. However, the organization can refer to the current list of software vulnerabilities found all throughout the software development industry, such as OWASP Top 10 Project or the Common Weakness Enumeration, CWE/25 (project maintained by MITRE and partnered with SANS Institute).

[4] STRIDE means Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege
[5] DREAD means Damage, Reproducibility, Exploitability, Affected Users, and Discoverability

6.1.2 Defensive coding practices

To recognize, evaluate, and reduce the attack surface of the software code. This is because the attack surface has potentially increased every time a single line of code is written. Some examples of attack surface reduction related to code are:

a) reducing the amount of code and services that are executed by default.
b) reducing the volume of code that can be accessed by untrusted users.
c) limiting the damage when the code is exploited.
   The most common defensive coding practices and techniques have been listed in the SSDLC checklist to ease the developer during development.

6.2 Secure software processes

Objective

To assure the security of software by conducted certain processes as an addition to writing secure code.

Implementation guide

6.2.1 Source code versioning

To ensure that the development team is working with the correct version of code. Gives the ability to roll back to a previous version should there be a need to. Additionally, it provides the ability to track ownership and changes of code, and for updated version purpose.

6.2.2 Code analysis

To perform an automated process of inspecting the code for quality and weaknesses that can be exploited. It is primarily accomplished by two means; static and dynamic.

Static code analysis involves the inspection of the code without executing the code (or software program). This analysis is normally done by a third party.

Dynamic code analysis is the inspection of the code when it is being executed (run as a program). This analysis is normally done by the developer.

6.2.3 Code review

To perform a manually systematic evaluation of the source code with the goal of finding out syntax issues and weaknesses in the code that can impact the performance and security of the software. This is normally done among the developer. Semantic issues such as business logic and design flaws are usually not detected in a code review, but a code review can be used to validate the threat model generated in the design phase of the software development project.

6.2.4 Developer testing

To perform the developers' intentional and systematic employment of testing tools and techniques in order to create testable and maintainable software with few defects as possible. Developer testing requires a structured approach and mastery of several core competencies, of which understanding testability drivers, fundamental testing techniques and unit testing are the most important.

The common types of tests that developers can write for applications are:

a) **Unit tests**
To perform the execution of a section of code or small program which is tested in isolation from the complete software. Tested in isolation means not calling the implementation of code not under test, e.g., database, web service calls, or other code dependencies. The concept of isolation is why mocking frameworks are commonly used for unit tests. Developers would write these during the development of a feature.

b) **Integration tests**
To perform the combined execution sections of code. In these types of tests, you would hit the database, make web service calls, or call other code dependencies. Developers would write these during the development of a feature.

c) **Regression tests**
To perform the repetition of previously executed test cases for the purpose of finding defects in software that previously passed the same set of tests. Such tests would commonly be used before shipping code to a new environment or as part of a build process. It's common to see tools such as selenium used to write these types of tests, where a web browser would be launched and user input automated. Human testers can also perform regression tests by using an application directly.

d) **Software tests**
To perform the execution of the software in its final configuration, including integration with other software and systems. It tests for security, performance, resource loss, timing problems, and other issues that can't be tested at lower levels of integration. As with regression testing, automated tools such as selenium can be used for this process as well as human testers.

6.3 Securing build environments

Objectives

a) To protect the source code from being accessed by an unrelated person with the project during the development phase.

b) To ensure the integrity of the source code developed.

Implementation guide

6.3.1 Physically securing access to the software's that building code.

6.3.2 Using access control lists (ACLs)

Access control lists (ACLs) that prevent access to unauthorizedusers.

6.3.3 Using the version control software

Version control software to assure that the code built is of the right version.

6.3.4 Build automation

Build automation is the process of scripting or automating the tasks that are involved in the build process. It takes the manual activities performed by the build team members daily and automates them. Some of these build activities include compiling source code into machine code, packaging dependencies, deployment, and installation. When build scripts are used to build automation processes, it is important to make sure that security controls and checks are not circumvented, when using these build scripts.

6.3.5 Code signing routine

Code signing is to ensure the integrity of the source code being developed.

7 Phase 4: Security testing

Objectives

a) To validate and verify the functionality and security of software using quality assurance testing
b) To ensure the code developed runs as intended

There are two (2) types of security tasks, which are i) attack surface validation, ii) test data management, to accomplish the security testing phase. The details will be provided in the next sub-section:

7.1 Attack surface validation

Objective

To verify the presence and effectiveness of the security controls that are designed and implemented in the software.

Implementation guide

7.1.1 Post-development testing

To ensure the correctness of developed software, this step will execute code analysis, which is specifically on dynamic code analysis, is the inspection of the code when it is being executed (run as a program).

7.1.2 Perform security testing using security testing methods

Methods or approaches used to accomplish security testing are as below:

a) **White box testing** – to test the structure of software which performed based on the knowledge of how the software is designed and implemented. It is broadly known as a *full knowledge* assessment because the tester has complete knowledge of the software. It can be used to test both the use case (intended behavior) as well as the misuse case (unintended behavior) of the software and can be conducted at any time post-development of code, although it is best advised to do so while conducting unit tests.
Data/information flow, control flow, interfaces, trust boundaries (entry and exit points), configuration, error handling, etc. are methodically and structurally analyzed for security.

b) **Black box testing** – to test the behavior of software, also known as *zero knowledge* assessment, because the tester has very limited to no knowledge of the internal working of the software being tested. Architectural or design documents, configuration information or files, use, and misuse cases or the source code of the software is not available to or known by the testing team.
Black box testing is performed using different tools. The common methodologies by which black box testing is accomplished with tools are fuzzing, scanning, and penetration testing.

c) **Cryptographic validation testing** – to ensure the applications that employ cryptographic mechanisms are using solid cryptographic algorithms and best practices for key management. Some of these requirements are available from Federal Information Processing Standards (FIPS) 140-2 Special Publications and MySEAL (National Trusted Cryptographic Algorithm List) website for all standard algorithms such as AES, RSA, DSA, etc. whenever applicable.

7.1.3 Perform software security testing for quality assurance

To apply with different types of tests and how they can be performed to attest to the security of code that is developed in the development phase of the SSDLC.

For software revisions, regression testing should be conducted, and for all versions, new or revisions, the following security tests should be performed, if applicable, to validate the strength of the security controls. Using a categorized list of threats as a template of security testing is effective in ensuring comprehensive coverage of the varied threats to software. Ideally, the same threat list that was used when threat modeling the software will be the threats list that is used for conducting security tests as well. This way, security testing can be used to validate the threat model.

The security tests are as follows:
a) Testing for input validation
b) Testing for injection flaws controls
c) Testing for scripting attacks controls
d) Testing for non-repudiation controls
e) Testing for spoofing controls
f) Testing for error and exception handling controls (failure testing)
g) Testing for privileges escalations controls
h) Anti-reversing protection testing
i) Stress testing

7.2 Test data management

Objective

To identify input test data and data that is expected to be output after normal operations of the software.

Implementation guide

7.2.1 Identify output test data to confirm software requirements

To identifying expected output test data helps to confirm if the software is meeting the requirements. The quality of test data is directly related to the quality of the test itself, and so test data needs to be managed. Production data should never be imported into and processed in test environments. It is advisable to use dummy data by creating it from scratch in the test or simulated environment.

7.2.2 Apply testing with synthetic transactions

To perform transactions that serve no business value that is related to the dummy data. Synthetic transactions can be passive or active. Passive synthetic transactions are not stored (or maintained) and do not have any residual impact on the software itself. However, if the query for finding orders of a 'dummy' customer is processed and stored within the software application, it would constitute an active synthetic transaction. The usage of active synthetic transactions requires one to give attention to setting up the data and environment in such a manner that it does not impact the production environment.

7.2.3 Test data management solutions

It can aid in the creation of referentially whole data subsets of production data. To reduce some of the concerns that come with the creation of quality test data and its management in test environments. These solutions automatically discover data relationships by analyzing and capturing table attributes. Ensure that the extraction rules consider the storage space that is available in the test environment, so that the extraction process does not end up extracting a large subset of data that cannot be imported into the test environment, due to size limitations.

7.2.4 Defect reporting and tracking

To report on defects/flaws and then track the coding bugs, design flaws, behavioral anomalies (logic flaws), errors, faults, and vulnerabilities of the software. Reporting defects should be comprehensive and detailed enough to provide the software development teams the information that is necessary to determine the root cause of the issue so that they can address it.

8 Phase 5: Security deployment

Objectives

a) To ensure completion of operation plan and application documentation
b) To ensure management approval and risk acceptance for deployment
c) To ensure the application is meets its functionality and secured
d) To ensure a secured environment and configuration for deployment

There are four (4) types of security tasks, which are i) software acceptance considerations; ii) verification and validation (V&V); iii) certification and accreditation (C&A); iv) installation; to accomplish the security deployment phase. The details will be provided in the next sub-section:

8.1 Software acceptance considerations

Objectives

a) To ensure application documentation and operation plan is ready.
b) To obtain approval and risk acceptance.

Implementation guide

8.1.1 Completion criteria

To validate and verified all functional and security requirements completed as expected. Completion criteria for functionality and software security with explicit milestones should be defined well in advance. Some examples of security-related milestones include, but are not limited to, the following:

a) generation of the security requirements besides functional requirements in the requirement phase;
b) completion of the threat model during the design phase;
c) review and sign-off on the security architecture at the end of the design phase;
d) review of code for security vulnerabilities after the development phases;
e) completion of security testing at the end of the application testing phase; and
f) completion of documentation before the deployment phase commences.

8.1.2 Change management

To ensure the process in place to handle change requests. Change management is a subset of configuration management. Changes to the computing environment and redesign of the security architecture can potentially introduce new security vulnerabilities, thereby increasing risk. Necessary support queues and processes for the software that is to be deployed/released should be established.

8.1.3 Approval to deploy or release

To ensure all the required authorities signed off. Without approvals, no change should be allowed to the production computing environment. Before any new installation of software, risk analysis needs to be conducted, and the residual risk determined. The results of the risk analysis, along with the steps taken to address it (mitigate or accept) should be communicated to the business owner. The authorizing official should be informed of the residual risk. The approval or rejection to deploy/release should include recommendations and support from the security team. Ultimately it is the authorizing official (AO) who is responsible for change approvals.

8.1.4 Risk acceptance and exception policy

To ensure the residual risk acceptable and/or tracked as an exception if it is not within the threshold. The risk that remains after the implementation of security controls (residual risk) needs to be determined first. The best option to address total risk is to mitigate it so that the residual risk falls below the business defined threshold, in which case the residual risk can be accepted. Risk should be accepted by the business owner and not by officials in the IT department.

8.1.5 Documentation of software

To ensure all necessary documentation are in place. Documenting what the software is supposed to do, how it is architected, how it is to be installed, what configuration settings need to be preset, how to use it, and administer it is extremely important for effective, secure, and continued use of the software. Some of the primary objectives for documentation are to make the software deployment process easy and repeatable and to ensure that operations are not disrupted, and the impact upon changes to the software is understood.

8.2 Verification and validation (V&V)

Objective

To ensure application functionality and secured for the production environment.

Implementation guide

8.2.1 Reviews

To conduct a review at the end of each phase for ensuring that the software performs as expected and meets business specifications. This can be done informally or formally.

8.2.2 Testing

To demonstrate that the software truly meets the requirements and determine any variances or deviations from what is expected using the actual results from the test. The different kinds of tests that are conducted as part of V&V are:

a) Error detection tests – unit and component level testing. Errors may be flaws (design issues) or bugs (code issues).
b) Acceptance tests – used to demonstrate if the software is ready for its intended use or not. Software that is deemed ready should not only be validated for all functional requirements but also be validated to ensure that meets assurance (security) requirements.
c) Independent (Third Party) tests – testing of software functionality and assurance is the process in which the software is reviewed, verified, and validated by someone other than the developer of the software. This is commonly also referred to as Independent Verification & Validation.

8.3 Certification and accreditation (C&A)

Objectives

a) To obtain technical verification of application functionality.
b) To obtain overall acceptance for deployment into the production environment.

Implementation guide

8.3.1 Obtain certification

To achieve the certification for the technical verification of the software functional and assurance levels. Security certification considers the software in the operational environment.

At the minimum, it will include assurance evaluation of the following:

a) User rights, privileges, and profile management
b) The sensitivity of data and application and appropriate controls
c) Configurations of software, facility, and locations
d) Interconnectivity and dependencies, and
e) Operational security mode

Organizations can obtain Common Criteria certification for ICT products through Malaysian Common Criteria Evaluation and Certification (MyCC) Scheme and also Malaysia Trustmark certification for e-business in the private sector, which are provided by the CyberSecurity Malaysia (CSM).

8.3.2 Obtain accreditation

To achieve the accreditation of management's formal acceptance, need verification from participant/target users regarding the software after an understanding of the risks to that software rating in the computing environment. It is the management's official decision to operate software in the operational security mode for a stated period and is the formal acceptance of the identified risk associated with operating the software.

8.4 Installation

Objective

To securing application production environment and configuration

Implementation guide

8.4.1 Hardening

To lock down the software to the most restrictive level so that it is secure. This minimum (or most restrictive) security levels are usually published as a baseline that all software in the computing environment should comply with. This baseline is commonly referred to as a minimum baseline created based on the usage of the operating system.

It is important to harden the host operating system by using baseline, updates, and patches. It is also critically important to harden the applications and software that run on top of these operating systems. Hardening of software involves setting the necessary and correct configuration settings and architecting the software to be secure by default.

8.4.2 Environment configuration

To ensure that the needed parameters required for the software to run are appropriately configured by using pre-installation checklists. However, since it is not always possible to statically identify dynamic issues, checklists provide no guarantee that the software will function without violating the security principles with which it was designed and built.

Therefore, it is crucial to ensure that the development and test environment match the configuration makeup of the production environment and simulation testing identically emulates the settings (including the restrictive settings) of the environment in which the software will be deployed post acceptance.

8.4.3 Release management

To ensure the properly released of software into the operating computing environment once hardware and software resources are hardened and the environment configured for secure operations.

Release management is the process of ensuring that all changes that are made to the computing environment are planned, documented, thoroughly tested, and deployed with the least privilege without negatively impacting any existing business operations, customers, end-users, or user support teams.

8.4.4 Bootstrapping and secure startup

To determine that the software startup processes do not in any way adversely impact the confidentiality, integrity or availability of the software upon the installation of software. The Power-on self-test (POST) is the first step in an Initial Program Load (IPL) and is an event that needs to be protected from being tampered so that the Trusted Computing Base (TCB) is maintained.

9 Phase 6: Security maintenance

Objectives

a) To monitor and guarantee that the software will continue to function in a reliable, resilient, and recoverable manner.
b) To identify the software and conditions under which software needs to be disposed or replaced.

There are five (5) types of security tasks, which are i) operations, monitor and maintenance; ii) incident management; iii) problem management; iv) change management; v) disposal; to accomplish the security maintenance phase. The details will be provided in the next sub-section:

9.1 Operations, monitor and maintenance

Objectives

a) To provide services to the business or end-users for the needs of software operations and maintenance.
b) To ensure assurance aspects of reliable, resilient, and recoverable processing of the software.

Implementation guide

9.1.1 Carry out the operations security

To ensure the operations security is about staying secure or keeping the resiliency levels of the software above the acceptable risk levels. It is the assurance that the software will continue to function as is expected to in a reliable fashion for the business without compromising its state of security by monitoring, managing, and applying the needed controls to protect resources (assets).

Different types of operations security controls are as follows:

a) **Detective Controls** are those that can be used to build historical evidence of user and software/process actions.
b) **Preventive Controls** are those who make the success of the attacker difficult as its goal is to prevent the attack actively or proactively.
c) **Deterrent Controls** are those, which don't necessarily prevent an attack, nor are they merely passive in nature.
d) **Corrective Controls** are those who aim to provide the recoverability of software assurance.
e) **Compensating Controls** are those controls that should be implemented when the prescribed software controls as mandated by a security policy or requirement cannot be met due to legitimate technical or documented business constraints.

9.1.2 Continuous monitoring

To perform monitoring to any system, software, or processes. It is important to first determine the monitoring requirements before implementing a monitoring solution. Monitoring requirements need to be solicited from the business early in the software development life cycle.

Along with the requirements, associated metrics that measure actual performance and operations should be identified and documented. Continuous security tests should be conducted at a planned interval or according to changes based on needs or requirements.

9.1.3 Audit for monitoring

To provide an independent review and examination of software records and activities. An audit is conducted by an auditor whose responsibilities include the selection of events to be audited on the software, setting up of the audit flags which enable the recording of those events and analyzing the trail of audit events. Audits should be conducted periodically and can give insight into the presence and effectiveness of security and privacy controls.

9.2 Incident management

Objective

To detect and monitor the incidence of a security breach.

Implementation guide

9.2.1 Determine events, alerts, and incidents

To determine if a security incident has truly occurred or not, firstly, to define what constitutes an incident. Failure to do so can lead to potential misclassification of events and alerts as incidents, and this could be costly.

a) Event is any action that is directed at an object which attempts to change the state of the object
b) Alerts are flagged events that need to be scrutinized further to determine if the event occurrence is an incident.
c) Alerts can be categorized into incidents, and adverse events can be categorized into security incidents if they violate or threaten to violate the security policy of the network, system, or software applications.

9.2.2 Identify types of incidents

To identify several types of incidents and the main security incidents, there are include the following:

a) **Denial of Service (DoS)** – is an attack that prevents or impairs an authorized user from using the network, systems, or software applications by exhausting resources.
b) **Malicious Code** – has to do with code-based malicious entities such as viruses, worms, and Trojan horses that can successfully infect a host.
c) **Unauthorized Access** – access control related incidents refer to those wherein a person gains logical or physical access to the network, system or software applications, data, or any other IT resource, without being granted the explicit rights to do so.
d) **Inappropriate Usage** – comprise of those in which a person violates the acceptable use of software resources or company policies.
e) **Multiple Component** – are those which encompass two or more incidents.

9.2.3 Incident response process

To enables and assure operations security of the organization and remain in business. The major phases of the incident response process involve are preparation, detection and analysis, containment, eradication and recovery, and post-incident analysis.

Brief explanations are as follows:

a) **Preparation** – the organization aims to limit the number of incidents by implementing controls that were deemed necessary from the initial risk assessments.
b) **Detection and analysis** – the organization able to detect security breaches will be aware of incidents before, or when they occur, and if the incident is disruptive and unknown, appropriate actions should be taken.
c) **Containment, eradication, and recovery** – upon the detection and validation of a security incident, the first course of action that needs to be taken is that the incident is contained to limit any further damage or additional risks. Containment is the steps necessary to remove and eliminate components of the incident should be undertaken. Eradication steps may be performed during recovery to enforce that any fixes or steps to eradicate the incident are steps only after appropriate authorization is granted. Recovery mechanisms aim to restore the resource (network, system or software application) back to its normal working state.
d) **Post-incident analysis** – This is a lesson learned activities that produce a set of objective and subjective data regarding each incident.

9.3 Problem management

Objective

To determine and eliminate the root cause of the problem (unknown incident) and to improves the service that the software provides to the business so that it will not be repeated.

Implementation guide

9.3.1 Incident notification

To identify and notify the unknown incident, a problem happened to the software.

9.3.2 Root cause analysis

To determine the reason for the problem by implementing the root causes analysis (RCA) steps. RCA is performed to determine ‘Why’ the problem occurred, repeatedly and systematically, until there are no more reasons (or causes) that can be answered.

9.3.3 Solution determination

To determine the solution, which is temporary workarounds or permanent fixes to be implemented.

9.3.4 Request for change

To include workarounds (to support existing users), known errors, update problem information, management information, and request for changes.

9.3.5 Implement solution

To implement the identified solution after initiating a request for change.

9.3.5 Monitor and report

To monitor the solution to the problem by preparing the report.

9.4 Change management

Objectives

a) To determine the recovery and resolution of the problem after the root cause is identified.
b) To track the vulnerability and monitor the problem resolution to ensure that it was effective and that the problem does not happen again.

Implementation guide

9.4.1 Patch and vulnerability management

To update or fix existing software with patches, which is a piece of code that is used so that the software is not susceptible to any bugs. Patching is the process of applying these updates or fixes. Patches can be used to address security problems in software or simply provide additional functionality. Patching is a subset of hardening.

Some of the necessary steps that need to be taken as part of the patching process include:

a) Notifying the users of the software or systems about the patch
b) Testing the patch in a simulated environment so that there is no backward compatibility or dependencies (upstream or downstream) issues.
c) Documenting the change along with the rollback plan. The estimated time to complete the installation of the patch, criteria to determine the success of the patch, and the rollback plan should be included as part of the documentation.
d) Identifying maintenance windows or the time when the patch is to be installed should be performed. The best time to install the patch is when there is minimal disruption to the normal operations of the business, but with most software operating in a global economy setting, identifying the best time for patch application is a challenge today.
e) Installing the patch
f) Testing the patch post-installation in the production environment is also necessary. Sometimes a reboot or restart of the software where the patch was installed is necessary to read or load newer configuration settings and fixes to be applied. Validation of backward compatibility and dependencies also needs to be conducted.
g) Validating that the patch did not regress the state of security and that it leaves the systems and software in compliance with the minimum-security baseline.
h) Monitoring the patched systems so that there are no unexpected side effects upon the installation of the patch.
i) Conducting post-mortem analysis in case the patch had to be rolled back and using the lessons learned to prevent future issues. If the patch was successful, the minimum-security baseline needs

to be updated accordingly.

j) Conducting virtual patching to cover the legacy system. Virtual patching refers to establishing an immediate security policy enforcement layer that prevents the exploitation of a known vulnerability, without modifying the application's source code, binary changes, or restarting the application. Virtual patching can be used to add temporary protection, giving the development teams time to deploy physical patches according to their own update schedules. Virtual patches can also be used permanently for legacy systems that may not be patchable.

9.4.2 Backups, recovery and archiving

To assure uninterrupted business operations and continuity. The continuity of business without disruptions is an important factor in secure software operations. Not only should the data be available but also the system itself.

In addition to regularly scheduled backups, when patches and software updates are made, it is advisable to perform a full backup of the software that is being changed. It is also crucial that the integrity and restorability of the backup (especially if it is data backups) are verified.

Additionally, when software has been infected by malware such as Trojan horses and spyware, the only option left for assuring continued integrity may be to completely format and reinstall the software accompanied by restoring the data from a secure, trusted and verified backup.

Archives can come in handy in user support, especially for past customers. The integrity of archives can be accomplished using hashing, and proper key management needs to be in place to make the cryptographically protected data in archives usable upon recovery.

9.5 Disposal

Objective

To identify software that not operational to be disposed to reduce the residual risk

Implementation guide

9.5.1 End-of-Life policies

To perform risk management activities for system components that will be disposed or replaced to ensure that the hardware and software are properly disposed.

In order to manage risk during the disposal phase, it is essential that we have an End-of-Life (EOL) policy to be developed and followed. The EOL policy is the first requirement in the secure disposal of software and its related data and documents.

9.5.2 Sun-setting criteria

To dispose or replace a product such as software or the hardware on which the software runs by following the sun-setting criteria as guidance.

9.5.3 Sun-setting processes

To dispose the software related technologies that are deemed insecure, but which have no means to mitigate the risk to the acceptable levels of the organization, should be sun-set as soon as it is possible. In compliance with the organization’s EOL policy, appropriate EOL processes should be established. EOL processes are the series of technical and business milestones and activities, which, when complete, make the hardware or software obsolete and no longer produced, sold, improved, repaired, maintained, or supported.

9.5.4 Information disposal and media sanitization

To ensure that software assurance is maintained by following the software disposal steps, an important part of that process is to also ensure that the media that stored the information is also sanitized or destroyed appropriately. Sanitization is the process of removing information from media such that data recovery and disclosure is not possible. It also includes the removal of classified labels, marking, and activity logs related to the information.

10 SSDLC Checklist

The provided guideline of Secure Software Development Life Cycle (SSDLC) has determined several security tasks that involve in each phase. Subsequent to that, checklists are used as a quick guide for the target audience. These checklists assisted the audience in tracking and monitoring the security tasks apply to the development of secure software that following the SSDLC phases. The listed checklists, as in the following sub-sections, are divided according to the phases in the SSDLC.

10.1 Phase 1: Security Requirements (4)

No.	Security Tasks		Action (Done ☑ / Not ☒)
4.1	**Sources for security requirement**		
	4.1.1	Identify core security requirements	
		Confidential requirements	
		Integrity requirements	
		Availability requirements	
		Authentication requirements	
		Authorization requirements	
		Accountability requirements	
	4.1.2	Identify general requirements	
		Session Management requirements	
		Errors & Exceptions Management requirements	
		Configuration Parameters Management requirements	
	4.1.3	Identify operational requirements	
		Deployment environment requirements	
		Archiving requirements	
		Anti-piracy requirements	
	4.1.4	Identify other requirements	
		Sequencing and timing requirements	
		International requirements	
		Procurement requirements	
4.2	**Data classification**		
	4.2.1	Types of data	
	4.2.2	Labeling the data	
	4.2.3	Data ownership and roles	
	4.2.4	Data lifecycle management (DLM)	
	4.2.5	Privacy requirements	
		Data anonymization	
		Disposition	
		Security models	
		Pseudonymization	
4.3	**Use case and misuse case modeling**		
	4.3.1	Analyze the use case scenarios	
	4.3.2	Analyze the misuse case scenarios	
	4.3.3	Creating attack model	
	4.3.4	Select mitigation control	

4.4	**Risk management**		
	4.4.1	Risk assessment	
	4.4.2	Risk mitigation	
		Risk mitigation options	
		Risk mitigation strategy	
		Approach for control implementation	
		Control categories	
		Cost-benefit analysis	
		Residual risk	
	4.4.3	Evaluation and assessment	

10.2 Phase 2: Secure Design (5)

No.	Security Tasks		Action (Done ☑ / Not ☒)
5.1	**Core Security Design Considerations**		
	5.1.1	Confidentiality design	
	5.1.2	Integrity design	
	5.1.3	Availability design	
	5.1.4	Authentication design	
	5.1.5	Authorization design	
	5.1.6	Accountability design	
5.2	**Additional Design Considerations**		
	5.2.1	Programming languages	
	5.2.2	Data type, format, range and length	
	5.2.3	Database security	
	5.2.4	Interface design	
	5.2.5	Interconnectivity	
5.3	**Threat modeling**		
	5.3.1	Step 1: Decompose the software	
	5.3.2	Step 2: Determine and rank threats	
	5.3.3	Step 3: Determine countermeasures and mitigation.	

10.3 Phase 3: Security Development (6)

No.	Security Tasks		Action (Done ☑ / Not ☒)
6.1	**Common software vulnerabilities and controls**		
	6.1.1	Vulnerability databases	
		The most common software security vulnerabilities and risks are:	
		buffer overflow	
		stack overflow	
		heap overflow	
		injection flaws	
		broken authentication and session management	
		cross-site scripting (XSS)	
		insecure direct object references	
		security misconfiguration	
		sensitive data exposure	
		missing function level checks	
		Cross-Site Request Forgery (CSRF)	
		using known vulnerable components	
		invalidated redirects and forwards	
		file attacks	
		race condition	
		side channel attacks	
	6.1.2	Defensive Coding Practices	
		The most common defensive coding practices and techniques are:	
		input validation	
		canonicalization	
		sanitization	
		error handling	
		safe application programming interfaces (API)	
		memory management	
		exception management	
		session management	
		configuration parameters management	
		secure startup	
		cryptography	
		concurrency	
		tokenization	

		sandboxing	
		anti-tampering	
6.2	**Secure software processes**		
	6.2.1	Source code versioning	
	6.2.2	Code analysis	
	6.2.3	Code review	
	6.2.4	Developer testing	
		Unit tests	
		Integration tests	
		Regression tests	
		Software tests	
6.3	**Securing build environments**		
	6.3.1	Physically securing access to the systems that build code.	
	6.3.2	Using access control lists (ACLs) that prevent access to unauthorized users.	
	6.3.3	Using version control software to assure that the code built is of the right version.	
	6.3.4	Build automation is the process of scripting or automating the tasks that are involved in the build process.	
	6.3.5	Code signing routine	

10.4 Phase 4: Security Testing (7)

No.	Security Activities		Action (Done ☑ / Not ☒)
7.1	**Attack surface validation**		
	7.1.1	Post-development testing	
	7.1.2	Perform security testing using security testing methods	
		White box testing	
		Black box testing	
		Cryptographic validation testing	
	7.1.3	Perform software security testing for quality assurance	
7.2	**Test data management**		
	7.2.1	Identify output test data to confirm software requirements	
	7.2.2	Apply testing with synthetic transactions	
	7.2.3	Test data management solutions can aid in the creation of referentially whole data subsets of production data.	
	7.2.4	Defect reporting and tracking	

10.5 Phase 5: Security Deployment (8)

No.	Security Tasks		Action (Done ☑ / Not ☒)
8.1	**Software acceptance considerations**		
	8.1.1	Completion criteria	
	8.1.2	Change management	
	8.1.3	Approval to deploy or release	
	8.1.4	Risk acceptance and exception policy ␟	
	8.1.5	Documentation of software	
8.2	**Verification and validation (V&V)**		
	8.2.1	Reviews	
	8.2.2	Testing	
8.3	**Certification and accreditation (C&A)**		
	8.3.1	Obtain certification.	
	8.3.2	Obtain accreditation.	
8.4	**Installation**		
	8.4.1	Hardening	
	8.4.2	Environment configuration	
	8.4.3	Release management	
	8.4.4	Bootstrapping and secure startup	

10.6 Phase 6: Security Maintenance (9)

No.	Security Tasks		Action (Done ☑ / Not ☒)
9.1	**Operations, monitor and maintenance**		
	9.1.1	Carry out the operations security.	
	9.1.2	Continuous monitoring	
	9.1.3	Audit for monitoring	
9.2	**Incident management**		
	9.2.1	Determine events, alerts, and incidents	
	9.2.2	Identify types of incidents	
	9.2.3	Incident response process	
	9.2.4	Problem management	
9.3	**Problem management**		
	9.3.1	Incident notification	
	9.3.2	Root cause analysis	
	9.3.3	Solution determination	
	9.3.4	Request for change	

	9.3.5	Implement solution	
	9.3.6	Monitor and report	
9.4	**Change management**		
	9.4.1	Patch and vulnerability management	
	9.4.2	Backups, recovery and archiving	
9.5	**Disposal**		
	9.5.1	End-of-Life policies	
	9.5.2	Sun-setting criteria	
	9.5.3	Sun-setting processes	
	9.5.4	Information disposal and media sanitization	

Annex A

Reference on Examples for Secure Software Architectures

1. Figure A-1 below shows the example of secure software architecture.

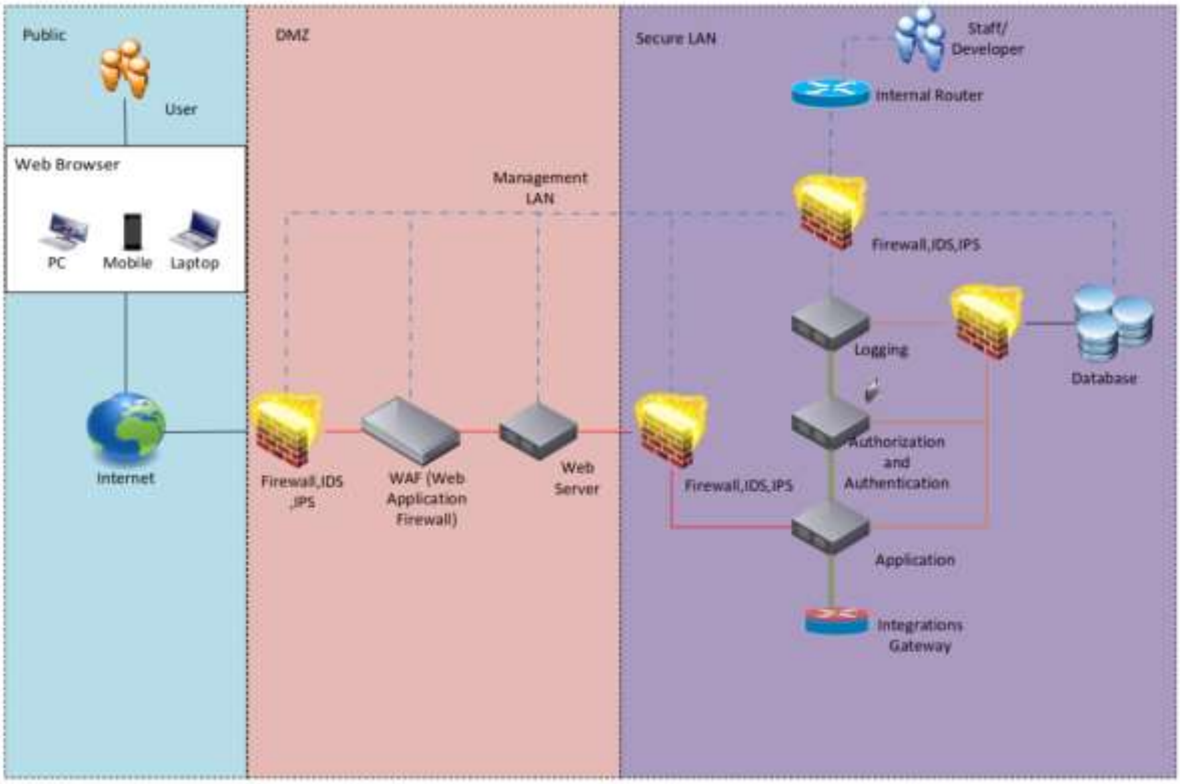

Figure A-1: Example of Secure Software Architecture

The example given in Figure A-1 is a sample of suggestions for secure software architecture, which basically applied to a secure software system. However, organizations can make their choice and decision on applying as many as possible controls according to the estimation cost of the development project.

Referring to Figure A-1, the requirements of controls are needed to assure the security of the proposed secure software by providing three (3) main segmentations, which are public, demilitarized zone (DMZ), and secure local area network (LAN). The DMZ segmentation is for the purpose of controls, filtering or monitoring conducted by the firewall, intrusion detection system (IDS) or intrusion prevention system (IPS). The management LAN is intended for staff and developers who are authorized to access the devices or applications. Furthermore, for secure LAN segmentation and accessing the databases are controlled, filtered, or monitored by the firewall, IDS or IPS.

2. Figure A-2 below shows the example of secure software architecture with protection controls.

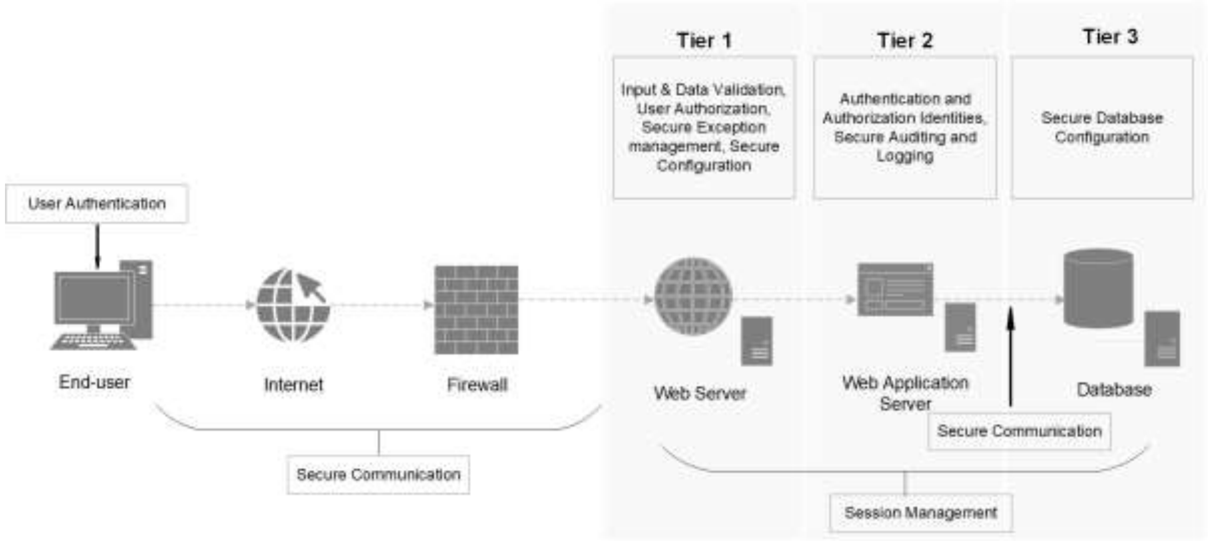
Tier 1
Tier 2
Tier 3
Input & Data Validation, User Authorization, Secure Exception management, Secure Configuration
Authentication and Authorization Identities, Secure Auditing and Logging
Secure Database Configuration
User Authentication
End-user
Internet
Firewall
Web Server
Web Application Server
Database
Secure Communication
Secure Communication
Session Management

Figure A-2: Example of Secure Software Architecture with Protection Controls

As refer to Figure A-2, the protection controls involve are started from the end-user site, which is the authentication that has been provided in order to access the secure software. An authorized end-user will be in a secure communication and with the authorization can pass through the firewall.

There are more protection controls can be provided. The possible controls can be provided specifically in web server are input and data validation, user authorization, secure exception management, secure configuration, or more others. For the web application server, protection controls that can be provided are authentication and authorization identities, secure auditing and logging, or others. For databases have secure database configuration or others as the protection controls.

The example given in Figure A-2 is a sample suggestion, which the organizations can make their choice and decision on applying any possible controls based on the estimation cost of the development project.

Annex B

Example of Security Requirements

An application can be described using different tools, such as informal drawings, pictures, sketches etc. In developing applications, high-level risk tables can be used to help identifying security requirements. Table B-1 gives an example of core security requirements or security goals, presenting methods that can be used to achieve the protection or mitigation and some tools that may be used to help. Table B-2 provides the example of security measures and the associated security mechanism. Table B-3 presents an example of a security requirement table, suggesting the priority level. The content should be reviewed during the initial analysis and/or always when some modification is necessary into the software.

Table B-1 Core Security Requirements (Security Goals) and Associated Security Measures

Core security requirements	Associated security measures
Confidentiality	Access control; Physical protection; Security policy
Integrity	Access control; Non-repudiation; Physical protection; Attack detection
Availability	System recovery; Physical protection; Attack detection
Accountability	Non-repudiation; Attack detection

Table B-2 Security Measures and Associated Security Mechanisms

Security measures	Associated security mechanisms
Access control	Biometrics; Certificates; Multilevel security; Passwords and keys; Reference monitor; Registration; Time limits; User permissions; VPN
Security policy	Administrative privileges; Malware detection; Multilevel security; Reference monitor; Secure channels; Security session; Single access point; Time limits; User permissions; VPN
Non-repudiation	Administrative privileges; Logging and auditing; Reference monitor
Physical protection	Access cards; Alarms; Equipment tagging; Locks; Off-site storage; Secured rooms; Security personnel
System recovery	Backup and restoration; Configuration management; Connection service agreement; Disaster recovery; Off-site storage; Redundancy
Attack detection	Administrative privileges; Alarms; Incident response; Intrusion detection systems; Logging and auditing; Malware detection; Reference monitor
Boundary protection	DMZ (Demilitarized Zone); Firewalls; Proxies; Single access point; VPN

Table B-3 Security Requirements with Priority Level

Security type	Requirement description	Comments	Priority
Authentication	The system should have authentication measures at all the entry points, front panel, or inbound network connection.	To avoid unauthorized access	1
	The system should support Windows domain authentication, and when authenticate to AD (Active Directory), should support NTLMv2, as well as Kerberos protocol, and should avoid transmitting username/password on the wire when authentication to the AD.	Currently many systems use Single Sign-On (SSO) using Kerberos and Windows domain.	2
	The system should support Smartcard, USB token (two factors) authentication.	Improving the security using smartcard technologies and facilitating the physical access in case of using SSO	1
	The system should support Proximity card, magnetic card authentication.	To adequate some technologies as NFC (Near Field Communication), for example.	3
	The system should support authentication based on device local authority.	In case of physical access.	1
	The system should support authentication to external 3rd party authentication server and protect the user credential during authenticating to the external server.	Security network aspects should receive special attention.	1
	The system should support multiple authentication approaches at the same time.	It is necessary a strong monitoring and auditing.	1
	The system should support multiple login sessions concurrently, and sessions should be protected in relation to each other.	To avoid session hijacking.	1
	The system should support the outbound authentication as a Single Sign-On (SSO) schema.	Security network aspects should receive attention.	1
	The system should support multi-level system access.	Security issues as bypassing permissions to be mitigated.	1
	The user accounts should possess privileges within the application to perform their signing activities. However, the privileges should be limited.	Avoid an authorized user execute activities as another user.	1
	The system should ensure business user accounts cannot be administrator accounts and vice-versa.	Help to avoid attackers escalate user's accounts to access administrator's features.	1

	The system should ensure system level accounts have limited privileges.	Help avoiding attackers escalate user's accounts to access administrator's features.	1
	The system should ensure the database access is performed using parameterized store procedures to allow all table access to be revoked.	Apply database security principles.	1
	The system should avoid store sensitive static content in web-accessible directory paths. The content should be stored in non-accessible directories and proxy access to this content using a handler that will implement authorization, logging, and other security functions.	Avoiding attackers gain access to the directory paths and discover some important information.	1
Availability	The backup system should store the recover sources in a network system.	To help in case of failure or intruder action	1
	The system should do mirroring to allow data and software to be available in physically separated locals (separate site when the application is on the Web)	Help minimize the risk of one single point of failure.	3
	The system should apply high availability solution such as clustering.	Help minimizing the impact of potential system failures.	3
Integrity	The system should ensure all data provided by software has consistency (either create a new and valid state of data, or, return all data to its state before a transaction was started).	To avoid an authorized person or system alter data inadvertently or intentionally.	1
Auditing	The system should keep historical records (logging) of events and processes executed in or by an application.	Define more specific security loggings to allow recreating a clear picture of security events.	1
Non-Repudiation	The system should implement cryptographic methods such as generating digital signatures or digital fingerprinting.	Help the application and system avoiding repudiation.	1

Notes: Priority depicted with 1-High, 2-Medium, 3-Low.

Annex C

Reference on Procurement Requirements

In gathering requirements, organizations should have basic specifications needed and the idea of a structure for security software to be developed. Organizations should state the common security requirement for the security software to be developed.

The targets for the security software to be developed are as follows:

a) Risk reduction should focus on reducing vulnerabilities and minimize the severity of cyber-attacks.
b) Software assurance where organization should take it as a strategic initiative to promote integrity, security and reliability in software.
c) Procurement specification for control software where organization put initiative to develop procurement document for control software, including hardware and software.

The goal for preparing the procurement requirements is to ensure the security software that will be developed has the complete available security features. The organization is responsible for:

a) applying the security requirements to the secure software project and allocating financial, technical and human resources as required for meeting the security requirements of the project
b) ensuring that the security controls are tested and validated during the acceptance test phase
c) maintaining security controls throughout the life cycle of secure software.

In terms of the procurement specification to describe the general idea of the secure software to be developed, as shown in Figure A-1 (Refer Annex A), which presents the structure of secure software architecture in the web environment. This diagram shows the essential elements or components in the security software and their relationship with users and the Internet. The importance of this architecture is to:

a) to give guidance to the organization for their new secure software project development
b) to clearly define possible elements, involve such as assets and controls,
c) to define the scope of secure software to be developed an in-house software as presented in Figure A-1

As refer to Figure A-1, the organization can provide detailed specifications for as follows: identifying assets to determine how the software needs to be protected. Asset identification can be made through different viewpoints: the customer's, the software owner's, and the attackers. After identifying the assets, further analysis should be done on each to identify a priority on protection. This will ensure that the most valuable assets receive the most attention in threat mitigation. Assets include data, software, and hardware components and communication services.

In advance, before the processes of proposing specific controls to the components of the software, the organization should put attention to critical risk management. With that, during the risk analysis process organization should consider the following:

a) The threats which are likely to want to attack our software
b) The risks present in each component in the software environment
c) The kinds of vulnerabilities that might exist in each component, as well as the data flow
d) The business impact of such technical risks were they to be realized
e) The probability of such a risk being realized
f) Any feasible countermeasures that could be implemented at each tier, taking into account the full range of protection mechanisms available

The continuing steps are defining and exploring the available possible security controls that should be suggested by the organization to ensure security. This document provides guidelines for specific security tasks of each phase in Secure Software Development Life Cycle (SSDLC) will assists organizations in providing security controls to the secure software to be developed. By following the phases from early to the end will guide the organization to produce a complete procurement document.

The mechanism for the purpose of procurement and accreditation of a resilient system against malicious encompasses the entire life cycle of an asset. This procurement can be accomplished through tenders, quotes, and direct negotiations in accordance with current regulations. Steps involve are:

a) Identify Needs – Organization should identify needs before any procurement either through a vendor or inhouse
development.
b) Procurement Specifications – The procurement specifications should contain specific clauses regarding security requirements, product security certifications, source code availability, data disposal requirements, preference for local technologies and expertise, as well as the requirement of development team competencies.
c) Vendor Management – Vendor management includes the management of vendors providing hardware and software, consultancy services, and managed services.
d) The footprint of Resources – Footprint of resources refers to the complete history of the movement of the assets from the origin to the organization. The acquisition process should ensure a complete record of the footprint of resources throughout the system life cycle. The footprint of resources should include the complete hardware and software supply chain.
e) System Life Cycle – Security should be incorporated into all the phases of the system development cycle, including software conceptualization, requirements gathering, design, implementation, testing, acceptance, deployment, maintenance, and disposal. It is recommended that planning for the Information Security Management Plan follows the stages described in most models of ICT System Life Cycle.
f) Commissioning Process – A commissioning process of administrator role and conduct security posture assessment prior to system commissioning and at regular intervals during implementation and when there are changes to the environment.
g) Decommissioning Process – Conduct backup and recovery test, and the data migration before decommissioning. Change management should be conducted to inform the relevant parties regarding the decommissioning of the system.
h) Disposal – Disposal can be in the form of physical destruction and/ or data sanitization.

Annex D

Reference on Example of Use Case and Misuse Case

A use case models the intended behavior of the software. This behavior describes the sequence of actions and events that are to be taken to address a business need. From use cases, misuse cases can be developed. Misuse cases, also known as abuse cases, help identify security requirements by modeling negative scenarios. Meanwhile, security use case should be used to specify requirements that the software application should successfully protect or mitigate the misuse case.

Referring to the following, Figure D-1 shows an example of the use case and misuse case diagram. The use cases identified are the managed item, register item, edit/modify items, view data, and borrow items.

The misuse cases identified are spoof users, invade privacy and manipulate items. Meanwhile, for the security use cases identified are ensure authorization for access control, ensure privacy, ensure the integrity, and ensure non-repudiation.

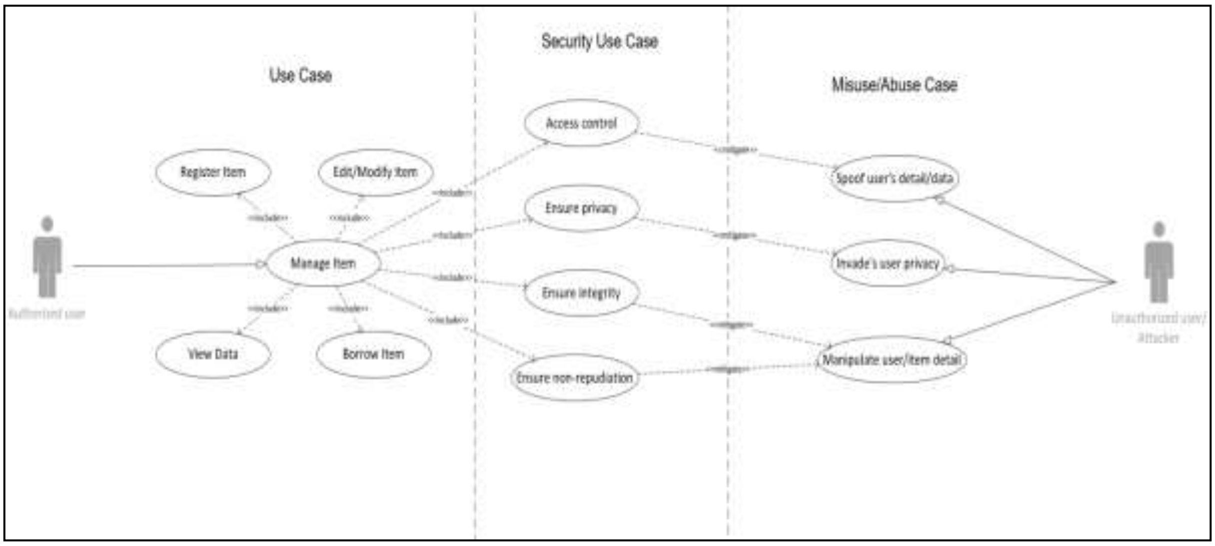

Figure D-1: Example of Use case and Misuse case

Annex E

Reference on Example of Threat Modeling

Microsoft Threat Modelling (MTM) allows software architects to identify and mitigate potential security issues early. This tool is providing structured analysis, guided analysis of threats, and mitigations based on the **STRIDE** taxonomy. STRIDE is an acronym with the meanings are **S**poofing, **T**ampering, **R**epudiation, **I**nformation Disclosure, **D**enial of Service, and **E**levation of Privilege. Besides, this tool also provides reporting capabilities that can be used later in secure coding review and security testing/verification phases.

The tool enables anyone to:

a) Communicate about the security design of their systems.
b) Analyze designs for potential security issues using a proven methodology.
c) Suggest and manage mitigations for identified security issues.

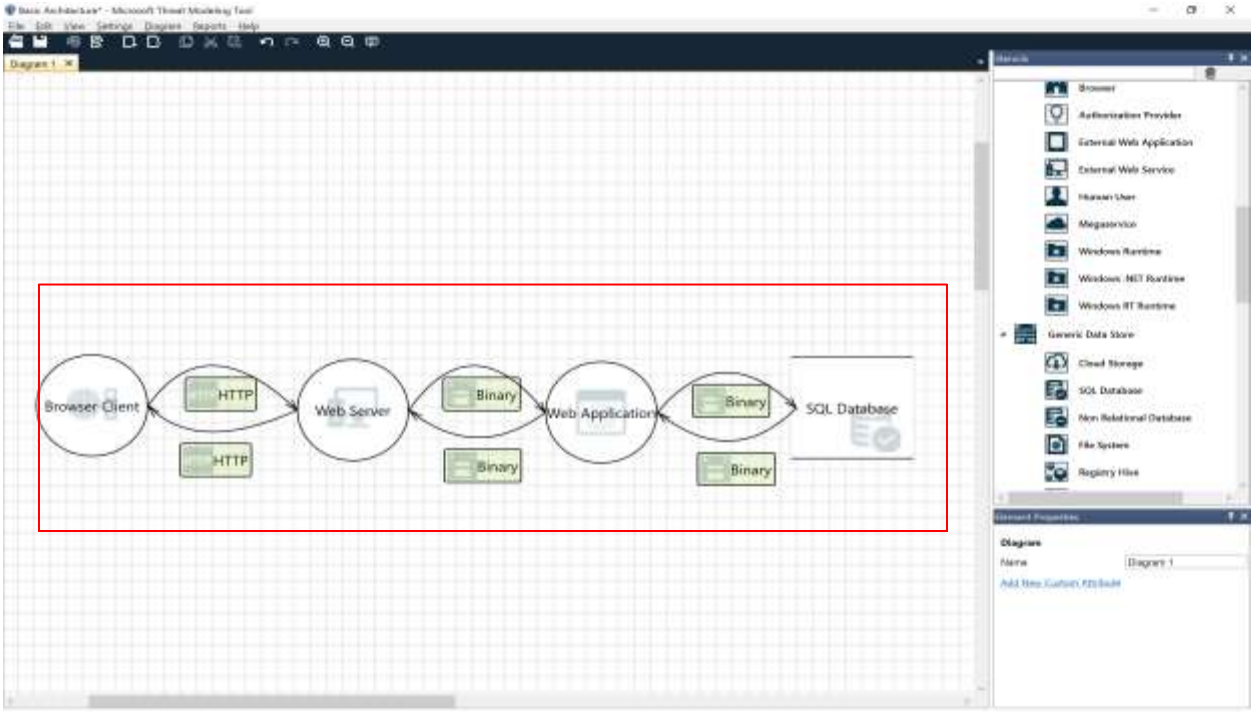

Figure E-1: Data Flow Diagram (DFD)

The Figure E-1 shows the data flow diagram (DFD) of the standard architecture diagram. This tool later will generate a set possible threat based on the architecture diagram provided. However, the result generated from the tool may produce false positive. An analysis is required to choose relevant possible threats against the application developed. The result may be varied due to the setting set for each component in the diagram.

Table E-1 Possible threats generated by MTM

Threat	Category (STRIDE)
A. Interaction: Web Server → Web Application	
1. Elevation using Impersonation	E
2. Cross site scripting	T
3. Replay attacks	T
4. Collision attacks	T
5. Weak authentication scheme	I
B. Interaction: Web Application → Web Server	
1. Elevation using Impersonation	E
2. Cross site scripting	T
3. Web server process memory tampered	T
4. Collision attacks	T
5. Replay attacks	T
6. Weak Authentication scheme	I
C. Interaction: SQL Database → Web Application	
1. Potential Excessive Resource Consumption for Web Application or SQL Database	D
2. Potential SQL Injection Vulnerability for SQL Database	T
3. Spoofing of Destination Data Store SQL Database	S
4. Authorization bypass	I
5. Weak credential storage	I
6. Authenticated data flow compromised	T
D. Interaction: Web Application → SQL Database	
1. Weak access control for a resource	I
2. Persistent cross site scripting	T
3. Cross site scripting	T
4. Spoofing of source data store SQL DB	S
E. Interaction: Browser → Web Server	
1. Elevation using impersonation	E
2. Web server process memory tampered	T
3. Collision attacks	T
4.Replay attacks	T
5. Weak Authentication scheme	I
F. Interaction: Web Server → Browser	
1. Elevation using impersonation	E
2. Cross site scripting	T

Bibliography

[1] Chris, R., "How to put the S (for security) into your IoT development," 18 July 2017. [Online]. Available: https://techbeacon.com/security/how-put-s-security-your-iot-development. [Accessed 7 May 2019]

[2] Giannakidis, A., 2018, "Overview and Evolution of Virtual Patching", 25 September 2018. [Online]. Available: https://dzone.com/articles/introduction-to-virtual-patching. [Accessed 28 August 2019]

[3] ISO/IEC 27000: 2014(E), Information technology – Security techniques – Information security management systems – Overview and vocabulary

[4] ISO/IEC 27002:2013(E), Information technology – Security techniques – Code of practice for information security controls

[5] Kanda Software, "Software Security Services", https://www.kandasoft.com/softwaredevelopment/software-security-services.html

[6] Limited, Ernst & Young Global and The Associated Chambers of Commerce and Industry of India (ASSOCHAM), "Cybersecurity for Industry 4.0: Cybersecurity implications for government, industry and homeland security," Ernst & Young LLP. Published in India., Kolkata, 2018

[7] Malaysia Common Criteria (MyCC), [Online]. Available: https://www.cybersecurity.my/mycc/about.html. [Accessed 28 August 2019]

[8] Malaysia Trustmark, [Online]. Available: https://mytrustmark.cybersecurity.my/. [Accessed 28 August 2019]

[9] Mano Paul, 2014, Official (ISC)^2® Guide to the CSSLP® CBK® Second Edition, CRC Press, Taylor & Francis Group

[10] McGraw, G., 2005, "Software Security: Building Security In", Addison-Wesley Software Security Series

[11] MySEAL (National Trusted Cryptographic Algorithm List), [Online]. Available: https://myseal.cybersecurity.my/en/about.html. [Accessed 28 August 2019]

[12] NIST Special Publication 800-115 Technical Guide to Information Security Testing and Assessment, 2008

[13] NIST special publication 800-64 Revision 2, Security Considerations in the System Development Life Cycle, 2008

[14] Null, C., "3 big IoT security fears, and how developers can tackle them," 31 March 2016. [Online]. Available: https://techbeacon.com/app-dev-testing/3-big-iot-security-fears-how-developers-can-tackle-them. [Accessed 7 May 2019]

[15] OWASP Code Review Guide Version 2.0, 2017

[16] OWASP Secure Coding Practices Quick Reference Guide, Version 2.0, 2010

[17] Pfleeger, C.P., Pfleeger, S.L., & Margulies, J. 2015. Security in Computing, Fifth Edition, Pearson Education, Inc.: Upper Saddle River, NJ

[18] Prasad, K.V.M. & Kishore, J.K., 2013, "Security Requirements Analysis for The Development of Secure GEOSCHEMACS", International Journal of Engineering Research & Technology (IJERT), 2(8): 308-321

[19] Rangka Kerja Keselamatan Siber Sektor Awam (RAKKSSA), Version 1.0, April 2016

[20] Rodriguez, J., 2006, "Common Security Requirement Language for Procurements & Maintenance Contracts", Homeland Security, 8 December 2006

[21] Silva, M.A. & Danziger, M., 2015, The importance of security requirements elicitation and how to do it. Paper presented at PMI® Global Congress 2015—EMEA, London, England. Newtown Square, PA: Project Management Institute. https://www.pmi.org/learning/library/importance-of-security-requirements-elicitation-9634

[22] Software Assurance Maturity Model, Version 1.0 (SAMM-1.0), A guide to building security into software development

[23] Srivastava, S., "Cybersecurity and IOT Industry Facing Skill Shortage Leading to Development Issues," 25 April 2019. [Online]. Available: https://www.analyticsinsight.net/cyber-security-and-iot-industry-facing-skill-shortage-leading-to-development-issues/. [Accessed 7 May 2019]

Acknowledgements

CyberSecurity Malaysia would like to express our appreciation and gratitude to all members of Technical Committee on Guidelines for Secure Software Development Life Cycle (SSDLC) who have participated tirelessly in the development of this guideline. Members are as follows:

Ts. Dr. Zahri Yunos/	CyberSecurity Malaysia
Ts. Dr. Solahuddin Shamsuddin/	
Ts. Dr. Maslina Daud/	
Mr. Abdul Fuad Abdul Rahman/	
Mr. Ahmad Zairi Coursesenu/	
Mr. Khairul Azri Zainal Abidin/	
Ms. Farihan Ghazali/	
Ms. Harmi Armira Mohamad Har/	
Ms. Norul Huda Rasdi/	
Mr. Shazil Imri Mohd Hizam	
Mr. Wilson Lim	Across Vertical Sdn Bhd
Mr. Saiful Bahry M. Hissham	Dell Global Business Center Sdn Bhd
Mr. Subki Zakaria	DXC Technology
Mr. Mohd Hazwan Hussin	HeiTech Padu Bhd
Ms. Harinawati Ahmad	KKMM
Dr. Syarbaini Ahmad	KUIS
Ms. Haslinda Mat Akhir/	MAMPU
Mr. Mohd Nawawi Mustafa/	
Mr. Muhammad Hadri Basri	
Ts. Hafizi Jalil	MyREN
Ms. Aina Nabila Abd Razak	NeuDimension
Mr. Hairul Anuar Mat Nor/	SKMM
Ms. Nor Hashikin Rohani	
Mr. Mohd Rafyiq Ramli/	TEKUN Nasional
Ms. Wan NoorFatin Wan Saris	
Mr. Harmizan Mohd Har	Telekom Malaysia
Mr. Mohd Hazman Hussin	Telekom (R&D) Malaysia
Prof Madya Dr. Fadzlida Mat Sani/	UPM
Dr. Noor Afiza Mohd Ariffin	
Mr. Meor Nazrin M Meor Ahmad/	Venture Nuclues (M) Sdn. Bhd
Mr. Muhammad Firdaus Yahaya	