Security Use Cases are Abuse Case Driven

Modeling Steps for Security Use Cases

1. **Analyze** the possible abuse cases generated for system
2. Identify the security steps required to **mitigate** each abuse case at granular level
3. Include "**mitigation**" mappings between security use case and abuse case
4. Construct a **generalized** use case model consisting of use, abuse, and security use cases

Mitigates Relationship

1. The mitigate relationship in a use case can **portray** the circumstances that can avoid the impact of any abuse case on any user or system
2. It provides **countermeasures** to uncover the abuse case

Abuse Case vs Security Use Case

	Abuse Cases	Security Use Cases
Use	To specify security threats	To specify security requirements
Evaluation Criteria	Attacker succeeds	Application security succeeds
Created by	Security team	Security team
Consumed by	Security team	Requirements Team
Actors	Attacker	User
Driven By	Threat analysis	Abuse cases

Security Use Case: Advantages and Disadvantages

Advantages

- **Large applications** mostly use security use cases
- Most of the security use cases depend on **scientific** and **precise** methods in order to define the security encounters in the system

Disadvantages

- Cannot be used for **small applications**
- Possibilities of **architectural** and **design** issues in the system

Security Use Case Template

- Use **normal** use case template to describe security use cases

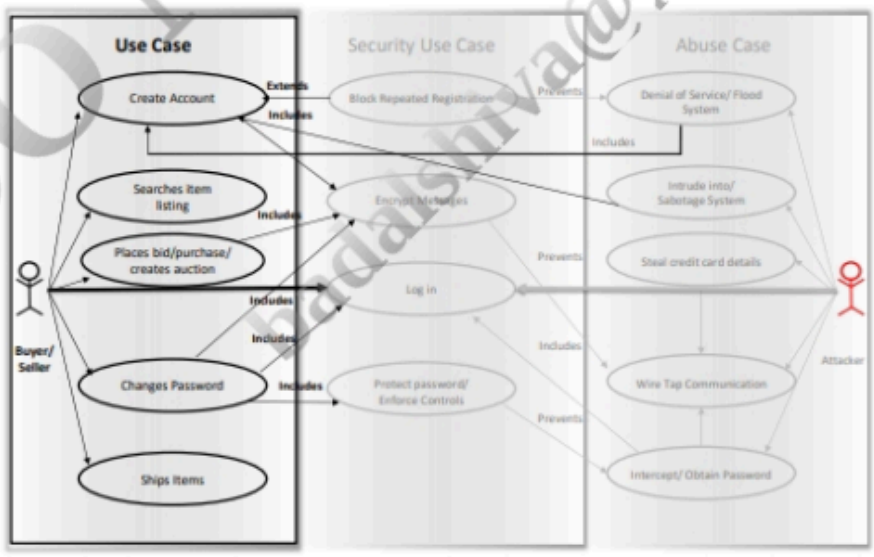
Typical security template consists of:

Use case name:	A unique name for the constructed use case that differentiates it from other use cases
Created By/-Author:	The person responsible for constructing the use case
Date Created/-Date:	The date on which the use case was constructed or created
Actor:	The person responsible for majority of the actions performed in the use case
Description/Summary:	This field provides details regarding the entire use case system in one or two lines
Preconditions:	This field specify those conditions that need to be satisfied by the system prior to the actual implementation of the use case
Post conditions:	This field describes the possible successful outcomes after the implementation of the use case
Basic/Normal course of events:	The field describes the most common path taken by the actors and the system to meet the objectives of the use case
Alternative courses/paths:	Here the less common path taken by the actors and system to meet the objectives of the use case is defined
Exceptions:	The exception field describes the path or the course taken by the actor and the system after the occurrence of an error
Assumptions:	The conditions specified are assumed to be true for the implementation of use case

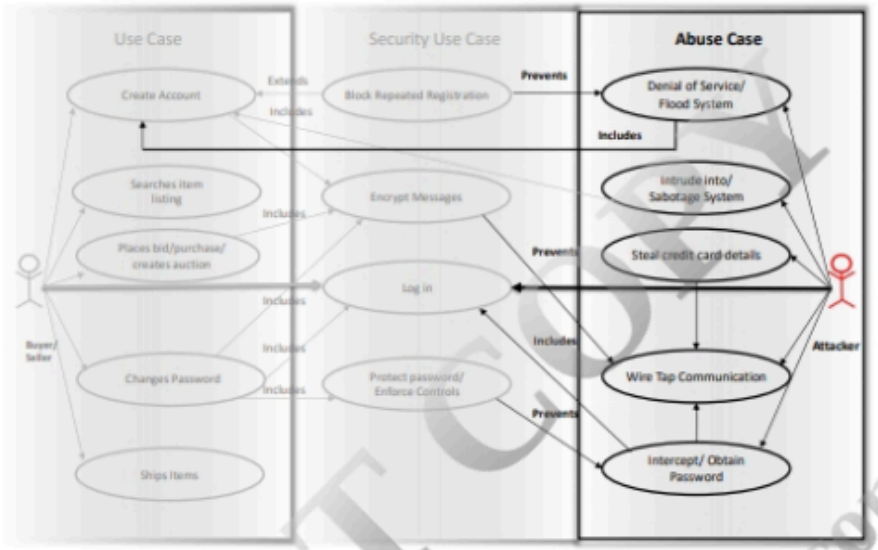
Security Use Case Guidelines

- Use cases should only specify the **vital requirements** needed for the development of the application
- Always ensure that the use cases do not include any **architectural mechanisms** like User ID, passwords, biometrics, etc.
- You may check the **structure** or **arrangement** of **interaction** of the user with the system and specify its order. In this way, it is possible to restrict any chances of evolution of the constraints in design
- It is advisable to create a base **security case** for various types of security requirements in the organization
- Always document the **threats** and **risks** with the help of security use cases
- The difference in **user** and **misuser interactions** must be specified in a transparent way in the security use cases
- The security use case should provide different scenarios for **externally visible** actions and for those which are hidden
- Always ensure to clearly document both **preconditions** and **post conditions** in a security use case

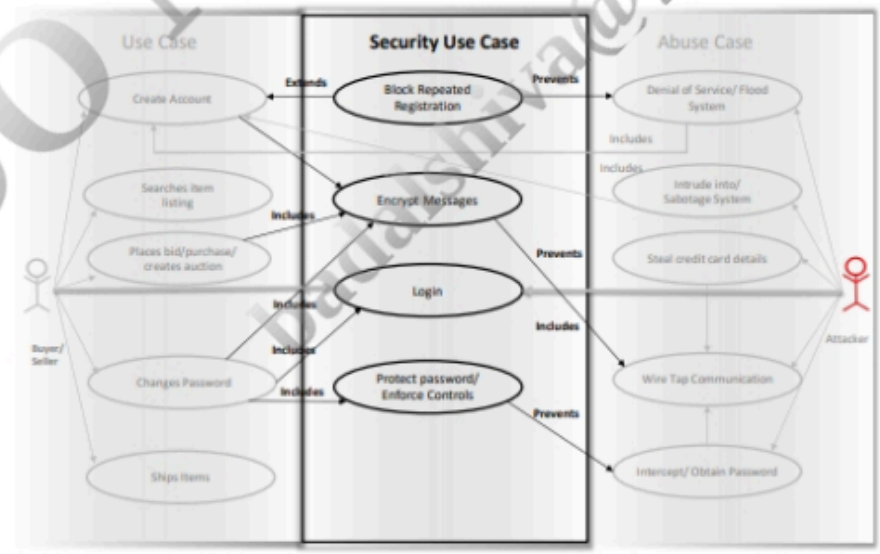
Example 1: Use Case for Online Bidding System



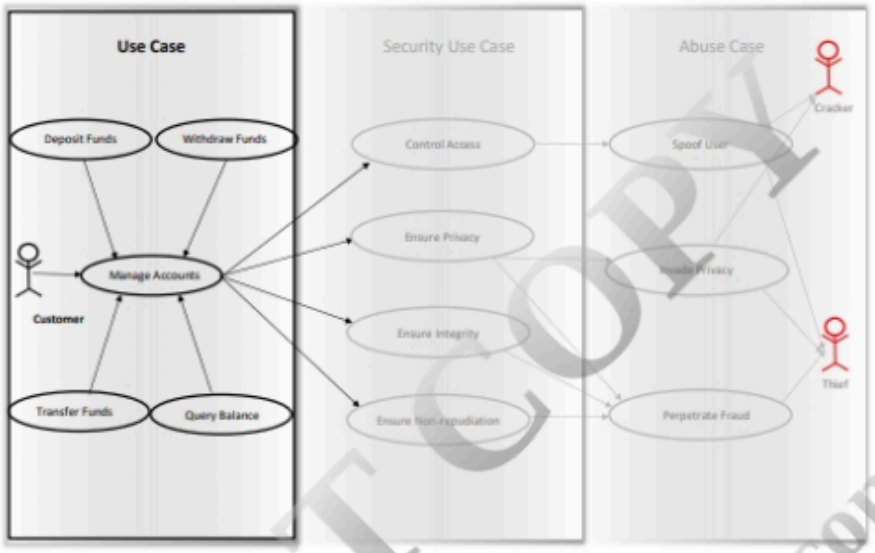
Example 1: Abuse Case for Online Bidding System



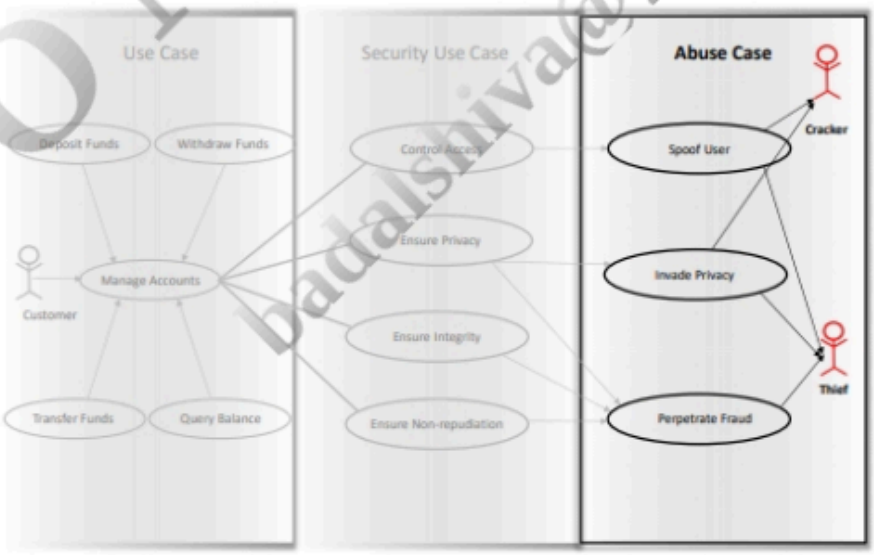
Example 1: Security Use Case for Online Bidding System



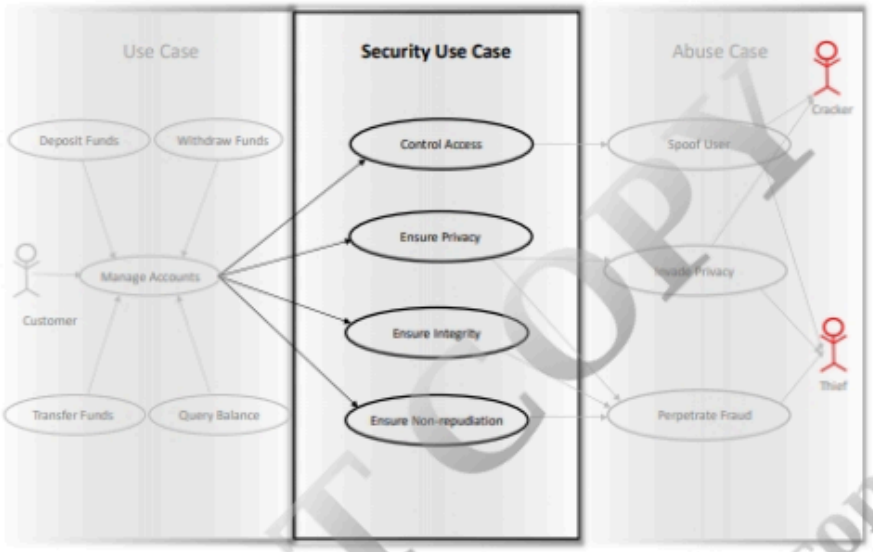
Example 2: Use Case for ATM System



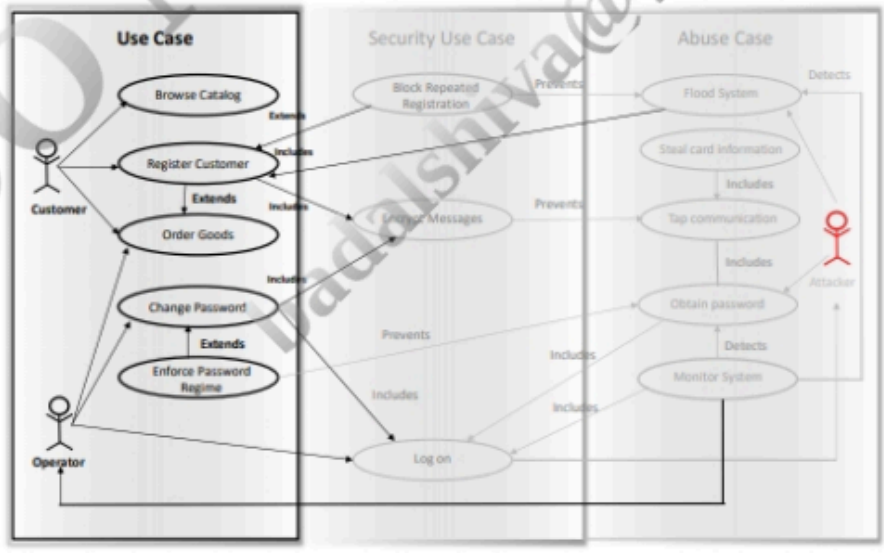
Example 2: Abuse Case for ATM System



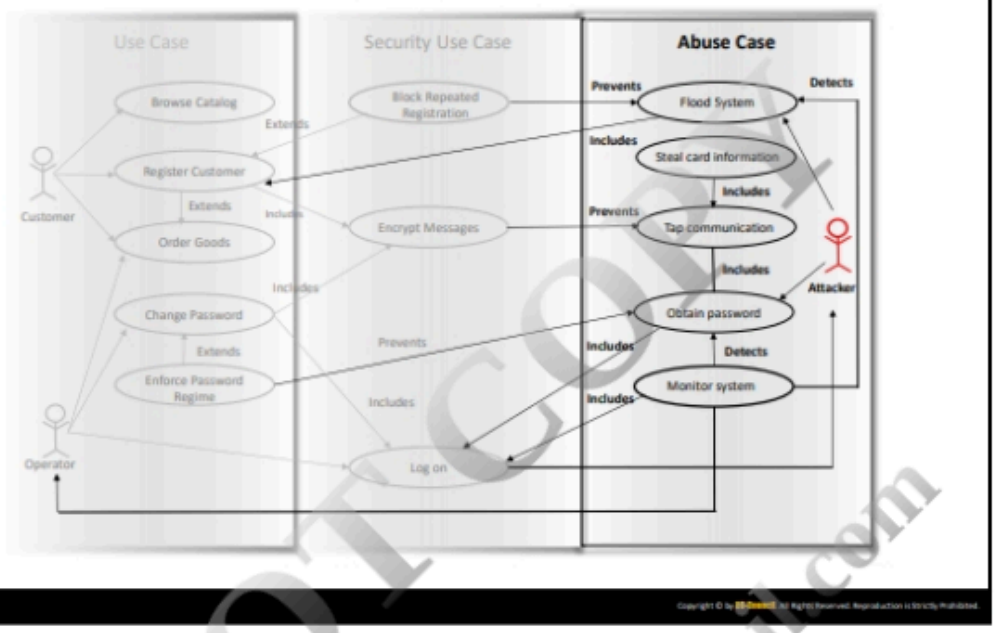
Example 2: Security Use Case for ATM System



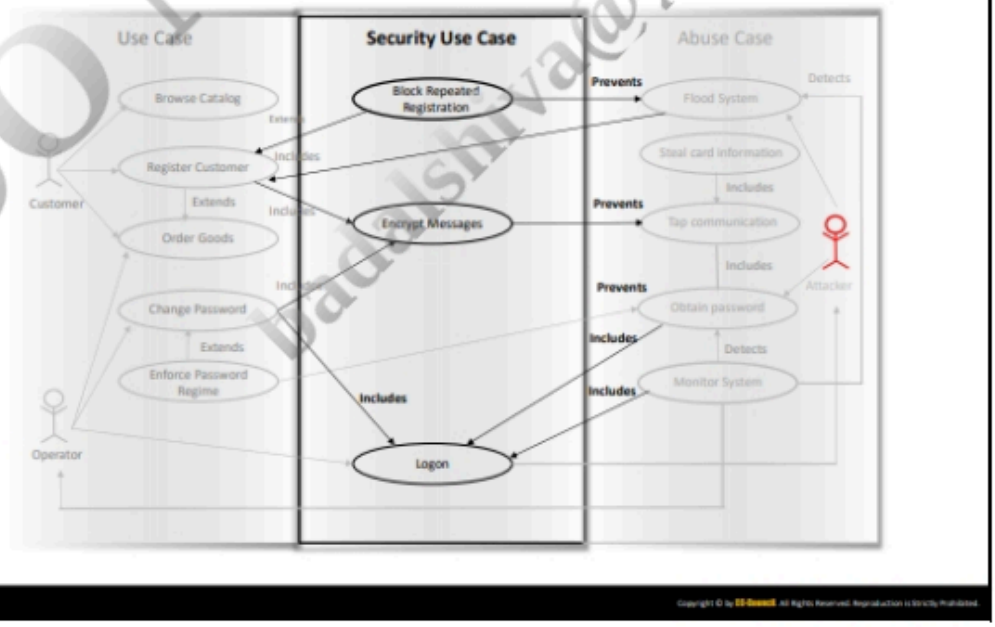
Example 3: Use Case for E-commerce System



Example 3: Abuse Case for E-commerce System



Example 3: Security Use Case for E-commerce System

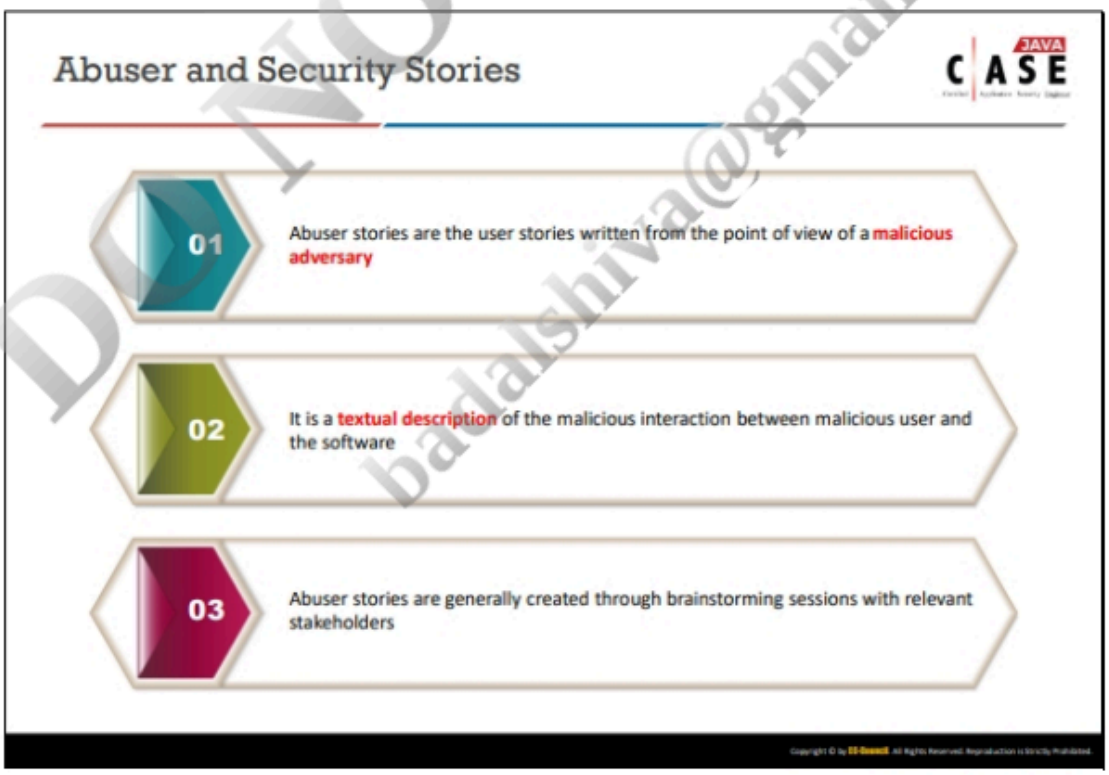


Effectiveness of Abuse and Security Case

Requirement Elicitation	Requirement Analysis
Degree of support provided for requirement elicitation: **High degree of support**	Types of analysis: **Internal and External analysis**
Method used for eliciting requirements: **Brainstorming including clients and developers**	Degree of unambiguity resolution of analysis: **Low level of detecting unambiguity and resolving it**
Degree of stakeholder identification provided: **Only moderate degree of stakeholder identification involved**	Degree of Completeness levels of analysis: **Only moderate degree of support available to ensure complete security requirements**
The level of involvement of the customer: **Ranges from moderate-high as misuse cases depend on customers**	Degree of clarity resolution level of analysis: **Low clarity level**
Elicitation of requirements other than security: **Non-functional requirements**	Degree of support level analysis for considering missed security requirements: **Moderate support level**
Dynamics of the elicitation process: **Iterative process**	Definite or Indefinite prevention of security requirement conflict: **Definite prevention**
Support for establishing system boundaries: **Moderate support**	

Effectiveness of Abuse and Security Cases (Cont'd)

Requirement Specification	Requirement Management
Degree of usage of specification as a baseline for validation and verification of a system after its implementation: **Moderate usage**	Degree of difficulty level in making changes to security requirements: **Easy to make changes**
Degree of cost/time estimation support provided by specification: **Moderate support**	Degree of support level provided for accepting the evolving requirements: **Do not support**
Traceability level provided by the specification: **Low level**	Degree of support level provided for automating any step or process: **Low support**
Degree of consistency level provided by the specification: **Moderately consistent**	Degree of easiness in understandability provided by the approach for a novice user: **Easy to understand**
Degree of support for specifying non-functional requirements: **Moderate-high**	Degree of scalability level : **Low level**
Degree of clarity and understandability level provided by the specification: **Moderate level**	Degree of availability of information available: **Abundant information**
Degree of correctness or formality produced by the specification: **Informal specifications**	
Degree of exactness or rigor of the specification process: **Informal process**	
Degree of security level provided by the specification: **Moderate level of security**	



Textual Description Template: Abuser Stories and Security Stories

Abuser Story Template: As (some kind of bad guy) I want to (do something bad)

Example: *"As a Malicious Hacker I want to steal credit card information so that I can make fraudulent charges"*

Security Story Template: As a (type of user) I want (something) so that (reason)

Example: *"As a(n) architect/developer, I want to verify that sensitive data is kept restricted to actors authorized to access it"*

Security Requirement Elicitation: *"Implement security for user information"*

Examples: Abuser Stories and Security Stories

Abuser Stories

- **Example #1**: "As a hacker I can send bad data in URLs, so that I can access data and functions for which I am not authorized"
- **Example #2**: "As a hacker I can send bad data in the content of requests, so that I can access data and functions for which I am not authorized"
- **Example #3**: "As a hacker I can send data in HTTP headers, so that I can access data and functions for which I am not authorized"
- **Example #4**: "As a hacker I can read and even modify all data that is input and output by your application"

Security Stories

- As a customer I want to have credentials protected
- As a customer I don't want my credentials to be public
- As an administrator I want to detect abuse of the system
- As an administrator I don't want the system to be misused
- As a controller I want data to be protected from unintended manipulation
- As a controller I don't want to have data manipulated by users unintendedly

Effectiveness of Abuser and Security Stories

Requirement Elicitation	Requirement Analysis
Degree of support provided for requirement elicitation: **Moderate – High degree of support**	Types of analysis: **Internal analysis**
Method used for eliciting requirements: **Interviews**	Degree of unambiguity resolution of analysis: **Low level of detecting unambiguity and resolving it**
Degree of stakeholder identification provided: **High degree of stakeholder identification involved**	Degree of completeness levels of analysis: **Low degree of support available to ensure complete security requirements**
The level of involvement of the customer: **Ranges from moderate-high as customers validate and update abuser stories**	Degree of clarity resolution level of analysis: **Provides only Moderate level of clarity as customers constantly validates and updates abuser stories**
Elicitation of requirements other than security: **Do not support**	Degree of support level analysis for considering missed security requirements: **Do not support**
Dynamics of the elicitation process: **Iterative process**	Definite or indefinite prevention of security requirement conflict: **Do not support**
Support for establishing system boundaries: **Low support**	

Effectiveness of Abuser and Security Stories (Cont'd)

Requirement Specification	Requirement Management
Degree of usage of specification as a baseline for validation and verification of a system after its implementation: **Low usage**	Degree of difficulty level in making changes to security requirements: **Easy to make changes**
Degree of cost/time estimation support provided by specification: **Moderate - High support**	Degree of support level provided for accepting the evolving requirements: **Do not support**
Traceability level provided by the specification: **High level**	Degree of support level provided for automating any step or process: **None**
Degree of consistency level provided by the specification: **Low Consistency level**	Degree of easiness in understandability provided by the approach for a novice user: **Moderate – Easy level**
Degree of support for specifying non-functional requirements: **Do not support**	Degree of scalability level : **Low level**
Degree of clarity and understandability level provided by the specification: **Moderate level**	Degree of availability of information available: **Less information**
Degree of correctness or formality produced by the specification: **Informal specifications**	
Degree of exactness or rigor of the specification process: **Informal process**	
Degree of security level provided by the specification: **Low level of security**	

Abuser Stories: Advantages and Disadvantages

Advantages

- Abuser stories identify the characteristics of a system when under threat or failure
- It helps in documenting all the flaws and gaps in a system that needs to be rectified

Disadvantages

- Abuser stories are best applicable only in agile process
- It is not possible to completely secure the system using abuser stories approach

Security Quality Requirements Engineering (SQUARE)

Security Quality Requirements Engineering (SQUARE)

- System Quality Requirements Engineering (SQUARE) is a process model that was developed at **Carnegie Mellon University**
- Security Quality Requirements Engineering (SQUARE) provides a means for **eliciting**, **categorizing**, and **prioritizing** security requirements for information technology systems and applications
- The focus of this methodology is to build security concepts into the early stages of the development life cycle

Source: *https://www.us-cert.gov*

SQUARE Effectiveness

Requirement Elicitation	Requirement Analysis
Degree of support provided for requirement elicitation: **Moderate – Low degree of support**	Types of analysis: **Internal and External analysis**
Method used for eliciting requirements: **Any technique like Interviews and Surveys**	Degree of unambiguity resolution of analysis: **High level of detecting unambiguity and resolving it**
Degree of stakeholder identification provided: **Moderate - High degree of stakeholder identification involved**	Degree of completeness levels of analysis: **Moderate level of support available to ensure complete security requirements**
The level of involvement of the customer: **High involvement of customers**	Degree of clarity resolution level of analysis: **Provides only High level of clarity**
Elicitation of requirements other than security: **Functional and Non-functional**	Degree of support level analysis for considering missed security requirements: **Low level of support**
Dynamics of the elicitation process: **Iterative process**	Definite or indefinite prevention of security requirement conflict: **Definite prevention**
Support for establishing system boundaries: **Low support**	

SQUARE Effectiveness (Cont'd)

Requirement Specification	Requirement Management
Degree of usage of specification as a baseline for validation and verification of a system after its implementation: **Moderate – High usage**	Degree of difficulty level in making changes to security requirements: **Moderate level**
Degree of cost/time estimation support provided by specification: **High support**	Degree of support level provided for accepting the evolving requirements: **Moderate - High**
Traceability level provided by the specification: **Low level**	Degree of support level provided for automating any step or process: **Low**
Degree of consistency level provided by the specification: **Moderate – High Consistency level**	Degree of easiness in understandability provided by the approach for a novice user: **Moderate difficulty**
Degree of support for specifying non-functional requirements: **Low support**	Degree of scalability level : **High level**
Degree of clarity and understandability level provided by the specification: **Moderate level**	Degree of availability of information available: **Abundant information**
Degree of correctness or formality produced by the specification: **Semi_ Formal specifications**	
Degree of exactness or rigor of the specification process: **Semi – Formal process**	
Degree of security level provided by the specification: **Moderate – High level of security**	

SQUARE Process

Number	Step	Input	Techniques	Participants	Output
1	Agree on definitions	Candidate definitions from IEEE and other standards	Structured interviews, focus group	Stakeholders, requirements engineer	Agreed to definitions
2	Identify assets and security goals	Definitions, Candidate goals, Business drivers, Policies and procedures, Examples	Facilitated work session, surveys, interviews	Stakeholders, requirement engineer	Assets and goals
3	Develop artifacts to support security requirements definition	Potential artifacts (e.g.: scenarios, misuse cases, templates, forms)	Work session	Requirements engineer	Needed artifacts: scenarios, misuse cases, models, templates, forms
4	Perform risk assessment	Misuse cases, scenarios, goals	Risk assessment method, analysis of anticipated risk against organizational risk tolerance, including risk analysis	Requirements engineer, stakeholder, risk expert	Risk assessment results

Source: https://www.us-cert.gov

SQUARE Process (Cont'd)

Number	Step	Input	Techniques	Participants	Output
5	Select elicitation techniques	Goals, definitions, candidate techniques, expertise of stakeholders, organizational style, culture, level of security needed, cost/benefit analysis, etc.	Work session	Requirements engineer	Selected elicitation techniques
6	Elicit security requirements	Artifacts, risk assessment results, selected techniques	Joint application development, surveys, interviews, model-based analysis, checklists, lists of reusable requirements types, document reviews	Stakeholders facilitated by requirement engineer	Initial out at security requirements
7	Categorize requirements as per level (system software, etc.) and whether they are requirements or other kinds of constraints	Initial requirements, architecture	Work session using a standard set of categories	Requirements engineer, other specialists as needed	Categorized requirements
8	Prioritize requirements	Categorized requirements and risk assessment results	Prioritization methods such as analytical hierarchy process (AHP) Triage, Win-Win	Stakeholders facilitated by requirement engineer	Prioritized requirements
9	Inspect requirements	Prioritized requirements, candidate formal inspection technique	Inspection method such as Fagan, Per-reviews	Inspection team	Initial selected requirements, documentation of decision-making process and rationale

Source: *https://www.us-cert.gov*

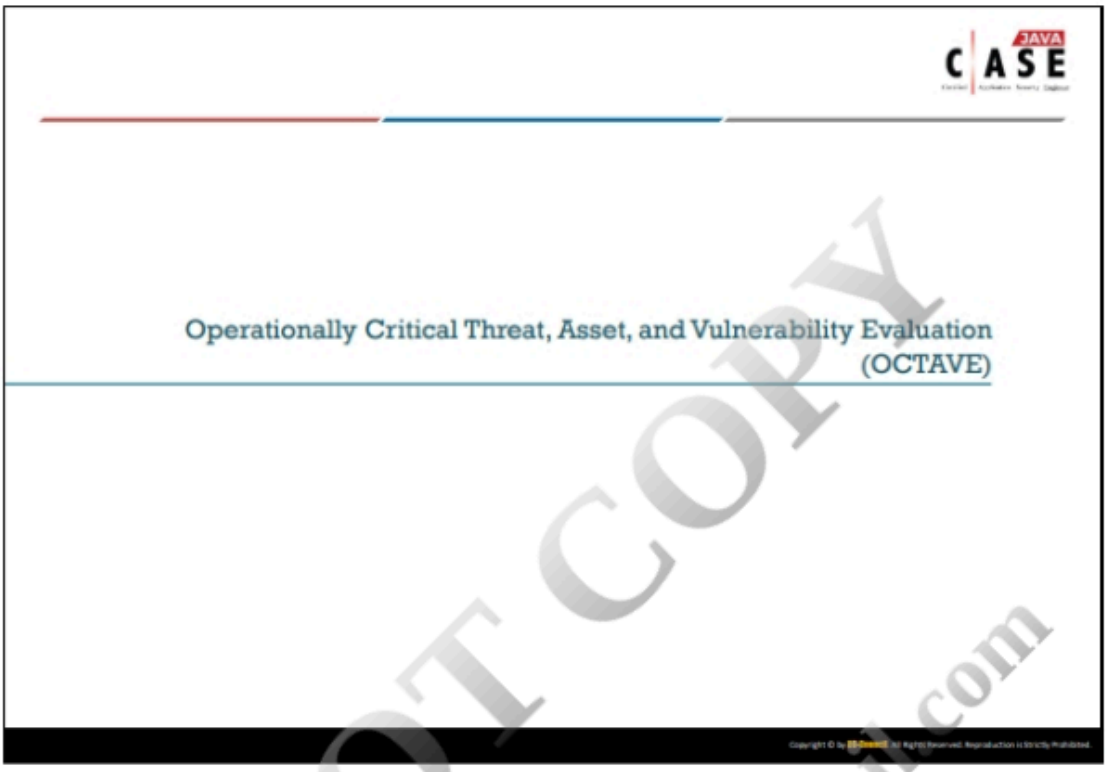
SQUARE: Advantages and Disadvantages

Advantages

- Enables **easy understanding** of the system and at the same time is easy to learn
- **Inexpensive** to implement

Disadvantages

- Only useful for **large projects**



Operationally Critical Threat, Asset, and Vulnerability Evaluation (OCTAVE)

- OCTAVE is an **information risk evaluation** method that enables organizations to determine the risk factors affecting the confidentiality, integrity, and availability of assets
- OCTAVE provides a **structured approach** for identifying, prioritizing, and managing the security risks in an organization
- It also enables the organization to make a better understanding of the security requirements needed for **maintaining security** of the assets

The OCTAVE approach consists of three phases:

1. **Phase 1**: Identifying the asset based threat
2. **Phase 2**: Evaluating and identifying the vulnerabilities
3. **Phase 3**: Drafting security strategies and plans

OCTAVE Effectiveness

Requirement Elicitation	Requirement Analysis
Degree of support provided for requirement elicitation: Moderate – **High degree of support**	Types of analysis: **High Internal analysis**
Method used for eliciting requirements: **Workshops**	Degree of unambiguity resolution of analysis: **None as only low level of support is provided to clear the unambiguity in the requirements gathered**
Degree of stakeholder identification provided: **High degree of stakeholder identification involved**	Degree of completeness levels of analysis: **Moderate level of support available**
The level of involvement of the customer: **High involvement of customers as they mostly participate in eliciting security requirements as well as gathering information required for the effectiveness of OCTAVE**	Degree of clarity resolution level of analysis: **Provides only Moderate – High level of clarity**
Elicitation of requirements other than security: **Non-functional requirements**	Degree of support level analysis for considering missed security requirements: **Low level of support**
Dynamics of the elicitation process: **Iterative process**	Definite or indefinite prevention of security requirement conflict: **Low prevention as it does not explain the steps for the developers to clear the conflict**
Support for establishing system boundaries: **Moderate support**	

OCTAVE Effectiveness (Cont'd)

Requirement Specification	Requirement Management
Degree of usage of specification as a baseline for validation and verification of a system after its implementation: **Moderate usage**	Degree of difficulty level in making changes to security requirements: **High level**
Degree of cost/time estimation support provided by specification: **High support**	Degree of support level provided for accepting the evolving requirements: **Do not support**
Traceability level provided by the specification: **Moderate level**	Degree of support level provided for automating any step or process: **Do not support**
Degree of consistency level provided by the specification: **Low Consistency level**	Degree of easiness in understandability provided by the approach for a novice user: **Difficult**
Degree of support for specifying non-functional requirements: **Do not support**	Degree of scalability level: **High level**
Degree of clarity and understandability level provided by the specification: **Moderate - High level**	Degree of availability of information available: **Abundant**
Degree of correctness or formality produced by the specification: **Semi - Formal specifications**	
Degree of exactness or rigor of the specification process: **Formal process**	
Degree of security level provided by the specification: **Moderate level of security**	

OCTAVE Steps

The steps involved in OCTAVE security requirement process are:

1. Identify the important and valuable assets
2. Determine the security objective for those assets
3. Identify and analyze the threats and risks involved in the identified critical assets
4. Determine security requirements for the identified threats and risks

OCTAVE: Advantages and Disadvantages

Advantages

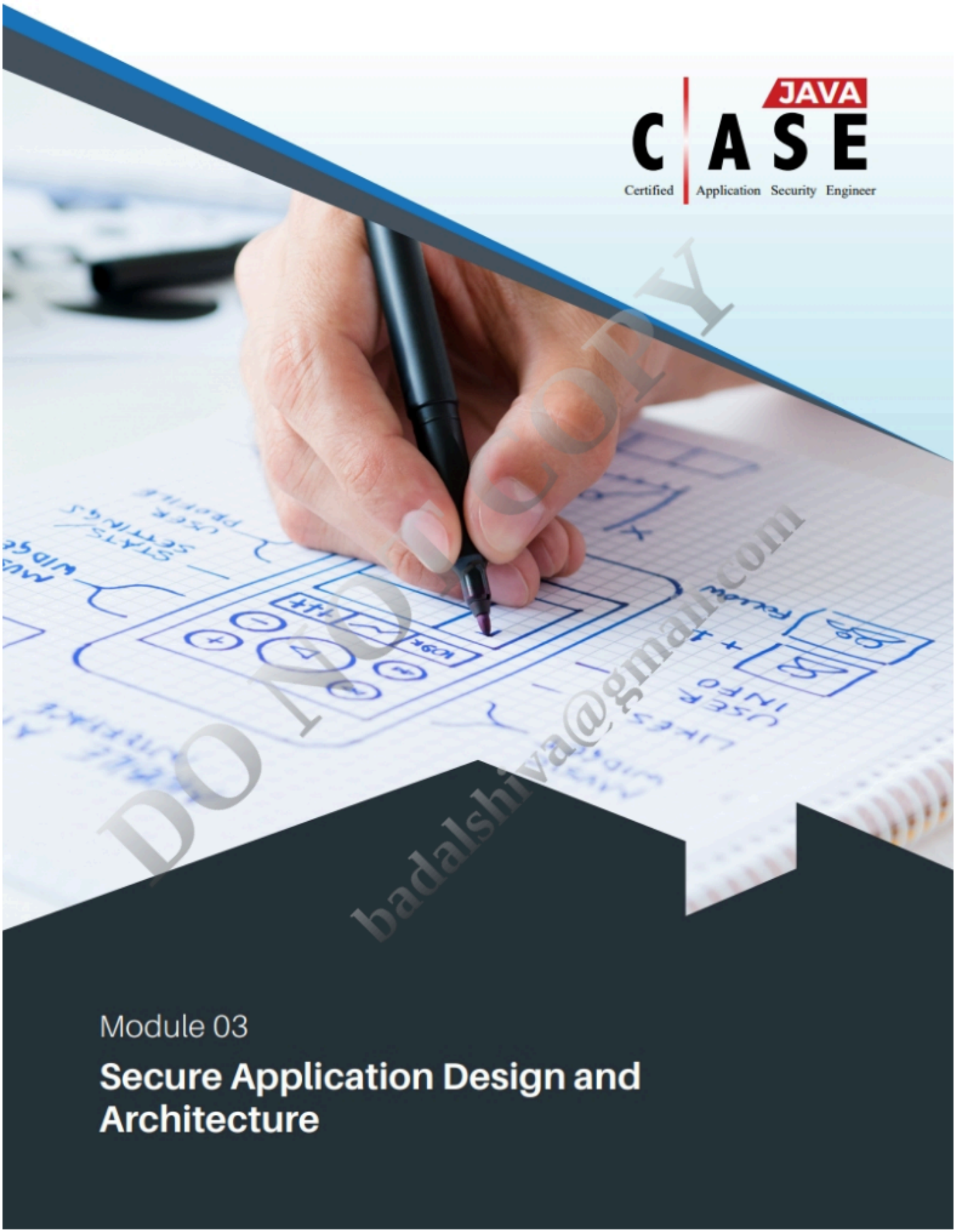
- It enables small and large organizations to apply OCTAVE approach during security elicitation process

Disadvantages

- It is difficult to understand and implement the concepts of OCTAVE approach
- The approach does not provide a detailed explanation of the quantitative analysis of the security risks in the organization

Module Summary

- Security requirements is negative requirements specifying what software should not do
- Gathering security requirements should be part of the strategic application development process
- Eliciting software security requirements takes different approach
- Security requirements should be enumerated separately from functional requirement so that they can be reviewed and tested separately
- Security Requirement Engineering (SRE) empowers security requirement gathering in the application development process



Printed by: badalshiva@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

This page is intentionally left blank.

Module Objectives

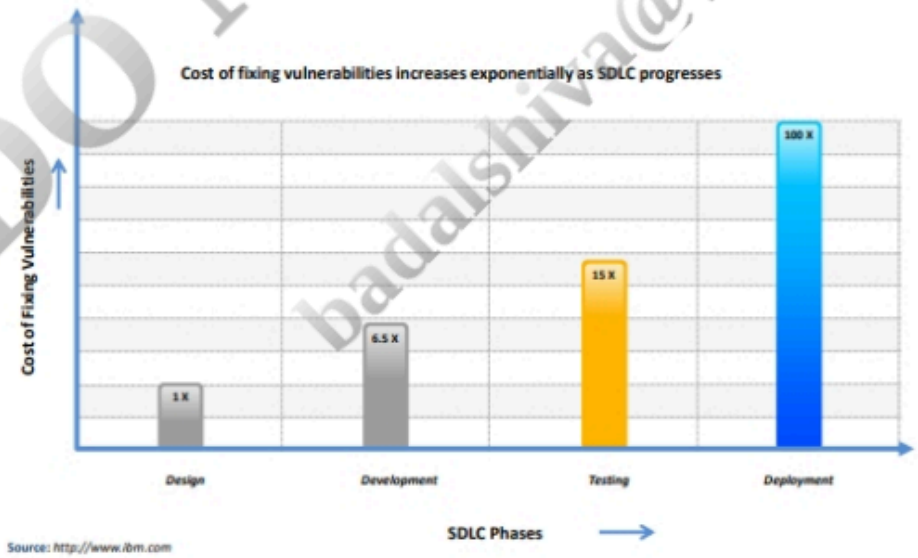
1. Understand the Importance of Secure Application Design
2. Discuss Various Secure Design Principles
3. Understand Threat Modeling
4. Describe Threat Modeling Process
5. Discuss STRIDE and DREAD Model
6. Describe Secure Application Architecture Design

Most of the web applications are vulnerable due to insecure design of application at design phase.

"Secure programming cannot make up for a poor design."

-- PHP Security Consortium

Relative Cost of Fixing Vulnerabilities at Different Phases of SDLC



Source: http://www.ibm.com

Secure Application Design and Architecture

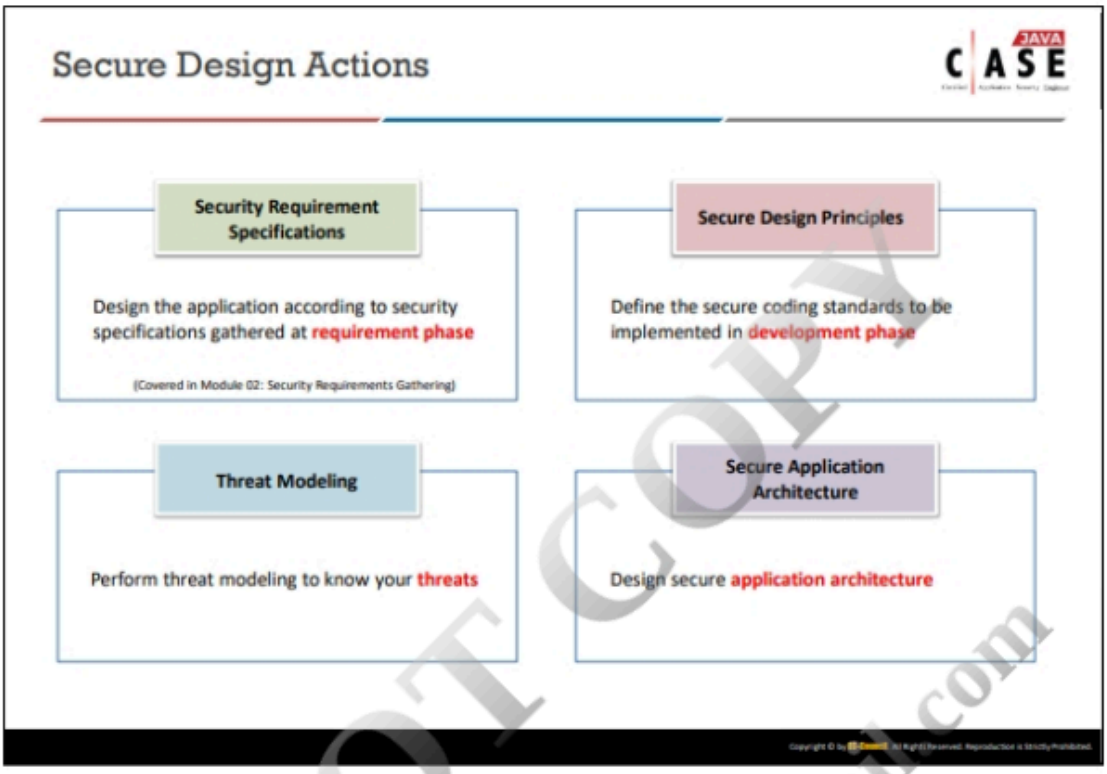
- A security negligence at **design** and **architecture** phase may lead to vulnerabilities that are difficult to detect and expensive to fix in production
- Security vigilance at design phase enables detecting potential **security flaws** early in the software development lifecycle
- Secure design of an application is based on **security requirements** identified in the previous phase of the SDLC
- Secure design is a **challenging process** as designing required security controls may obstruct the business functionality requirements

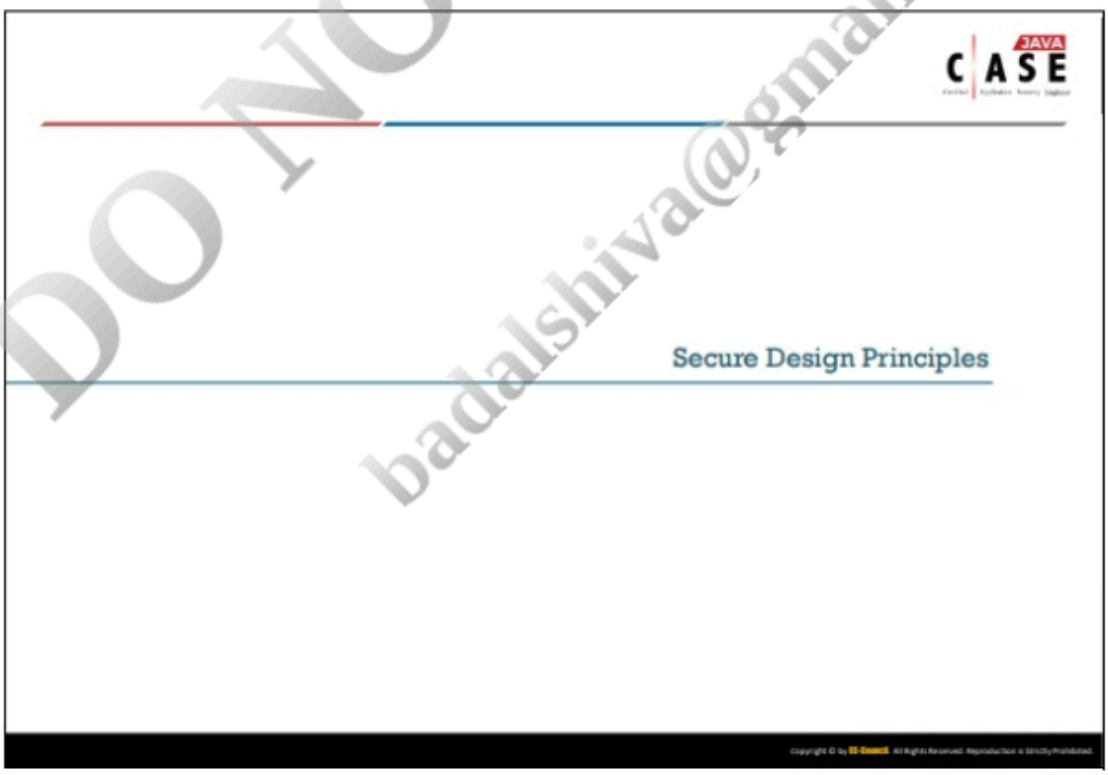
Goal of Secure Design Process

I. Identifying the threats in sufficient details for **developers** to understand and code accordingly to mitigate the risk associated with the threats

II. Designing a **architecture** in such a way that it mitigates as many threats as possible

III. Enforcing **secure design principles** that force developers to consider security while coding





Define Secure Design Principles

- Secure design principles are the state of **practices or guidelines** that should be enforced on the developers to follow during development phase
- It helps in deriving **secure architectural decisions**
- It helps to eliminate design and architecture **flaws** and mitigate common security vulnerabilities within the application

Secure Design Principles

- **List of secure design principles to prevent common security vulnerabilities:**
  - Security through obscurity
  - Secure the weakest link
  - Use least privilege principle
  - Secure by default
  - Fail securely
  - Apply defense in depth
  - Do not trust user input
  - Reduce attack surface
  - Enable auditing and logging
  - Keep security simple
  - Separation of duties
  - Fix security issues correctly
  - Apply security in design phase
  - Protect sensitive data
  - Exception handling
  - Secure memory management
  - Protect memory or storage secrets
  - Fundamentals of control granularity
  - Fault tolerance
  - Fault detection
  - Fault removal
  - Fault avoidance
  - Loose coupling
  - High cohesion
  - Change management and version control

Secure Design Principles (Cont'd)

Security through Obscurity

- Security Through Obscurity (STO) relies on **preventing access** to certain users to protect internal data
- STO systems may have theoretical or actual security vulnerabilities, but designers believe that **flaws are unknown** and attackers are unlikely to find them
- Its usefulness has declined with the rise of **open systems**, networking, greater understanding of **programming techniques**, and increased capabilities of home users

Secure the Weakest Link

- Attackers target a system that is easy to **penetrate**
- For example, to gain access to the **encrypted** data on the network, attackers will not intercept the data and crack encryption; instead they will go after the **end points** of communication to find a **flaw** that discloses the data
- Identify and strengthen the **areas at risk** until levels of risk are satisfactory

Secure Design Principles (Cont'd)

Use Least Privilege Principle

- Applications with maximum **system privileges** are vulnerable to the **attacks**
- For example: Many web applications use **database admin account** though not required to connect to the backend database, enhancing the impact of SQL injection exploits
- Protects application from malicious attacks by:
  - Determining and assigning rights only to those who require **privileges** to complete the specific task
  - Avoiding applications that get **installed** and **run** by **default**
  - Writing applications that can be used by users having **non-administrative privileges**

Secure by Default

- The software solution or application provided to the users should be **security enabled** by default. If permitted, it is up to the user to reduce the security
- For example: By default the security feature **password aging** and **complexity** should be enabled

Secure Design Principles (Cont'd)

Fail Securely

- The developer should not give application secrets by default **error messages**
- Application that discloses **confidential information** on failure assists attackers in creating an **attack**
- When an application fails, determine what may occur and ensure that it does not threaten the application
- Provide logical and useful **error messages** to the users and store the details in the **log file**

Apply Defense in Depth

- The architects and developers should consider all the **levels of the software** to impose security while developing software
- Implement security mechanisms at **different layers** that include network layer, kernel layer, physical layer, and the file system layer

Do Not Trust User Input

- Protect the application from all **malicious inputs** coming from the user input to the application
- Consider all inputs as a malicious input and apply **security measures** to restrict them

Secure Design Principles (Cont'd)

Reduce Attack Surface

- Application attack surface area is to be minimized by **reducing the number of entry points** into the application
- Remove or turn off the features, protocols, and functionality which are **not in use** to minimize number of vulnerabilities and the overall risk
- For example: If a vulnerability exists in a way an XML is parsed, denying XML from unknown users minimizes that security vulnerability

Enable Auditing and Logging

- Auditing and logging states how the **security related events** are recorded by an application
- Auditing enables **identification of attacks or intruders in progress**, whereas logging aids in identifying how an attack is performed
- Perform auditing and logging to gather information about **attacks**

Secure Design Principles (Cont'd)

Keep Security Simple

- If the design is complicated, it is hard to understand and **errors** are likely to occur in implementation, configuration, and use
- On the other hand, if the **complexity** of security mechanisms increases, the effort required to reach the appropriate level of **software assurance** also increases
- Avoid complex **architectures** and opt for **simpler approaches** that are fast and simple

Separation of Duties

- Separation of duties is the **key control of fraud**
- When assigning privileges, system roles are to be considered. In general, system administrators are also the users as some super user privileges are required to make the system run
- For example: **System administrator** can set the password policy, turn off or on the system, etc. but should not be able to log in as a **super-privileged** user

Secure Design Principles (Cont'd)

Fix Security Issues Correctly

- When a security issue is identified, fix it, considering it as the actual problem, and then go through the **security process** as you do for the new code, ensuring that the fix does not introduce new errors
- **For example:** User capable of viewing **another user's account balance** by simply adjusting cookie. In this context once the security issue is fixed it has to be tested on all the applications as cookie handling code is shared among all applications

Apply Security in Design Phase

- Before starting the application development process, always consider **security issues** that can help prevent many **security vulnerabilities**
- Considering security issues helps you understand the **coding weaknesses** and **vulnerabilities** from the most obvious exploits

Protect Sensitive Data

- Do not **hard code** the sensitive data such as passwords in the program
- Use **data encryption mechanism** to transmit data over the network

Secure Design Principles (Cont'd)

Exception Handling

- Events that disrupt the **coding process** are called exceptions
- Exception handling occurs when error conditions **interrupt** the normal flow of a program's execution
- Programmers have difficulty in designing for exception handling, as continuous checking for **error conditions** is necessary
- Proper use of exception handling helps to ensure proper **error handling**

Secure Memory Management

- Check memory bounds on the length of input variables, arrays, and arguments to prevent **buffer overflow attacks**
- Apply coding standards for simplicity, which help in **implementing security** in the **program** and keep things simple

Protect Memory or Storage Secrets

- **Encrypt secrets** to protect memory storage from ending up in crash dump file
- Use a perfect **cryptographic method** to perform encrypting secrets process
- Scrub secrets in memory storage before **deletion**

Secure Design Principles (Cont'd)

Fundamentals of Control Granularity

- Applications managing sensitive data should possess **privacy protection** that can be achieved through flow control
- **Flow control** prevents data from non-secure information flows
- Systems with low-level granularity may lack control in **protecting data** from misuse
- System administrators are given better control of granularity at the URL level, shared directory level, and application level
- Different granularity is needed to control **information flows** within an application
- **Role-based Access Control (RBAC)** is a flow control model offering multiple levels of **control granularity**
  - Also called an **n-leveled RBAC (LnRBAC)**, as a level of granularity is controlled by a level of RBAC control
  - Handles and solves problems caused by abnormal program stopping
  - Controls method invocation, write accesses, and avoids Trojan horses

Secure Design Principles (Cont'd)

Fault Tolerance

- Strategy applied to **software design** (or system design) to permit system to continue functioning even in the presence of faults by enhancing its robustness

Fault Detection

- Closely linked to fault tolerance, used in **detecting faults** and producing appropriate responses of **system behavior**
- Examples include system monitors, safety monitors, built-in tests, loop-back tests, etc.

Fault Removal

- Removes faults during **design process**
- Examples include error detection, verification through inspection, built-in testing, correction functions, etc.

Fault Avoidance

- Avoids errors that contribute to system faults during the **development process**
- Examples include defensive programming, error minimization during design process, minimization of safety critical code, using appropriate SDLC techniques, etc.

Secure Design Principles (Cont'd)

Loose Coupling

- Procedures that **operate independently** from other procedures are called loosely coupled procedures
- Loose coupling describes the **exchange relationship** that occurs between two or more systems or organizations
- The concept loose coupling of systems is adopted when either source or destination machines are supposed to be **changed frequently**
- Loose coupling can be achieved in **web services** or **service-oriented architecture** by hiding the implementation details from the caller

Well-known Techniques that Create Loose Coupling

- Vendor and platform independent messages
- Coarse-grained, self-describing, and self-contained messages
- Well-defined interfaces
- Extensible versionable interfaces
- Constrained interfaces
- Stateless messaging
- Human readable strings (URIs) for service and instance addresses
- Stateless messaging where possible and appropriate
- Humans controlling clients where possible and appropriate
- Asynchronous exchange patterns where possible and appropriate

Secure Design Principles (Cont'd)

High Cohesion

- Procedures that perform a single function
- Cohesion is a measure of identifying the extent of strongly related functionalities in the **source code** of the single module
- A particular class is said to have high cohesion if the methods in that class perform **similar actions** in many aspects
- This high cohesively code increases the code **readability** and **reusability** without increasing complexity
- Procedures that are more reliable, easy to read, and maintain are usually **loosely coupled** and have **high cohesion**

Change Management and Version Control

- **Integrity** of system changes is controlled by a well-defined process called **configuration management**
- Allows all **stakeholders** to know upcoming changes
- Change management controls integration **costs** and **development**
- Required changes to code may cause **vulnerabilities**
- Anticipation of change requirements may **limit negative effects**

Threat Modeling

"It is not possible to design and develop a secure application until you know your threats."

Source: msdn.microsoft.com

Threat Modeling

- Threat modeling is a process of **identifying, analyzing,** and **mitigating the threats** to the application
- It is a structured approach that allows the developer to **rate the threats** based on the architecture and implementation of the application
- It is performed at the **design phase** of the secure development lifecycle
- It is an **iterative process** that starts from the design phase of the application and iterates throughout the application lifecycle until all possible **threats to the applications** are identified
- The output of threat modeling is a threats model exposing all the possible **threats** and **vulnerabilities** on an application

Threat Modeling Phases

Attack Surface Evaluation	Application is decomposed and its entry points are reviewed from an attacker's perspective
Threat Identification	Each entry points are then reviewed against potential threats
Impact Analysis	Impact of potential threats is calculated in terms of risk
Control Recommendations	Security controls are recommended to meet security objectives

Threat Modeling Process

- **The steps involved in Threat Modeling process are as follows:**

Identify the Security Objectives

- To proceed with creating threat model for an application, the **security objectives** of the application should be identified clearly
- These security objectives are **goals** and **constraints** that needs to be attain to maintain confidentiality, integrity, and availability of application and its data
- Security objectives are the subset of the application's **functional objectives**
- With security objectives clearly identified,
  - You can identify the areas which need more closer **attention** and where to focus your **efforts**
  - You can understand goals of potential attackers

How to Identify Security Objectives

- Security objectives are identified by conducting **brain storming sessions, questionnaires, storyboarding, etc.** with relevant stakeholders of the project

Sample Questionnaires	Sample Security Objectives
Does your application handles or processes sensitive data that you need to protect from unauthorized access? Identify such data that you need to protect from customers/clients Example of Sensitive data: user names, passwords, customer account details, personal information, financial details, transaction details, credit card numbers, bank details, etc.	Application should prevent unauthorized person from accessing confidential information of the customers
Does your application requires to comply with certain security policies, privacy laws, regulations, or standards Example: PCI-DSS, HIPPA, etc.	Application should meet the compliance requirements
Does organization wants application to satisfy certain quality of service requirements Example: availability, performance, etc.	Application should meet the quality of service requirements
Does application has any intangible assets that you need to protect For example: Organization's reputation, trade secrets, intellectual property, etc.	Application should protect the company's online business credibility

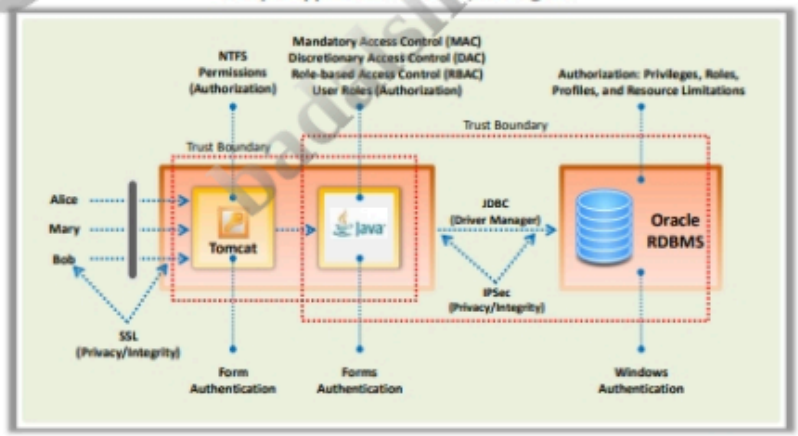
Create an Application Overview

- The aim of creating application overview is to understand **what the application does**
- Creating an application overview involves:
  - **Understanding** the application functionality
  - Understanding the **purpose** and the key **features** of the application
  - Identifying application **users** or **clients**
- **Create application overview with the help of these activities:**
  1. Draw the end-to-end deployment architecture
  2. Identify various user roles
  3. Identify use case scenarios
  4. Identify technologies
  5. Identify application security mechanisms

Draw the End-to-End Deployment Architecture

- Draw a **high-level** diagram that describes the overall structure of the application and its subsystems in the deployment environment
- Identify end-to-end deployment topology, logical layers, key components and services, communication ports and protocols, etc.
- Add **authentication**, **authorization**, and **communication** mechanisms at broad level in the deployment diagram

Sample Application Architecture Diagram



Identify Various User Roles

- Identify **who can do what** in the application
- Identify users or group of users with **higher privileges**
- Identify the **rights** of specific users

Examples of Roles
Anonymous Web User
User With Valid Login Credentials
User With Invalid Login Credentials
Librarian
Database Server Administrator
Website Administrator
Web Server User Process
Database Read User
Database Read/Write User

Identify Use Case Scenarios

- Identify the **functionality** and **usage** of an application
- Identify **Create**, **Read**, **Update**, and **Delete** operations in the application
- Review following items to gain understanding of key usage of the application
  - Review existing documents
  - Review the use cases
  - Review requirements
- This will help you understand how the application should be **used** and how it can be **misused**

Identify Technologies

- Identify the **technologies** used to develop and deploy the application
  1. Operating systems
  2. Web server software
  3. Database server software
  4. Technologies used in the presentation, business, and data access layers
  5. Development languages
- This will help you identify **technology-specific threats**

Identify Application Security Mechanisms

- Identify and highlight **key elements** with respect to following application security mechanisms
  - Input and data validation
  - Authentication
  - Authorization
  - Configuration management
  - Sensitive data
  - Session management
  - Cryptography
  - Parameter manipulation
  - Exception management
  - Auditing and logging
- This will help you add **details** wherever necessary

Decompose the Application

- Decompose the application to the level where it is possible to identify external entities, trust boundaries, data flow, entry points, and privileged code in the application and recognize the **areas of vulnerabilities** among them
- It helps to **conduct attack surface analysis**

- **Decompose application with the help of these activities:**
  - Prepare and Document Threat Model Information
  - Enumerate External Dependencies
  - Enumerate Entry Points
  - Enumerate Assets
  - Enumerate Trust levels
  - Data Flow Diagrams

Prepare and Document Threat Model Information

- The following fields should be **documented** to begin with threat modeling

Application Name	The name of the application
Application Version	The current version of the application
Description	A high level description of the application
Document Owner	The name of the threat modeling document owner
Participants	The people involved in the threat modeling for the application
Reviewer	The name of the reviewer(s) of the threat model

Example: Threat Model Information

Threat Model Information	
Application Version	1.0
Description	The college library website is the first implementation of a website to provide librarians and library patrons (students and college staff) with online services As this is the first implementation of the website, the functionality will be limited. There will be three users of the application: 1. Students 2. Staff 3. Librarians Staff and Students will be able to log in and search for books, and staff members can request books. Librarians will be able to log in, add books, add users, and search for books
Document Owner	John
Participants	Jason
Reviewer	Mathew

Source: *https://www.owasp.org*

Identify the External Dependencies

- External dependency defines the application's dependency on outside entities such as servers, firewalls, security policies, OS, network, etc.
- These entities are the essential components of application but lay outside of the application's code
- Though these entities are outside of the application's control, they are still within the control of the organization
- Identifying such external dependencies may help to minimize the application's overall risk
- To find out external dependencies, analyze different factors on which application security will depend in a production environment
- For example
  - How the application will be deployed?
  - Will application is expected to run on the server?
  - Will there be firewall sitting before server?
- Document such external dependencies with ID and Description field

External Dependencies: Example

External Dependencies	
ID	**Description**
1	The college library website will run on a Linux server running Apache. This server will be hardened as per the college's server hardening standard. This includes the application of the latest operating systems and application security patches
2	The database server will be MySQL and it will run on a Linux server. This server will be hardened as per the college's server hardening standard. This will include the application of the latest operating system and application security patches
3	The connection between the Web Server and the database server will be over a private network
4	The Web Server is behind a firewall and the only communications is TLS

Source: *https://www.owasp.org*

Identify the Entry Points

- Entry points are **interfaces** through which users input data or interact with the application
- A potential attacker may take advantages to such entry points to attack an application
- There can be multilayer entry points in the application; for example, external entry are exposed to the users and internal entry points exposed to subcomponents across the layers of the application
- Attacker can **bypass** first level of entry point and directly attack an internal entry point
- Document all the entry points including multi layer entry points; they should be documented with ID, Name, Description field
- **Typical entry points are:**
  - Login/authentication entry points
  - Admin interfaces
  - Inquiries and search functions
  - Data entry (CRUD) forms
  - Business workflows
  - Transactional interfaces/APIs
  - Operational command and monitoring interfaces/APIs
  - Interfaces with other applications/systems

Entry Points: Example

Entry Points		
ID	**Name**	**Description**
1	HTTPS Port	The college library website will be accessible via TLS. All pages within the college library website are layered on this entry points
1.1	Library Main Page	The splash page for the college library website is the entry point for all users
1.2	Login Page	Students, faculty members and librarians must log in to the college library website before they can carry out of the use cases
1.2.1	Login Function	The login function accepts user supplied credentials and compares them with those in the database
1.3	Search Entry Page	The page used to enter a search query

Source: https://www.owasp.org

Identify the Assets

- Application's asset is **resources/ items/areas** which has a value of attacker's interest
- They are essential **threats targets** which puts the application under risk of being attacked
- These assets can be **physical** and **abstract**
- For example,
  - Physical: data
  - Abstract: reputation, safety
- Assets can interact with other assets
- Identify all the assets of an application which needs to be protected from unauthorized access
- Document both physical and abstract assets of the application with its **ID**, **Name**, and **Description**

Assets: Example

Assets		
ID	**Name**	**Description**
1	Library Users and Librarians	Assets relating to student, faculty members, and librarians
1.1	User Login Details	The login credentials that a student or a faculty member will use to log into the college library website
1.2	Librarians Login Details	The login credentials that a librarian will use to log into the college library website.
1.3	Personal Data	The college library website will store personal information relating to the students, faculty members, and librarians
2	System	Assets relating to the underlying system
2.1	Availability Of College Library Website	The College Library website should be available 24 hrs. a day and can be accessed by all students, college faculty members, and librarians
2.2	Ability To Execute Code as a Web Server User	This is the ability to execute source code on the web server as a web server users

Source: https://www.owasp.org

Assets: Example (Cont'd)

Assets		
ID	**Name**	**Description**
2.3	Ability To Execute SQL as a Database Read User	This is the ability to execute SQL select queries on the database, and thus retrieve any information stored within the College Library database
2.4	Ability To Execute SQL as a Database Read/Write User	This is the ability to execute SQL select, insert, and update quires on the database, and thus have read and write access to any information stored within the College Library database
3	Website	Assets relating to the College Library website
3.1	Login Session	This is the login session of a user to the College Library website. This user could be a student, a member of the college faculty, or a Librarian
3.2	Access to The Database Server	Access to the database server allows you to the administrator the database, giving you full access to the database user and all the data contain within the database
3.3	Ability to Create Users	The ability to create users would allow an individuals to create a new users on the system. These could be students users, faculty member users, and librarians users
3.4	Access to Audit Data	The audit data shows all audit-able events that occurred within the college library application by students, staff, and librarians

Source: https://www.owasp.org

Identify the Trust Levels

- Trust level defines the **access rights** which applications should grant to the external entities

- Define the possible **roles** including set of privileges and trust levels assigned to the roles

- Document the **trust levels** with its ID, Name, and Description filed

Trust Levels: Example

Trust Levels		
ID	**Name**	**Description**
1	Anonymous Web User	A user who has connected to the college library website but has not provided valid credentials
2	User with valid login credentials	A user who has connected to the college library website and logged in using valid credentials
3	User with invalid login credentials	A user who has connected to the college library website and is attempting to log in using invalid credentials
4	Librarian	The librarian can create users on the library website and view their personal information
5	Database Server Administrator	The database server administrator has read and write access to the database that is used by the college library website
6	Website Administrator	The website administrator can configure the college library website
7	Web Server User Process	This is the process/user that the web server executes code as and authenticates itself against the database server as.
8	Database Read User	The database user account used to access the database for read access
9	Database Read/ Write User	The database user account used to access the database for read and write access

Source: https://www.owasp.org

Define Trust Levels to Entry points

- **Define the appropriate trust level required for each identified entry point**

Example:

Entry Points			
ID	Name	Description	Trust Levels
1	HTTPS Post	The college library will be only be accessible via TLS. All pages within the college library website are layered on this entry point	(1) Anonymous Web User (2) User with Valid Login Credentials (3) User with Invalid Login Credentials (4) Librarian
1.1	Library Main Page	The Splash page for the college library website is the entry point for all users	(1) Anonymous Web User (2) User with Valid Login Credentials (3) User with Invalid Login Credentials (4) Librarian
1.2	Login Page	Students, faculty members and librarians must log into the college library website before they can carry out any of the cases	(1) Anonymous Web User (2) User with Valid Login Credentials (3) User with Invalid Login Credentials (4) Librarian
1.2.1	Login Function	The login function accepts user supplied credentials and compares them with those in the database	(2) User with Valid Login Credentials (3) User with Invalid Login Credentials (4) Librarian
1.3	Search Entry Page	The page used to enter a search query	(2) User with Valid Login Credentials (4) Librarian

Source: https://www.owasp.org

Define Trust Levels to Assets

- **Define the appropriate trust level required for each identified asset**

Example:

Assets			
ID	Name	Description	Trust Levels
1	Library Users and Librarian	Assets relating to students, faculty members, and librarians	
1.1	User Login Details	The login credentials that a student or a faculty member will use to log into the College Library website	(2) User with Valid Login Credentials (4) Librarian (5) Database Server Administrator (7) Web Server User Process (8) Database Read User (9) Database Read/Write User
1.2	Librarian Login Details	The login credentials that a Librarian will use to log into the College Library website	(4) Librarian (5) Database Server Administrator (7) Web Server User Process (8) Database Read User (9) Database Read/Write User
1.3	Personal Data	The College Library website will store personal information relating to the students, faculty members, and librarians	(4) Librarian (5) Database Server Administrator (6) Website Administrator (7) Web Server User Process (8) Database Read User (9) Database Read/Write User
2	System	Assets relating to the underlying system	

Source: https://www.owasp.org

Define Trust Levels to Assets (Cont'd)

2.1	Availability of College Library Website	The College Library website should be available 24 hours a day and can be accessed by all students, college faculty members, and librarians	(5) Database Server Administrator (6) Website Administrator
2.2	Ability to Execute Code as a Web Server User	This is the ability to execute source code on the web server as a web server user	(6) Website Administrator (7) Web Server User Process
2.3	Ability to Execute SQL as a Database Read User	This is the ability to execute SQL select queries on the database, and thus retrieve any information stored within the College Library database	(5) Database Server Administrator (8) Database Read User (9) Database Read/Write User
2.4	Ability to Execute SQL as a Database Read/Write User	This is the ability to execute SQL Select, insert, and update queries on the database and thus have read and write access to any information stored within the College Library database	(5) Database Server Administrator (9) Database Read/Write User
3	Website	Assets relating to the College Library website	
3.1	Login Session	This is the login session of a user to the College Library website. This user could be a student, a member of the college faculty, or a Librarian	(2) User with Valid Login Credentials (4) Librarian
3.2	Access to the Database Server	Access to the database server allows you to administer the database, giving you full access to the database users and all data contained within the database	(5) Database Server Administrator
3.3	Ability to Create Users	The ability to create users would allow an individual to create new users on the system. Theses could be students users, faculty member users, and librarian users	(4) Librarian (6) Website Administrator
3.4	Access to Audit Data	The audit data shows all audit-able events that occurred within the College Library application by students, staff, and librarians	(6) Website Administrator

Source: https://www.owasp.org

Perform Application Modeling using Data Flow Diagrams (DFDs)

- The collected information such as **external entities, entry points, assets and trust level** will help in accurately modeling the application by using Data Flow Diagrams (DFDs)
- DFD will help in understanding how the data flows through the application and what happens to data while flowing
- Start with higher level DFD and go on creating lower DFDs to decompose the application into different process and their lower-level sub processes or functions
- The higher level DFD will help you understand overall the scope of the application while lower level DFDs will help you understand working of lower-level specific processes

Perform Application Modeling using Data Flow Diagrams (DFDs) (Cont'd)

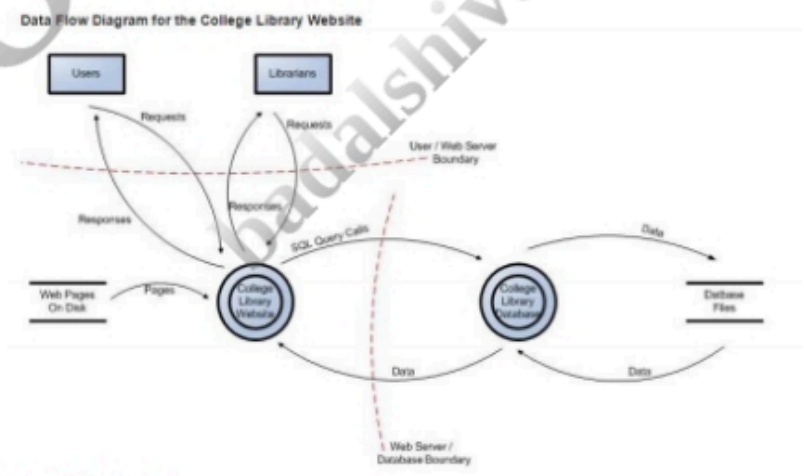
Different Components in DFD

External Entity — It is used to represents external entity which **interacts** with the **application from outside**	Process — It is used to represents **actions** or **function** to accomplish specific task
Multiple Process — It is used to represents **collection of sub processes**	Data Store — It is used to represents **locations** where data is stored
Data Flow — It is used to represents the **movement of data** within the application	Privilege Boundary — It is used to represents **change of privilege levels**

Perform Application Modeling using Data Flow Diagrams (DFDs) (Cont'd)

- **Create Context level DFD to map the scope of the application**

Example:

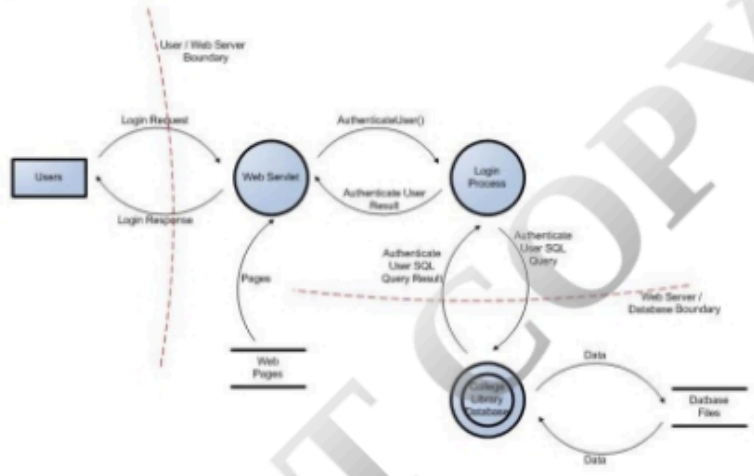


Source: https://www.owasp.org

Perform Application Modeling using Data Flow Diagrams (DFDs) (Cont'd)

- **Go on creating lower level DFD to depict specific process**

Example:



Source: https://www.owasp.org

Determine the Threats: Identify the Goal of an Attacker and Create Threat Profile

- Once the DFD's are **created at lowest level**, you can identify all the external dependencies, entry/exits points, trust levels in an application
- It helps to easily identify what data need to be provided to specific process in DFD and corresponding **attacker's goal** to exploit the functionality of specific process
- This goal is used in DFD to determine the **threat path** in the application
- Identify all the possible **attacker's goals** and **vulnerabilities** which allows attackers to accomplish their goals

Example: Attacker's Goal/Threat Profile and Vulnerabilities Associated

Example #1:

Goal: Attacker may steal user authentication credentials

Vulnerability:

- Insecure storage of credential at the client system
- Credentials are sent with GET parameters

Example #2:

Goal: Attacker may login with brute force attempt

Vulnerability :

- Use of Improper input validation and weak password

Example#3:

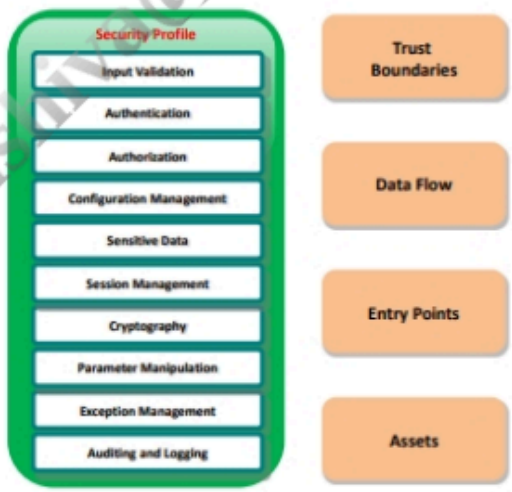
Goal: Attacker may launch a denial of service attack

Vulnerability:

- Lack of account lockout functionality
- Absence of CAPTCHA

Determine the Threats: Create a Security Profile

- To create **security profile** for an application, focus on:
  - Various **design** and **implementation approaches** of an application which can be most susceptible to vulnerabilities
  - Various **areas of applications** which can be most susceptible to vulnerabilities
- Document the **security profile** for application depending on the area of vulnerabilities



Determine the Threats: Create a Security Profile (Cont'd)

- **Following considerations help you to identify vulnerabilities in design and implementation of your application**

Category	Considerations
Input Validation	Is all input data validated? Could an attacker inject malicious commands or data in the input fields? Is data validated as it is passed between separate trust boundaries (by the recipient entry point)? Can data in the database be trusted?
Authentication	Are credentials secured if they are passed over the network? Are strong account policies used? Are strong passwords enforced? Are you using certificates? Are password verifiers (using one-way hashes) used for user passwords?
Authorization	What gatekeepers are used at the entry points of the application? How is authorization enforced at the database? Is defense in depth strategy used? Do you fail securely and only allow access upon successful confirmation of credentials?
Configuration Management	What administration interfaces does the application support? How are they secured? How is remote administration secured? What configuration stores are used and how are they secured?
Sensitive Data	What sensitive data is handled by the application? How is it secured over the network and in persistent stores? What type of encryption is used and how are encryption keys secured?

Determine the Threats: Create a Security Profile (Cont'd)

Category	Considerations
Session Management	How are session cookies generated? How are they secured to prevent session hijacking? How is persistent session state secured? How is session state secured as it crosses the network? How does the application authenticate users with the session store? Are credentials passed transmitted and are they maintained by the application? If so, how are they secured?
Cryptography	What algorithms and cryptographic techniques are used? How long are encryption keys and how are they secured? Does the application put its own encryption into action? How often are keys recycled?
Parameter Manipulation	Does the application detect tampered parameters? Does it validate all parameters in form fields, ViewState, cookie data, and HTTP headers?
Exception Management	How does the application performs error handling? Are exceptions ever allowed to propagate back to the client? Are generic error messages that do not contain exploitable information used?
Auditing and Logging	Does your application audit activity across all tiers on all servers? How are log files secured?

Identify the Threats

1. To identify threats, they need to be broadly categorized into **different possible categories**
2. This threat categorization provides structured way to **systematically identify different threats** on the application
3. Threat identification process uses **STRIDE model** to categorize threats and examine each aspect of the security profile of the application

Identify the Threats: The STRIDE Model

- The STRIDE model categorizes the application threats based on the **goals** and **purpose of the attack** which helps developer to **develop a security strategy**
- It also includes the **countermeasures** for all threat categories
- **This model describes the following threat categories:**

Desired Property	Threats	Description
Authentication	Spoofing	Unauthorized access to a system by using a false identity
Integrity	Tampering	Code and data modification without authorization
Non-repudiation	Repudiation	Ability of users to claim that they do not perform some actions against the application
Confidentiality	Information disclosure	Unwanted exposure of private data to unauthorized users
Availability	Denial of service	Ability to deny a system or application unavailable to the legitimate users
Authorization	Privilege escalation	Ability of user with limited privileges to elevate their privileges with an application without authorization

Source: https://www.owasp.org

Example: Threat Categorized and Identified using STRIDE

STRIDE Threat List

Type	Examples
Spoofing	Threat action aimed to illegally access and use another user's credentials, such as username and password
Tampering	Threat action aimed to maliciously change/modify persistent data, such as persistent data in a database, and the alteration of data in transit between two computers over an open network, such as the internet
Repudiation	Threat action aimed to perform illegal operations in a system that lacks the ability to trace the prohibited operations
Information disclosure	Threat action to read a file that one was not granted access to, or to read data in transit
Denial of Service	Threat aimed to deny access to valid users, such as by making a web server temporarily unavailable or unusable
Elevation of privilege	Threat aimed to gain privilege access to resources for gaining unauthorized access to information or to compromise a system

Source: https://www.owasp.org

Determine Countermeasures and Mitigation Security Controls

- **Identify the set of controls and countermeasures to prevent identified threat**

Desired Property	Threats	Countermeasures/Controls
Authentication	Spoofing	Use strong authentication mechanisms Avoid storing credentials in clear text Do not pass sensitive data in clear text over networks
Integrity	Tampering	Use strong authentication mechanisms Secure authentication tokens with encrypted communication channels Use digital signature
Non-repudiation	Repudiation	Identify malicious behavior Create secure audit trails Use digital signature
Confidentiality	Information disclosure	Use strong authorization and encryption mechanism Avoid storing credentials in clear text
Availability	Denial of service	Use bandwidth controlling techniques
Authorization	Privilege escalation	Use limited privileged service accounts

Source: https://www.owasp.org

Document the Threats

- A standard template that describes several **threat attributes** including threat description and threat target are used to document the threat
- The **risk rate** is left blank as it is to be filled in during the final risk rating step

A Sample Template for Documenting the Threats

Threat Description	Attacker obtains Authentication Credentials by Monitoring the Network
Threat Target	Web application user authentication process
Risk	
Attack Techniques	Use of network monitoring software
Countermeasures	Use SSL to provide encrypted channel

Rating the Threats

1. This step rates the threats based on the **risks they possess**
2. Threat resolution is prioritized based on the **risk ratings**
3. The formula for calculating risk is:
   RISK = PROBABILITY * DAMAGE POTENTIAL

Rating the Threats: DREAD Model

DREAD Model

- DREAD stands for damage potential, reproducibility, exploitability, affected users, and discoverability
- DREAD model is used to rate the various security threats on the application by **calculating risks of each threats**
- The severity levels are assigned to various threats on the application so as to **mitigate these threats as per their severity**

	Rating	High (3)	Medium (2)	Low (1)
D	Damage Potential	The attacker can subvert the security system; get full trust authorization run as administrator; upload content	Leaking sensitive information	Leaking trivial information
R	Reproducibility	The attack can be produced every time and does not require a timing window	The attack can be reproduced, but only with a timing window and a particular race situation	The attack is very difficult to reproduce even with knowledge of the security hole
E	Exploitability	A novice programmer could make the attack in a short time	A skilled programmer could make the attack then repeat the steps	The attacker requires an extremely skilled person and in-depth knowledge every time to exploit
A	Affected Users	All users, default configuration key customers	Some users, non-default configuration	Very small percentage of users, obscure feature affects anonymous users
D	Discoverability	Published information explains the attack. The vulnerability is found in the most commonly used feature and is very noticeable	The vulnerability in a seldom-used part of the product and only a few users should come across it. It would take some thinking to see malicious use	The bug is obscure and it is unlikely that users will work out damage potential

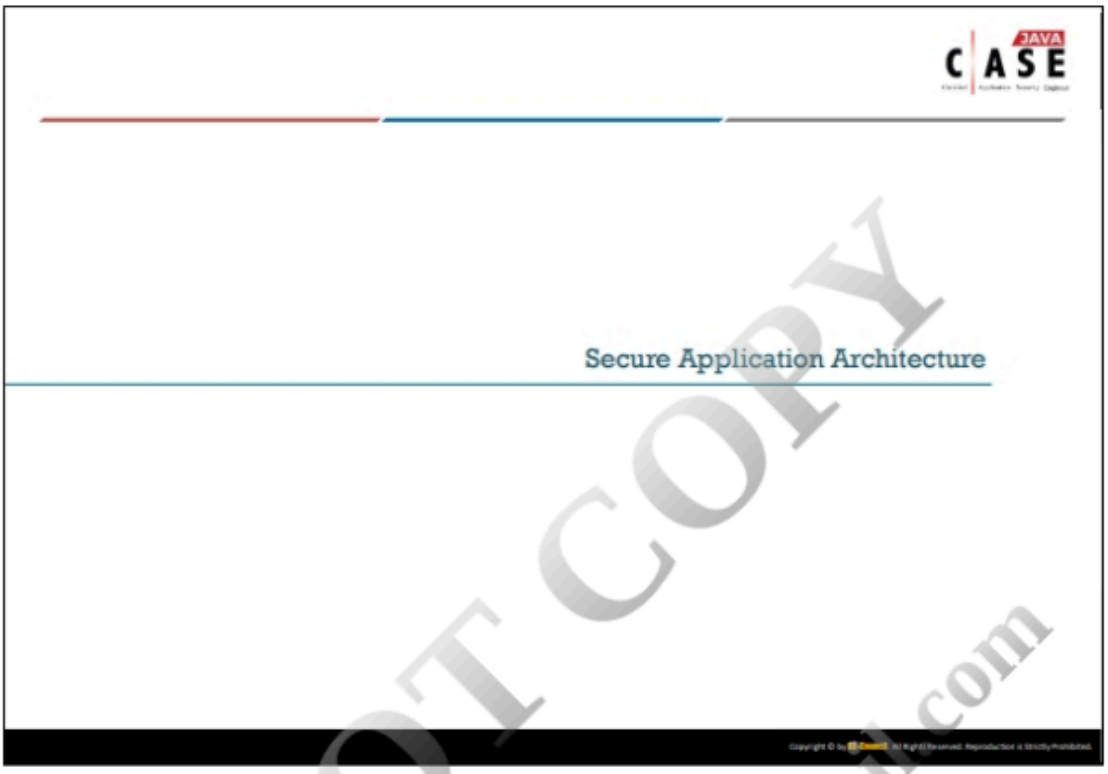
Rating the Threats: DREAD Model (Cont'd)

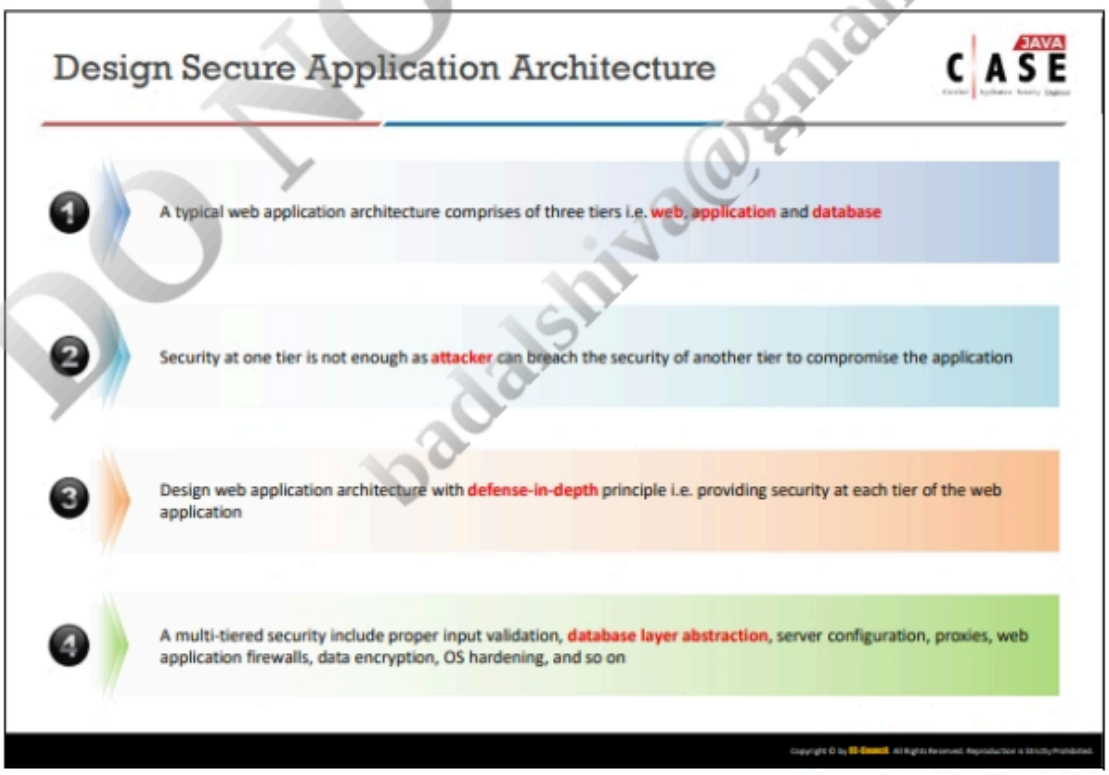
Sample DREAD Rating Table

Threat	D	R	E	A	D	Total	Rating
Attacker obtains authentication credentials by monitoring the network	3	3	2	2	2	12	High
SQL commands injected into application	3	3	3	3	2	14	High

A Sample Template for Documenting the Threats after Risk Rating

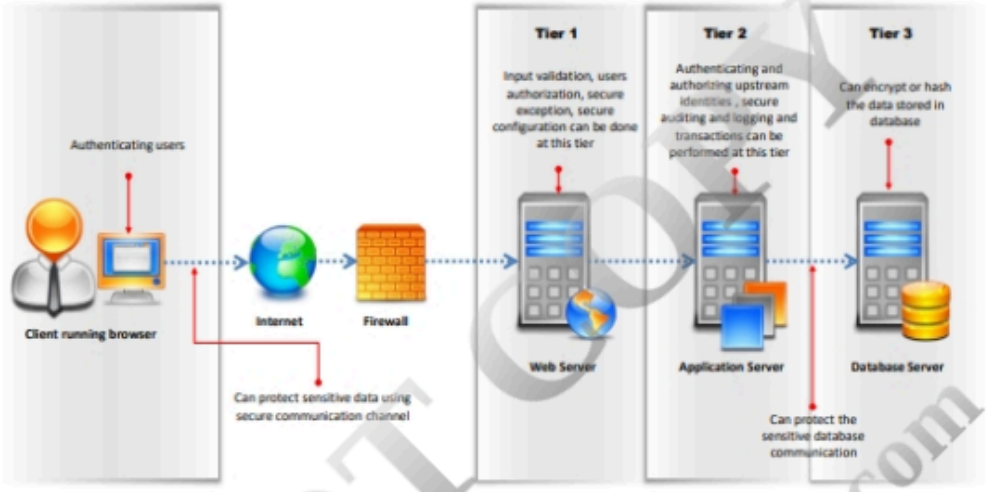
Threat Description	Attacker obtains Authentication Credentials by Monitoring the Network
Threat target	Web application user authentication process
Risk rating	High
Attack techniques	Use of network monitoring software
Countermeasures	Use SSL to provide encrypted channel





Design Secure Application Architecture (Cont'd)

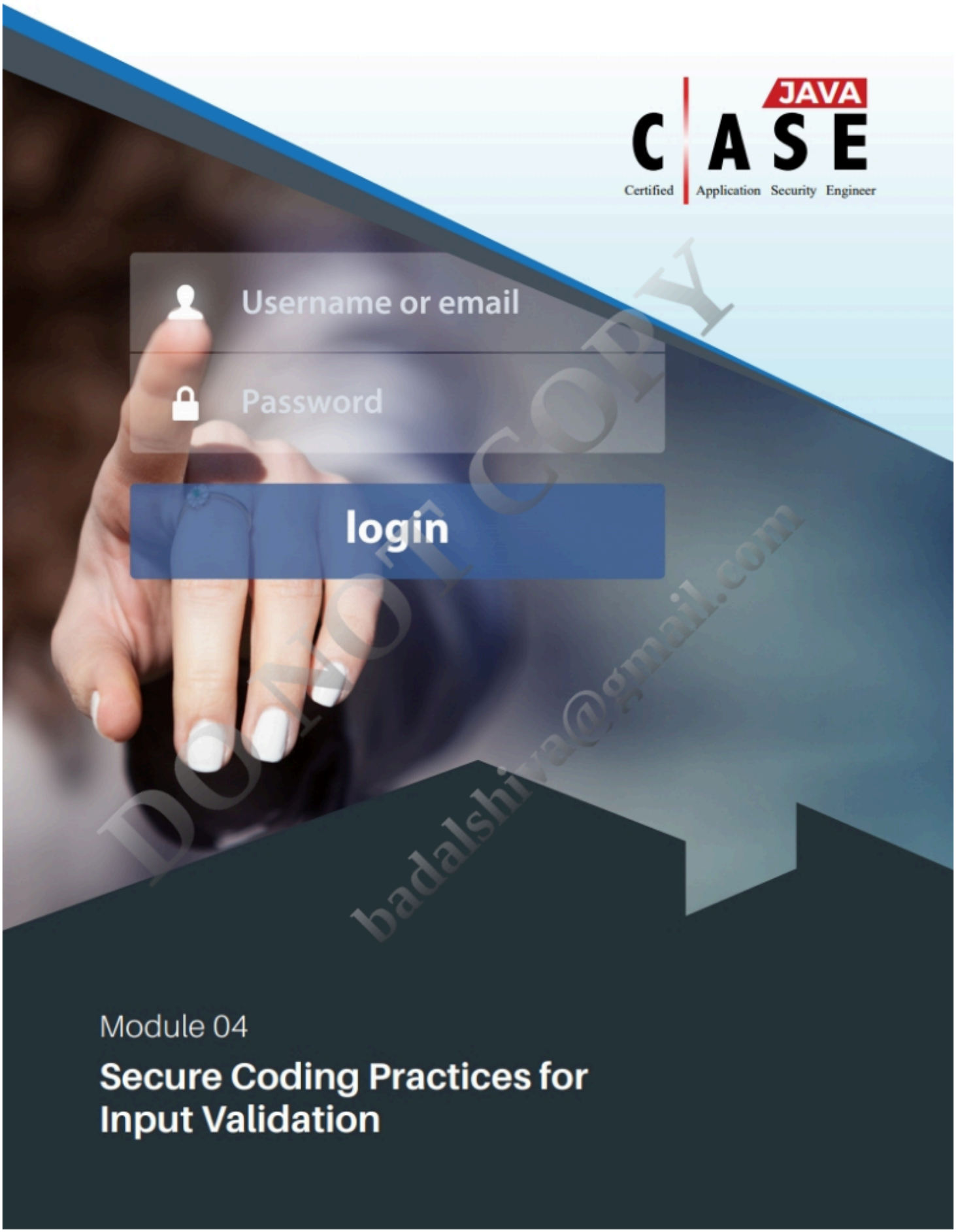
- **Applying multiple layer security in application architecture design makes application robust and secure**



Module Summary

- Security negligence at design and architecture phase may lead to vulnerabilities that are most difficult to detect and expensive to fix in production
- Secure design of an application is more straightforward once the security requirements are identified
- Secure design principles are the state of practices or guidelines which should be enforced on the developers to follow during development phase
- Threat modeling is a process of identifying, analyzing, and mitigating the threats to the application
- A multiple tired security include proper input validation, database layer abstraction, server configuration, proxies, web application firewalls, data encryption, OS hardening, etc.

This page is intentionally left blank.

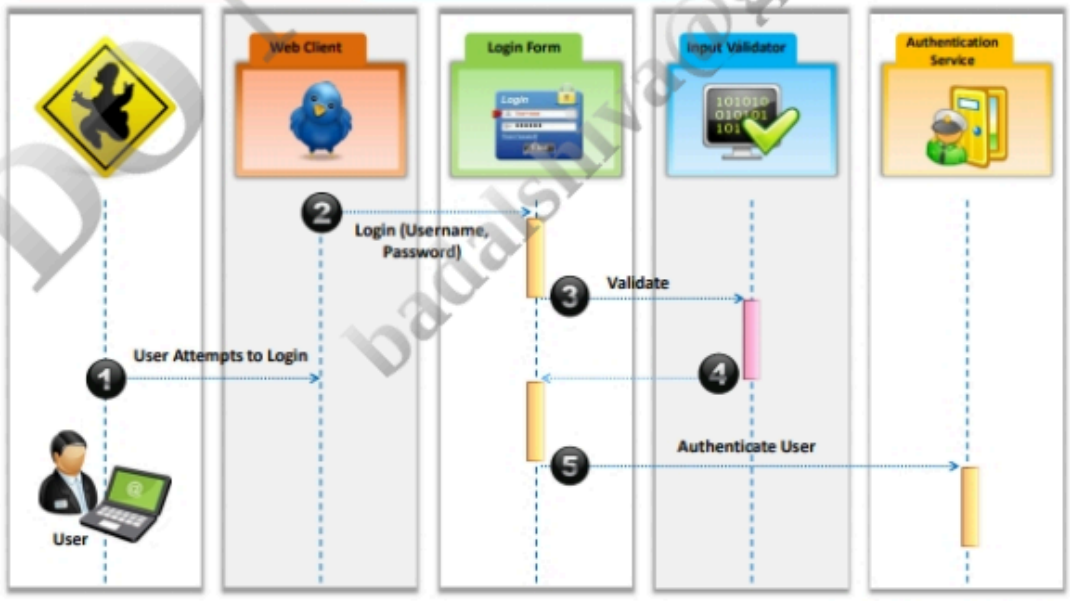


This page is intentionally left blank.

Module Objectives

1. Understand the Need of Input Validation
2. Discuss Data Validation Techniques
3. Discuss Data Validation in Strut Framework
4. Discuss Data Validation in Spring Framework
5. Discuss Common Input Validation Errors
6. Learn Common Secure Coding Practices for Input Validation

Input Validation Pattern



Validation and Security Issues

- Humans intentionally or unintentionally give invalid inputs that can pose computer **security risks** to applications and organizations
- A web application surely needs to check various **mandatory application fields** for correct expected format else it results in altered control flow, arbitrary control of a resource, or arbitrary code execution
- Java **action forms** provide input data validation in the source code, providing error message in case of invalid data input

Impact of Invalid Data Input

- Input data validation is most important for any software application because of the dangers posed by **tainted input** at various levels of its execution
- An application that does not validate the input data is prone to **risks** that are **critical** to application security

Risks of Improper Data Validation

Availability (Technical Impact)

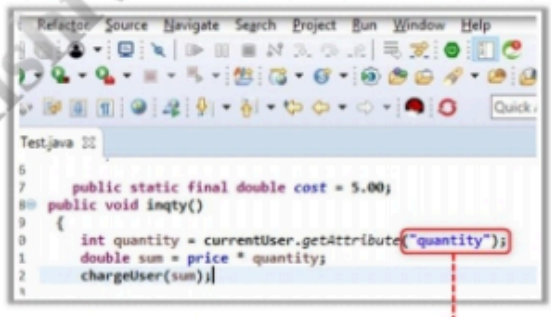
- A system can be DOS attacked, **crashed**, **exited**, **restarted** or resources like CPU, memory are consumed

Confidentiality

- Attackers can steal, read or change the data such as **memory**, **files** and **directories**

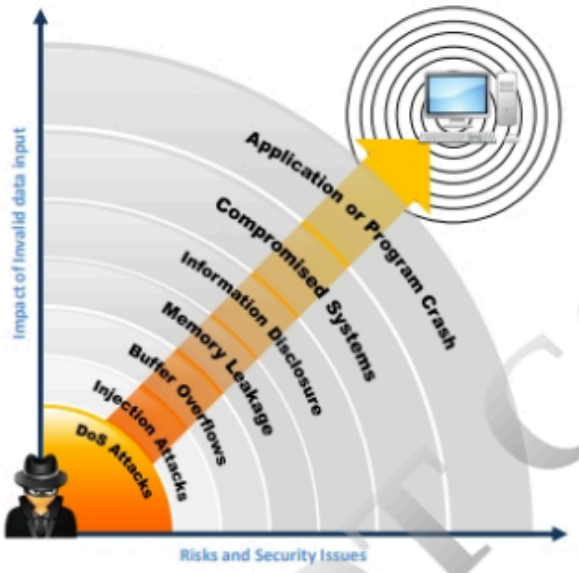
Integrity

- Hackers can illegitimately modify the **control flow**, **execute arbitrary commands**, and modify the data and memory



In this code, an e-commerce application price of a particular article is constant, where as the quantity is not; a hacker can even insert negative values for the quantity resulting in crediting his account instead of debiting on buying the good

Impact of Invalid Data Input (Cont'd)



Target	Type of Attacks
Browsers	HTML Splitting, XFS, XSS
Data Repositories	LDAP Injection, SQL Injection
Server Attacks	Xpath Injection, XML Injection, XXE Injection
Application, Server and OS Attacks	Buffer Overflows and File uploads

Data Validation Techniques

1. Data validation techniques include various methods used in **preventing attackers** from taking advantage of the vulnerabilities arising due to **invalid inputs**
2. Canonicalization of input or encoding of the input that can be easily interpreted by the system is always **safe**

Character set that mostly cause trouble are:

- Characters that have special meaning
- Normal control characters (values less than 32)
- Characters with values higher than 127
- Metacharacters

Data Validation Techniques (Cont'd)

Data validation techniques include some of the basic general strategies such as:

Encode Known Bad

- Encoding the known bad entries within the code

Accepting Exact Match

- Accepting a finite list of only known values

Reject Known Bad

- Do not let known bad entries to be entered into the system

Known Good

- If a finite list is not available, then use known good approach

Input Validation using Frameworks and APIs

Java Framework

- Java framework allows the control of user behavior in Java and helps save time by eliminating the creation of **test scripts**
- This is possible by embedding Java source code into the framework

Java API

- Java API contains a **set of classes** in Java development environment
- It allows programmers to perform common tasks such as file transfer, networking, and data structures

Need for Input Validation

- Every application should be designed in such a way that it can ensure by itself that only valid data is inserted through **predetermined parameters**
- Any attempt to insert or update malicious data should be **detected** and **rejected** by the application

Open Source Validation Framework for Java

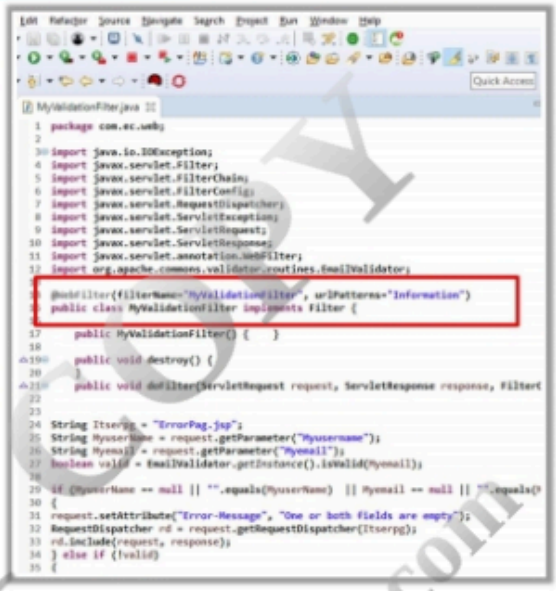
- **Following are commonly used data validation libraries/framework**
  - **Commons Validator**: Supports both server side and client side validation
  - **spring-modules-validation**: It contains many tools and add-ons and it is based on validator specification
  - **OVal**: It validates Java objects as per user request. It checks the parameters on calling the constructor
  - **iScreen**: It validates Java objects via configuration. It is used for validating Java beans
  - **jReform**: It is mainly used for HTML form and it automatically validates inputs
  - **JValidate**: This framework supports Java 1.4 and above. It implements validations with annotations
  - **JValidations**: Validation is done by the object itself without any external classes

Servlet Filters

- Input validation through servlet filters in Java web application is effective due to **minor modifications** needed for input validation and servlets filters are **centralized** in nature
- Input validation in servlets is through **multipart encoded content** by handling multipart requests
- Applications should not rely only on one form of defense, i.e. servlet filters alone; **security** is a multi layered approach
- Modern frameworks facilitate input validation in the application itself
- Servlets can be configured on an application if it does not require multipart requests where **automatic multipart requests** should be disabled

Validation Filters for Servlet

- The validation filter is executed before processing the request
- Validation filters **Error-Message** attribute stores error messages and displays the same when validation fails
- Filter mapping can be done with the help of **@webFilter** annotation
- @webFilter when applied on a url, the filter is invoked before invocation of url pattern eg. **/Information**
- If the data fails to validate then it redirects to the **error page** specified using **RequestDispatcher**



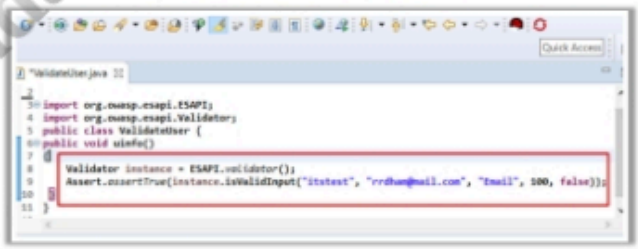
```
MyValidationFilter.java
1  package com.ec.web;
2
3  import java.io.IOException;
4  import javax.servlet.Filter;
5  import javax.servlet.FilterChain;
6  import javax.servlet.FilterConfig;
7  import javax.servlet.RequestDispatcher;
8  import javax.servlet.ServletException;
9  import javax.servlet.ServletRequest;
10 import javax.servlet.ServletResponse;
11 import javax.servlet.annotation.WebFilter;
12 import org.apache.commons.validator.routines.EmailValidator;
13
14 @WebFilter(filterName="MyValidationFilter", urlPatterns="Information")
15 public class MyValidationFilter implements Filter {
16
17     public MyValidationFilter() {    }
18
19     public void destroy() {
20     }
21     public void doFilter(ServletRequest request, ServletResponse response, FilterC
22
23
24 String Itserpg = "ErrorPag.jsp";
25 String Myusername = request.getParameter("Myusername");
26 String Myemail = request.getParameter("Myemail");
27 boolean valid = EmailValidator.getInstance().isValid(Myemail);
28
29 if (MyuserName == null || "".equals(MyuserName)  || Myemail == null || "".equals(M
30 {
31 request.setAttribute("Error-Message", "One or both fields are empty");
32 RequestDispatcher rd = request.getRequestDispatcher(Itserpg);
33 rd.include(request, response);
34 } else if (!valid)
35 {
```


Data Validation using OWASP ESAPI

- OWASP ESAPI is free open source API for securing web applications
- It can be used to **implement** and **enhance** security of existing applications
- It contains:
  - A set of security control interfaces
  - Reference implementation for security control
  - Custom implementations for each security control

- Package **org.owasp.esapi.reference.validation** provides following **data validation classes** :
  - **BaseValidationRule**: Validating untrusted source
  - **CreditCardValidationRule**: Validating credit card no.
  - **DateValidationRule**: Validating date
  - **HTMLValidationRule**: Validating HTML to protect from XSS attack
  - **IntegerValidationRule**: Validating integers
  - **NumberValidationRule**: Validating number input format
  - **StringValidationRule**: Validating strings

Example : Input Validation using OWASP ESAPI



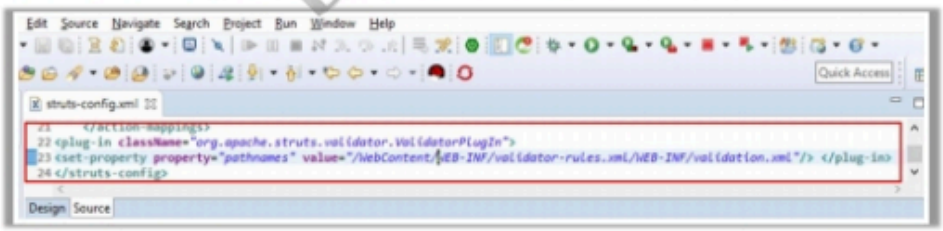
```
ValidateUser.java
3
4  import org.owasp.esapi.ESAPI;
5  import org.owasp.esapi.Validator;
6  public class ValidateUser {
7  public void userFn()
8  {
9      Validator instance = ESAPI.validator();
10     Assert.assertTrue(instance.isValidInput("itstest", "rrdhan@mail.com", "Email", 100, false));
11 }
12 }
```

Data Validation: Struts Framework

Struts Validator

- The Struts validator framework is used to validate input on both the user browser and the server side
- The Struts framework makes use of a utility called commons validator for validating input
- The common validator gives the user a central area for data validation

The validator plug-in is mapped in the struts-config.xml file as follows:



Struts Validation and Security

I. Struts validation is done to prevent attacks caused through **unchecked input**

II. All the inputs should be checked using the **struts validator** before it is processed by the application

III. Each and every field included in the form should be validated in the **corresponding validation form**

IV. The absence of validation for a **single field** may allow attackers to exploit the application

V. Vulnerabilities in input validation may lead to **buffer overflow attacks, cross-site scripting, SQL injection**, etc.

Data Validation using Struts Validator

- **The following principles should be implemented in code when validating data using the Struts Validator**
  - Avoid **duplicate validation** forms in the validation xml file
  - Implement **struts validator** class
  - Enable the struts validator in the action form mapping
  - Check for similar number of fields in **action form** and **validation form**

Avoid Duplication of Validation Forms

Validation forms in struts are used to **validate** the **input fields**

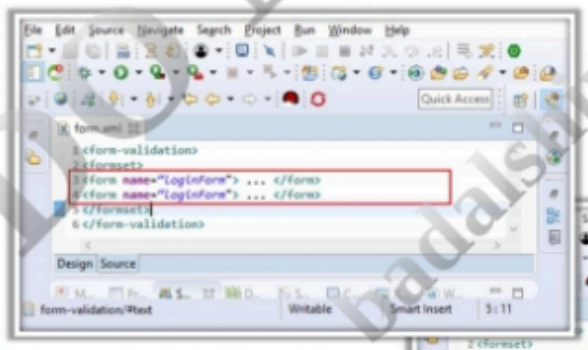
Validation forms with the same name should be avoided when mapping in validation xml files

If two forms are given the same name, it indicates that the validation logic is incorrect

When similar names are given, the **struts validator** selects one form in a random manner and discards the other form

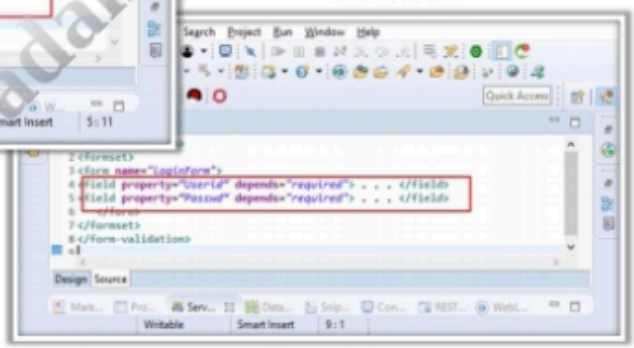
Secure and Insecure Struts Validation Code

Vulnerable Code



- The vulnerable code should be avoided in **validation** of **xml file**

Secure Code



- The code is secure as it does not **duplicate form names**

Struts Validator Class

- The **struts form** should extend one of the following validator classes to implement struts validator
  - ValidatorForm
  - ValidatorActionForm
  - DynaValidatorActionForm
  - DynaValidatorForm
- The method **validate()** can be implemented in the user's application through the use of these classes
- The validation of form fields are done under the **validate()** method
- The validator form that defines a **validate()** method should call **super.validate()** function
- The validator forms derived from **ActionForm** and **DynaActionForm** classes cannot use the struts validator

Secure and Insecure Code for Struts Validator Class

Vulnerable Code

```
package com.valila.form;

import javax.servlet.http.HttpServletRequest;

import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionMessage;
import org.apache.struts.validator.ValidatorForm;

public class MyLoginForm extends ValidatorForm {
    /**
     *
     */
    private static final long serialVersionUID = 1L;
    private String empname;
    public String getEmpname() {
        return empname;
    }
    public void setEmpname(String empname) {
        this.empname = empname;
    }
    public String getEmpemail_id() {
        return empemail_id;
    }
    public void setEmpemail_id(String empemail_id) {
        this.empemail_id = empemail_id;
    }
    private String empemail_id;

    public MyLoginForm()
    {
        super();
    }
    public ActionErrors validate(ActionMapping map, HttpServletRequest request) {
        ActionErrors act_err = new ActionErrors();
        if (getEmpname() == null || getEmpname().length() < 1) {
        act_err.add("name", new ActionMessage("error.name.required"));
        }
        return act_err;
        }
```

Secure Code

```
package com.valila.form;

import javax.servlet.http.HttpServletRequest;

import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionMessage;
import org.apache.struts.validator.ValidatorForm;

public class MyLoginForm extends ValidatorForm {
    /**
     *
     */
    private static final long serialVersionUID = 1L;
    private String empname;
    public String getEmpname() {
        return empname;
    }
    public void setEmpname(String empname) {
        this.empname = empname;
    }
    public String getEmpemail_id() {
        return empemail_id;
    }
    public void setEmpemail_id(String empemail_id) {
        this.empemail_id = empemail_id;
    }
    private String empemail_id;

    public MyLoginForm()
    {
        super();
    }
    public ActionErrors validate(ActionMapping map, HttpServletRequest request) {
        ActionErrors act_err = super.validate(map, request);
        if (getEmpname() == null || getEmpname().length() < 1) {
        act_err.add("name", new ActionMessage("error.name.required"));
        }
        return act_err;
        }
```

- The code is vulnerable as it does not call **super.validate()** function
- The code calls **super.validate()** function