22/8/25

ASSIGNMENT 2

Unit-2&3 What Makes Software Secure? & Requirements Engineering for Secure Software, Secure Software Architecture and Design.

Q.1 Define and explain core properties of secure software.

-4 Secure software is designed and implemented in such a way that it protects data, resources and functionality from authorized access, misuse and malicious attacks.

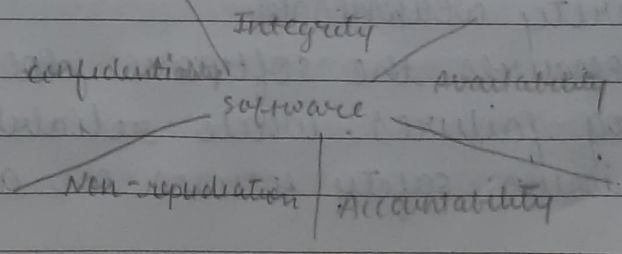
① Confidentiality: This property ensures that the software's characteristics, assets and data are hidden from unauthorized access. For example, an SQL injection that exposes personal data violates confidentiality.

② Integrity: Refers to the software and its data being protected from unauthorized modifications or tampering. A cross-site scripting (XSS) attack can compromise system integrity.

③ Availability: Ensures the software remains accessible to authorized users when needed, while blocking access to unauthorized users.

④ Accountability: All security-relevant actions must be recorded and traceable to their source, both during and after they occur.

⑤ Non-repudiation: Prevents denial of actions performed by the software of its users, reinforcing accountability.



* For diagram, Refer Chapter 2 Book, Pg No: 3

Q.2 Explain influential properties of secure software.

—↓ These properties shape the overall strength, resilience and trustworthiness of software in real-world environments. They act as supporting factors to ensure the system remains secure throughout its lifecycle.

① Dependability: Ensures software consistently work as intended. It supports integrity and availability, sharing traits like reliability, safety and fault-tolerance, though it doesn't guarantee confidentiality or accountability.

② Correctness: Means the software functions properly under expected conditions. For security, it must also behave correctly under unexpected conditions, such as during attacks — something traditional engineering overlooks.

③ Predictability: Ensures the software behaves as expected, even in abnormal or hostile conditions. This is achieved by monomizing vulnerabilities and blocking malicious code.

④ Reliability: It is the software's ability to function correctly under normal conditions, even with minor faults or changes. It reduces the risk of crashes or failures that could expose security weaknesses.

⑤ Safety: It ensures the software doesn't cause harm during failures. While reliability handles accidental issues, safety focuses on avoiding

* For diagram, refer to Chapter 2, Book, Pg No: 5

dangerous outcomes, including those triggered by malicious actions.

Q.3 Discuss the importance of threat modeling in the software development process. How does it help in enhancing security?

→ Threat modeling is a structured approach used to identify, assess and mitigate potential security threats early in the software development lifecycle. It helps developers and security teams understand how attackers might exploit a system, allowing them to build more secure software from the ground up.

① Early Identifications of Security Risks: Performed during the design phase, it helps uncover vulnerabilities before development begins. Early detection reduces the cost and effort of fixing issues later.

② Understanding the Attacker's Perspective: It helps developers think like attackers, identifying how components might be exploited. This reveals hidden attack surfaces and misuse cases.

③ Prioritizing Security Efforts: By evaluating the impact and likelihood of threats, teams can focus on the most critical risks, ensuring efficient use of security resources.

④ Improving Architecture and Design: It highlights design flaws early, promoting secure coding practices, safe data flows and proper access controls

from the start.

⑤ Enhancing Team communication : It offers a shared framework for developers and security teams, ensuring everyone understands the system's risks and their roles in mitigating them.

- Ways it Enhances Security :-
  ① Proactively identifies threats before code is written.
  ② Reduces vulnerabilities through secure design.
  ③ Helps in selecting appropriate security controls.
  ④ Ensures continuous risk assessment throughout SDLC
  ⑤ Leads to software that is resilient against attacks.

Q.4 Justify the statement "Small Faults, Big Consequences."

→ The statement underscores how minor flaws in software design, coding or configuration can lead to serious security breaches, causing financial loss, reputational harm or risks to human safety.

① Exploitable Minor Vulnerabilities : Small issues like weak passwords, improper input validation or default settings can open doors to larger attacks such as SQL injection or remote code execution.

② Escalation Through Chaining : Attackers often combine multiple small flaws to escalate

privileges, access sensitive data or bypass protections.

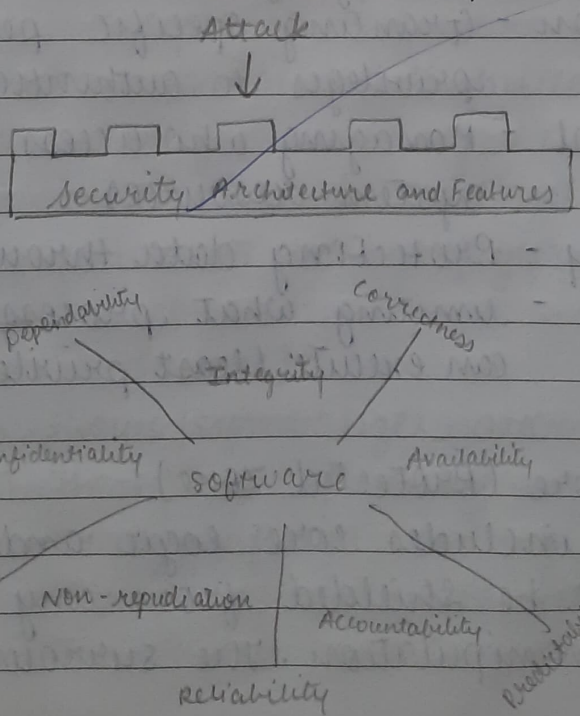
(3) Violation of Core Principles: Small faults can compromise confidentiality, integrity ~~or~~ and availability.

(4) High cost of Small Mistakes: Breaches caused by minor bugs can lead to financial loss, legal issues, reputational harm and loss of user trust — often costing far more than prevention.

eg: The Equifax Data Breach (2017) is a example where a single, unpatched software vulnerability in Apache Struts led to a catastrophic leak of personal data for 147 million people, providing that a small fault can have a huge impact.

Q.5 Draw and explain the diagram of addressing expected issues with security architecture and features.

Ans



*For diagram, refer to chapter 2 Book, Pg No: 15

A layered, fortress-like model protects core functionality by using security architecture and features to block common threats before they can cause harm.

① Attack (External Threat Vector)

It represents potential security threats targeting the system like Malware, Brute-force attacks, unauthorized access, exploitations of vulnerabilities. These are common, expected threats that every software system must defend against.

② Security Architecture and Features (Protective Wall).

It represents the various security mechanisms that are designed into the systems to block or mitigate known and expected attacks.

i) Authentication - Verifying the identity of a user.
ii) Authorization - Granting specific permissions and privileges to authenticated users.
iii) Access Control - Managing who access and use system resources.
iv) Cryptography - Protecting data through encryption.
v) Permission / Privileges - Limiting what processes or users can execute (least privilege principle).

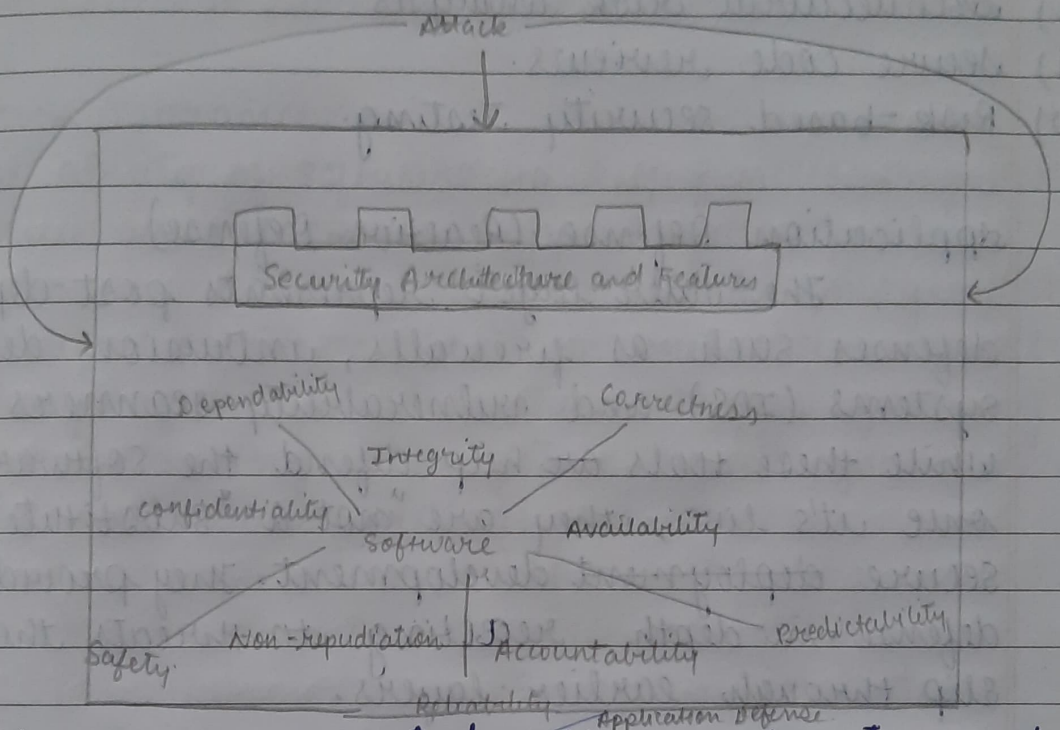
③ Software Core (Protected zone)

It includes core logic and data. This layer must be shielded from any unauthorized access or manipulation. The surrounding security

architecture ensures only safe, authorized interactions reach this level.

Q.6 Draw and explain the diagram of Addressing the unexpected through software security.

-4



Addressing unexpected software threats needs a multi-layered approach that builds security into the software itself, ensuring it remains resistant, tolerant and resilient – even against unforseen flaws or misconfigurations.

① Attack (Threat Source).

It shows/symbolizes attempts to exploit weaknesses. These are often ~~by~~ unexpected threats caused by design flaws, coding errors and misconfigurations. Software security practices ~~to~~ aim to prevent such threats from reaching the deployed application.

* For diagram, refer to chapter 2 Book, Pg No: 17

② Software Security (Proactive Defense)

This inner layer surrounds the software core and focuses on preventing vulnerabilities during development. It includes:

i) Security requirements (eg., misuse/abuse cases).
ii) Architectural risk analysis
iii) Secure code reviews.
iv) Risk-based security testing.

③ Application Defense (Reactive Defense)

The outer layer represents post-deployment defenses such as firewalls, intrusion detection systems (IDS) and vulnerability scanners. While these tools help defend the software once it's live, they are "not a "substitute" for secure development. They provide defense-in-depth, reacting to threats that slip through earlier layers.

Q.7 Discuss Attack Resistance, Attack Tolerance and Attack Resilience.

→ These three important concepts — describe the system's design to withstand, adapt to and recover from security attacks.

1. Attack Resistance.

The ability of software to prevent attacks from succeeding by eliminating common vulnerabilities.

It focuses on keeping threats out through secure coding, strong authentication and

encryption. What makes it distinct is its "proactive nature" - it tries to stop attacks before they can affect the system."

eg: A Banking app uses multi-factor authentication and input validation to block brute-force and injection attacks.

2. Attack Tolerance.

The ability of software to keep functioning even while an attack is happening.

It assumes some attacks may break through, so the goal is to limit their impact. Through ~~that~~ fault isolation, traffic filtering or load balancing, the system avoids full failure and continuous critical operations.

eg: During a DDOS attack, traffic is rerouted to backup servers to maintain service availability.

3. Attack Resilience.

The software's ability to recover quickly after an attack, restoring normal operations with minimal disruption.

It focuses on recovery - using backups, automated response systems and monitoring to return to a safe state. Unlike others, it's about restoring and adapting after an incident.

eg: After a ransomware attack, data is restored from secure backups and the system resumes normal operation.

Q.8 Describe The Attacker's Perspective with its advantages.

→ The Attacker's ~~Perp~~ Perspective involves thinking like an attacker to identify and exploit software vulnerabilities. By focusing on how systems can be misused - rather than just their intended functions - security teams can better anticipate and defend against real-world threats.

- Advantages :-

① Identifies Hidden Weaknesses : Viewing the system like an attacker helps reveal/uncover unexpected paths or misuse cases, developers might miss.

② Improves Threat Modeling : It strengthens the quality of threat modeling by focusing on how a system can fail, not just how it should work.

③ Enhances Secure Design : Informs better and stronger design choices by revealing potential attack vectors.

④ Boosts Testing Effectiveness : Enables realistic, targeted security testing based on actual attack scenarios.

⑤ Reduces Risk Early : By anticipating how a system could be exploited, vulnerabilities can be caught and fixed early in the development cycle, saving time and cost.

eg: If developers adopt the attacker's perspective while building an e-commerce site, they might realize :
- weak password storage could allow credential theft.
- SQL queries without input validation could expose the database.

session hijacking risks require secure cookie management. So addressing these in advance reduces the probability of a data breach.

Q.9 How to Assert and Specify Desired Security Properties?
Ans To develop secure software, it is essential to "assert" (state clearly) and "specify" (formally define) the required security properties of the systems.

1. Identify Security Goals: Define what the system must protect like confidentiality, integrity, availability (CIA).
eg: "Customer financial data must remain confidential and protected from unauthorized access."
2. Define Security Requirements: Translate goals into actionable requirements (functional + non-functional).
eg: "The system shall enforce multi-factor authentication for all privileged users."
3. Formal Specification of Properties: Use formal methods or structured notations to specify desired properties clearly.
eg: "Access control policies, Security Models (Bell-LaPadula for confidentiality, Biba for integrity)."
4. Assertion Techniques: state security properties as assertions within the systems design or code.
eg: "Assertion in code that validates user input must never contain SQL control characters."
5. Verification and Validation: Use testing, static analysis, model checking and formal verification tools to ensure properties hold true.

eg: "Use symbolic execution tools to confirm that no authorized data flows exist."

⑥ Continuous Monitoring and Enforcement: Security properties must be enforced at runtime to detect and prevent violations.

eg: "Intrusion Detection System (IDS) monitor for suspicious behaviour violating defined policies."

Q.10 Explain partially expanded security assurance case that focuses on buffer overflow.

—4 A Security Assurance Case (SAC) is a structured argument with evidence showing a system is secure. A partially expanded SAC targets a specific vulnerability – such as Buffer overflow and shows how its has been mitigated.

1. Top-level Claim
"The System is free from exploitable buffer overflow vulnerabilities", this claim is supported using Goal ~~Structuring~~ Structuring Notation (GSN).

2. Supporting Subclaims
① Design Defenses: Use of secure coding standards (eg. CERT), safe libraries and defensive programming practices.
② Implementation Defenses: Input validation, use of secure functions (snprintf, strncpy) and compiler protections like ASLR and stack canaries.
③ Testing & Detection: Code reviews, static analysis tool (eg: Fortify) and robustness testing confirm

* For diagram, refer Chapter 2 Book, Pg No: 41

the absence of buffer overflows.

3. Evidence Examples.
① Reviewed design documents and coding standards.
② Static analysis reports showing no buffer-related warnings.
③ Test results confirming safe handling of invalid input.

Q.4 Describe the importance of Requirements Engineering for secure software.

→ Requirement Engineering (RE) is the process of eliciting, analyzing, specifying, validating and managing what a system must do. When applied to secure software, RE ensures that security is built-in from the start, rather than added later as a patch or workaround.

① Early Integration of Security: Security flaws introduced during requirements are the most expensive to fix. Identifying them early avoids ~~it~~ insecure design and reduces rework.
② Clear and Specify Security Goals: Defined requirements clarify essential properties like CIA, authentication and accountability, avoiding vague security expectations.
③ Risk and Threat Awareness: RE includes threat modeling and risk analysis, helping teams understand how attackers might exploit the system and define countermeasures.
④ Legal and Compliance Alignment: It support compliance with standards (eg: GDPR, HIPAA), which is critical in

fields like healthcare, finance and defense.

(5) Traceability and Verification: Well-Specified requirements can be traced through design, implementation and testing, making it easier to verify that critical security needs are met.

Q.12 Explain techniques of ensuring that the resulting product effectively meets security requirements.

-4 To build secure software, it's not enough to identify security requirements — they must be effectively implemented and validated throughout the development lifecycle.

(1) Misuse and Abuse Cases: These extends traditional use-cases analysis by documenting what the system should not do. It focuses on negative scenarios from an attacker's perspective, helping uncover hidden weaknesses and define defensive behaviours.

(2) SQUARE (Security Quality Requirements Engineering): A systematic process that integrates security early in the SDLC through steps like defining goals, risk assessments and prioritizing requirements based on business impact.

(3) CLASP (Comprehensive, Lightweight Application Security Process): An activity-based framework embedding security across the lifecycle. It aligns requirements with secure design and coding practices, while remaining lightweight and practical.

(4) Threat Modeling and Attack Patterns: Analyzes the

system from an attacker's view, asking "what can go wrong?" Using known attack patterns, it helps define countermeasures and identify key requirements early.

(5) Requirements Prioritization: Not all requirements can be implemented equally. Techniques like AHP prioritize them based on risk, cost and impact - ensuring the most critical security features come first.

Q.13 Explain Misuse cases and Abuse case with appropriate example.

-> These cases extendes user-modeling to identify threats, malicious behaviour and vulnerabilities early in development, helping integrate security requirements efficiently.

1. Misuse Case

It is a negative usecase that describes how a system can be misused either intentionally by an attacker or unintentionally by a user.

The purpose is to identify potential security vulnerabilities and define corresponding ~~activity~~ requirements to prevent or mitigate such actions.

eg:
- System: Online Banking Application
- Misuse Case: An attacker attempts to perform a SQL injection attack on the login page.
- Security Requirement: Implement input validation and parameterized queries to prevent injection.

2. Abuse Case

It describes a malicious activity explicitly

performed by an attacker's ~~behaviour~~ to harm the system or its users.

The purpose is to focus on intentional, hostile actions and understand the attacker's behaviour in order to build defensive countermeasures.

eg:
- System : E-commerce website
- Abuse case : An attacker uses a brute-force attack to guess user passwords.
- Security Requirement : Implement account lockout after multiple failed attempts and enable multi-factor authentication (MFA).

Q.14 — Explain first six steps of SQUARE Process.

SQUARE (Security Quality Requirements Engineering) is a structured methodology that ensures security requirements are systematically integrated into the software development lifecycle.

Step-1: Agree on Definitions

Ensure all stakeholders share a common understanding of key security terms (eg: authentication, authorization). This prevents miscommunications & flawed assumptions.

Outcome: Glossary of agreed-upon security terms.

Step-2: Identify Security Goals.

Define high-level security objectives based on business needs, ~~pots~~ policies and risks. These goals guide all security requirements and align with organizational priorities.

Outcome: Documented list of security goals (eg: Protect data, ensure availability).

Step-3: Develop Artifacts
Create supporting ~~artifacts~~ artifacts like architecture diagram, use/misuse cases and scenarios to provide content for understanding threats and system behaviour.
Outcome: Set of models and diagrams for analysis.

Step-4: Perform Risk Assessment
Identify threats, vulnerabilities and impact to prioritize what needs protection. Use techniques like threat modeling, risk matrices and asset identification
Outcome: Prioritized list of security risks.

Step-5: Select Elicitation Techniques.
Choose suitable methods (interviews, workshops, checklists, etc.) to gather security requirements based on the project's content.
Outcome: Documented strategy for elicitation.

Step-6: Elicit Security Requirements.
Apply selected techniques to collect security requirements from stakeholders, covering expectations, legal obligations and threat responses.
Outcome: Initial set of elicited security requirements.

Q.15 — Discuss Ranking of Assessment Techniques.
Assessment techniques are methods used to evaluate how well a system meets its security requirements and to identify possible weaknesses.

① Suitability for small companies: Evaluates if a technique is pratical for small organization with limited budgets and staff. Lightweight, easy-to-use methods that require minimal tools rank higher for scalability and flexibility.

② Feasibility of completion in the Time allotted: Considers whether the technique can be completed efficiently within tight project timelines. Methods that avoid ~~for~~ lengthy data collection or complex documentation score better.

③ lack of Dependence on Historical Threat Data: Assesses if the technique works without needing prior threat or incident data. Forward-looking approaches like threat modeling are preferred for new projects or data-scarce environments.

④ Suitability in Addressing Requirements: Measures how well the technique support identifying and validating clear, actionable security requirements, often involving stakeholders collaboration and misuse case analysis to align with business needs.

Q.16 — 4 Describe any six elicitation methods.

Requirements elicitation is the process of gathering requirements from stakeholders, users, customers and other sources to ensure the developed software meets expectations.

① Misuse Cases: Extend use cases by focusing on unwanted actions or attacker behaviour to identify security flaws early.

eg: Documenting attempts to bypass login controls.

② Soft Systems Methodology: Explores different stakeholder views to clarify ambiguous requirements through iterative discussions.

eg: Interviewing users with varying security needs in healthcare.

(8) Quality Function Deployment (QFD): Transforms customer needs into technical requirements using structured matrices.

eg: linking data confidentiality to encryption design.

(4) Joint Application Development (JAD): Facilitates workshops with developers and stakeholders to quickly define and prioritize requirements.

eg: Collaborating on authentication needs with security experts.

(5) Feature-Oriented Domain Analysis: Analyzes features across similar systems to identify common and unique security requirements.

eg: Using banking app features to guide a new financial platform.

(6) Controlled Requirements Expression: Uses formal languages to specify requirements precisely and reduce ambiguity.

eg: Defining password rules clearly (eg: minimum length).

Q.17 Explain Elicitation Evaluation Criteria in detail.

→ Elicitation evaluation criteria are the standards used to judge the effectiveness, efficiency and reliability of different elicitation methods in the requirements engineering process.

(1) Adaptability: The ability of the technique to adjust to different project contexts, team sizes and evolving requirements without losing effectiveness.

(2) Computer-Aided Software Engineering (CASE) Tool Support: Availability of tools that assist in automating, documenting or managing the elicitation process, making it more efficient and organized.

③ Stakeholder Acceptance: The degree to which stakeholders are comfortable and willing to participate in the elicitation technique, ensuring better collaboration and input.

④ Graphical Output: The capability of the technique to produce visual representations (eg: diagrams, charts) that aid understanding and communication among stakeholders.

⑤ Shallow learning curve: Techniques that require minimal training or expertise, enabling team members to use them effectively without extensive preparations.

★ Also Easy implementation, Quick implementation, High maturity & Scalability.

Q.18 — 4 Explain Binary Search Tree (BST) and Theory W.

A Binary Search Tree (BST) is a structured data model used to prioritize requirements through comparative ranking. It helps organize requirements efficiently by inserting them based on importance.

- Working for Requirements Prioritization:
  - All requirements start in a single list.
  - One requirement is selected as the root (reference point).
  - Each new requirement is compared with the root:
    - > If less important, it goes to the left.
    - > If more important, it goes to the right.
  - The process continues recursively, forming a tree structure based on priority.

- To obtain the final prioritized list, perform an in-order traversal, which arranges them from least to most important.

- Theory W is a stakeholder-centered approach to negotiating software requirements. It aims to achieve a solution where all stakeholders feel like winners — balancing business, technical and user needs.
  1. Plan the flight and fly the plan: Encourage well-structured and realistic planning.
  2. Identify and manage your risks: Address potential compromises to stakeholder goals.

- Working :-
  - Each stakeholder privately ranks their winning and losing conditions (what they want vs. what they can't accept)
  - A collaborative negotiation follows to align these preferences.
  - The final outcome ensures no major stakeholder feels disadvantaged.

⇒ Theory W states: "Make Everyone a Winner."