30/7/25

ASSIGNMENT 1

Unit-1 Why Security is a Software Issue?

Q.1 What are the common sources of software insecurity?

→ Software insecurity refers to vulnerabilities or weakness in software systems that can be exploited by attackers to compromise the confidentiality, integrity or availability of the system and its data.

These insecurities typically arise from both high-level design flaws and low-level implementation issues.

1. Design and Development Issues (Conceptual Causes).
   (i) Flawed Specifications: Missing or insecure input validation, weak error or exception handling and unclear trust boundaries.
   (ii) Lack of security knowledge: Developers may not be trained in secure coding practices or may misunderstand the security implications of their design decisions.
   (iii) Software Complexity: larger and more interconnected systems are more difficult to secure and more likely to have overlooked vulnerabilities.
   (iv) Incorrect Assumptions: Developers may incorrectly assume users, inputs or external systems will behave safely or predictably.
   (v) Unintended Interactions: Components (especially third-party) may interact in unexpected ways that introduce security flaws.

2. Technical Vulnerabilities (Implementation Issues)

ASSIGNMENT 1

(i) Coding Errors: Issues like buffer overflows or null pointer dereferencing can be exploited by attackers.

(ii) Lack of Authentication / Authorization: Weak or missing access control mechanisms can lead to unauthorized access.

(iii) Lack of security Testing: Skipping static analysis, dynamic testing or penetration testing means vulnerabilities remain undiscovered.

(iv) Hardcoded secrets: Embedding credentials or API keys in code can lead to easy extraction and misuse.

Q.2 Define software assurance and its role in software security.

—4 Software Assurance is defined as "The level of confidence that software is free from vulnerabilities, either intentionally designed or accidentally inserted and functions as intended in its operating environment." It ensure that software is secure, reliable and trustworthy.

(i) Risk Mitigation: Identifies and reduces the impact of software vulnerabilities before they can be exploited.

(ii) Verification & Validation: Ensures the software behaves as intended and is free from security flaws through rigorous Testing.

(iii) Threat & Post-Deployment Monitoring: Anticipates

potential attacks before deployment and ensures ongoing security through continuous monitoring, updates and patching after release.

(iv) Secure & Compliant Development : Promotes secure coding from the early stages while ensuring adherence to recognized security standards and frameworks like OWASP, NIST & ISO/IEC 27001.

(v) Cost-Effective security : By integrating security early reduces costly post-release fixes and offers a 12-21% return on investment.

Q.3 list the benefits of detecting software security defects early in the SDLC.

→ Detecting security defects early means finding vulnerabilities during requirements, design or coding phases.

(i) Cost savings : Fixing defects early — during design or requirements — is much cheaper, often 50 to 200 times less costly than post-deployment fixes.

(ii) Faster Development : Early detection avoids delays from late fixes, shortening development cycles and speeding time-to-market.

(iii) Improved Quality : Catching security flaws early results in more secure, stable software and higher customer satisfaction.

(iv) High ROI : Early security practices can yield a 12-21% return by reducing costly rework later.

(v) lower Security Risks : Early fixes reduces the chance of serious attacks or data breaches after release.

(vi) Avoid Post-Release Issues: Early QA prevents defect surges, leading to more stable releases.

* Additional are supports avoiding schedule overrun, Meeting standards, reducing error-prone modules and better resource use.

Q.4 Identify the primary problem with software security.

→ The primary problem is the widespread presence of software vulnerabilities that attackers exploit, causing significant risks.

(i) Global Connectivity Risks: Modern software's extensive internet connectivity increases exposure to unauthorized access, putting sensitive data - like financial information and personal identities - at higher risks.

(ii) Increasing System Complexity: As software systems grow larger and more complex (eg, Microsoft windows), the number of bugs and vulnerabilities rises, making strong security harder to achieve.

(iii) Ubiquitous Vulnerabilities: Coding errors and design flaws are common and difficult to eliminate, allowing attackers to exploit these weaknesses through malware and botnets.

(iv) Sophisticated Attack Techniques: Cyber threats such as information warfare, cyberterrorism and cybercrime have led to advanced, easier-to-launch attacks targeting software lacking resilience

(v) Lack of Disciplined Development: Without rogorous, disciplined development practices, software often contains uncontrolled vulnerabilities and inconsistent security.

Q.5 Explain why security is considered a software issue.

-4 In today's digital world, software powers everything from banking systems to healthcare devices, making software security central to overall system security. Security is no longer just about hardware or networks - it is fundamentally a software issues because most of cyberattacks target flaws within the software itself.

(i) Ubiquitous Presence of Software: Software is embedded in critical sectors like finance, healthcare and transportation. Its integration across connected systems makes it a prime target, where a single flaw can cause widespread impact.

(ii) Inherent Vulnerabilities: Most threats exploit software flaws - coding errors, design weakness or misconfigurations. Examples include buffer overflows, SQL injection and cross-site scripting (XSS).

(iii) Insider & External Threats: Flaws may be introduced unintentionally by insiders or exploited by external attackers. Both must be addressed for secure systems.

(iv) System Complexity & Internet Connectivity: Integrating

complex or legacy systems with ~~other~~ online features introduces vulnerabilities. Internet exposure increases the risk of attacks like data theft or cyberterrorism.

(iv) Evolving Attack Methods: Attackers use tools like botnets and exploit kits to target known software flaws, as shown by the rising number of CERT-reported vulnerabilities.

Q.6 Describe the relationship between defect rates and development time.

-4 Defect Rate refers to the number of defects per unit of code usually measured as defects per KLOC. The Development Time refers to the time allocated for planning, designing, coding and testing the software.

There is a direct and inverse relationship between defect rates and development time: "the more defects a project accumulates, the longer and more costly it becomes."

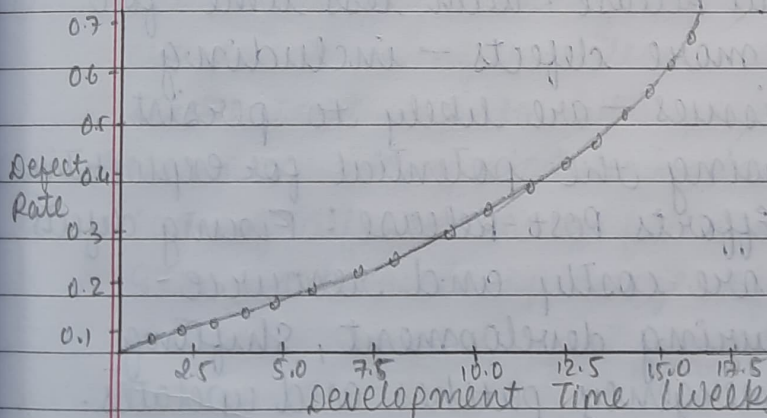
(i) Early ~~bugs~~ defects are costlier if Delayed: Defects introduced during early phases — like requirements or design — often stem from unclear specifications. If not caught early, they become far more expensive and time-consuming to fix later, especially post-release.

(ii) Quality Shortcuts Backfire: Under deadline ~~pressure~~ pressure, teams may skip testing or reviews. While this may seem to speed things up, it typically leads

to more defects, longer schedule and higher costs. Post-release fixes can be 50-200 times more expensive then early corrections.

(iii) Quality Enables Speed: Projects that eliminate at least 95% of defects before release tend to have the shortest schedules and highest user satisfaction. Prioritizing quality through testing, reviews and continuous integration ensures smooth delivery.

(iv) Iterative Development Helps: Agile and iterative methods reduce defect rates by promoting early testing, continuous feedback and small, manageable releases — helping teams catch and fix issues before they "snowball".



Q.7 Summarize the impact of compressing the testing schedule on software security.

—4 Testing is a critical phase in the SDLC for ensuring security. Compressing the testing schedule — often to meet tight deadlines — can seriously harm the security, quality and reliability of the final product.

(i) Increased Risk of Undected Vulnerabilities: Shortened

testing phases often lead to incomplete coverage, allowing security flaws such as coding errors or design weaknesses to go unnoticed and remain in the released software.

(ii) Reduced Confidence and Trust: Software released with unresolved security issues can undermine user trust, damage reputation and lead to compliance failures, all of which negatively impact the organizations.

(iii) Compromised Compliance and Standards: Insufficient testing may ~~be~~ lead to failure in meeting security standards and regulations, exposing the organization to legal and financial risks.

(iv) Higher Defect Rates at Release: With less time for thorough testing, more defects - including serious security issues - are likely to persist at release, increasing the potential for exploitation

(v) Greater Costs and Efforts Post-Release: Fixing defects after release is more costly and resource-intensive than during development, shifting the burden to emergency patches and updates.

eg: In a real-world web application project:

> The team skips penetration testing to meet a release deadline.

> Post-deployment, the application is exploited via cross-site scripting (XSS) due to lack of input validation checks.

> Result: User data is compromised, legal abilities arise and emergency patches are required.

Q.8 Illustrate the significance of the "95 percent defect removal" line.

→ Defect Removal Efficiency (DRE) refers to the percentage of defects identified and fixed before the software is released. The "95% defect removal" line is a benchmark that indicates the minimum acceptable standard for high-quality and secure software.

A 95% defect removal rate means that 95% of all introduced defects are detected and fixed before release, leaving only 5% to potentially reach end-users.

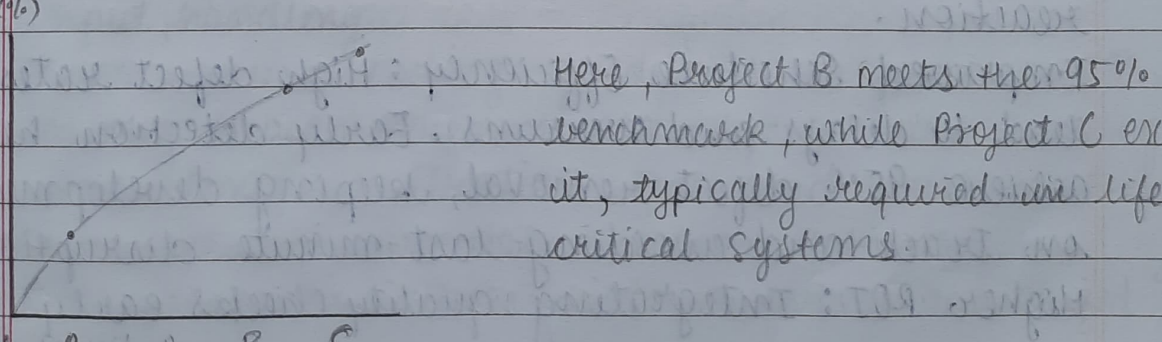
(i) Faster Development: Early defect removal reduces rework, helping projects finish on time with fewer days.

(ii) Cost Efficiency: Fixing defects early is 50-200 times cheaper than after release. A $1 issue during design could cost up to $100 post-release.

(iii) Focus on Risky Modules: Since 20% of modules causes 80% of defects, early focus on these supports the 95% goal and optimize effort.

eg:

Project	Total Defects Introduced	Defects Removed Before Release	DRE(%)
Project A	1000	800	80%
Project B	1000	950	95%
Project C	10,000	990	99%



Here, Project B meets the 95% benchmark, while Project C exceeds it, typically required in life-critical systems.

Q.9 Apply the concept of early defect detection to explain its cost-saving potential.

—4 Early defect detection means identifying and resolving software defects during the early stages of the SDLC, rather than during testing or after deployment.

It is closely tied to DRE and CoQ (Cost of Quality) — as the cost to fix defects increases exponentially the later they are discovered in the SDLC.

(i) Lower Fixing Costs: Fixing defects during early phases costs around $1, while the same issue may cost $60-$100 after release.

(ii) Reduced Rework Effort: Rework can account for 40-50% of development costs. Early defect detection minimizes unnecessary changes across code, tests, and documentation — saving 3-10 hours of repair for every hour of prevention.

(iii) Prevention of Cascading Issues: Undetected defects in early phases can propagate. For example, a single flawed requirement can result in hundreds of lines of faulty code, inflating correction costs. Early detection stops this chain reaction.

(iv) Improved Schedule Efficiency: High defect rates lead to delays and overruns. Early detection helps achieve 95% defect removal, keeping development on track and avoiding last-minute disruptions.

(v) Higher ROI: Integrating quality checks early

delivers a 12-21% ROI, as fewer defects post-release mean lower maintenance and support costs.

Q.10 How would you integrate security practices into the SDLC to enhace software security?

→ To build secure software, security must be integrated into every phase of the SDLC - not treated as an afterthought. This approach is known as "Secure SDLC" or "security by Design."

(i) Requirements Phase: Define clear security requirements and include misuse/abuse cases to anticipate attacker behaviour. Focus on input validation and handling of malicious inputs.

eg: Include input validation to prevent injection attacks or buffer overflows.

(ii) Design Phase: Conduct risk analysis and threat modelling to identify attack vectors. Ensure the design supports resistance and resilience against common threats.

eg: Design APIs to detect and block SQL injection.

(iii) Development / Coding Phase: Use secure coding practices and static analysis tools. Train developers to avoid common flaws like ~~it~~ buffer overflows and insecure input handling.

eg: Follow secure coding guidelines to prevent XSS.

(iv) Testing Phase: Perform security-focused testing, including penetration tests and abuse case scenarios to stimulate attacker actions.

eg: Run penetration tests to check for unauthorized access.

(v) Deployment & Operations Phase: Apply secure configuration controls and monitor for emerging threats. Keep systems updated and review configurations regularly.

eg: Use scanners to detect unpatched vulnerabilities in production.

(vi) Cross-Cutting Pratice (RMF): Apply continuous risk assessment throughout the SDLC. Use RMF to identify and prioritize high-risk components.

eg: Focus security controls on external interfaces and critical modules.

Q.11 Demonstrate how a risk management framework can be used to manage software security risks.

→ A Risk Management Framework (RMF) helps organizations identify, assess, prioritize and mitigate software service risks throughout the SDLC. It provides a structured and repeatable approach to manage security vulnerabilities and threats effectively.

(i) Risk Identification: At each SDLC phase, RMF helps identify potential threats and vulnerabilities — such as insecure inputs, third-party dependencies or exposed interfaces — that could compromise software security.

(ii) Risk Assessment: Risks are analyzed based on likelihood and impact. RMF supports prioritization by focusing resources on high-risk areas, such as modules handling sensitive data or

external communication

(iii) Risk Mitigation : The framework guides the selection of appropriate controls to reduce eliminate the identified risks.

(iv) Risk Monitoring and Review : RMF ensures continuous monitoring of security risks, especially after deployment. It supports updating controls based on new vulnerabilities, threat intelligence or system changes.

(v) Continuous Risk-Based Decision Making : By integrating RMF throughout the SDLC, organizations can make informed security decisions, balance, cost and risk, and maintain compliance with standards.

- Risk Assessment Techniques are STRIDE (Microsoft), DREAD (Microsoft), CVSS (Common Vulnerability Scoring System).

Q.12 Use an example to show how error-prone modules can affect software development.

→ Error-prone modules are components or units of a software system that are frequently associated with bugs, defects or vulnerabilities due to their complexity, poor design, inadequate testing or weak documentation. They negatively impact software quality, maintenance and security

eg: Banking System
In a banking application with modules like :
- User Interface - Handles input / display

- Transaction Module - Manages deposits, withdrawals, transfers
- Authentication - Handles login / security
- Reporting - Generates statements.

Suppose the Transaction Module is error-prone due to:

- Poor balance-check logic
- Incomplete input validation
- lack of concurrency control.
- No rollback on failure.

⇒ Impact of the Error-Prone Transaction Module:

Area affected	Explanation.
Data Integrity	- Double withdrawals or inconsistent balances from unvalidated inputs.
Security	- Exploitable flaws
Development Time	- More debugging delays the overall release
Maintenance Cost	- Frequent hotfixes and rework increase long-term cost
Customer Trust	- Users lose confidence due to failed or incorrect transactions
System Stability	- A single failure can affect multiple components or crash the systems