جامعة المستقبل
AL MUSTAQBAL UNIVERSITY

كلية العلوم

قسم الأمن السيبراني

Lecture: 9

Defense Strategies

Subject: software security

second Stage

Lecturer: Asst. Lecturer. Suha Alhussieny

Defense Strategies

A 'Defense Strategy' in the context of Computer Science refers to the systematic approach of layering defenses to enhance effectiveness and reduce costs in protecting against cyber-attacks. It involves implementing controls to mitigate potential impacts and improve decision-making for responding to attacks. Defense strategies in software security aim to protect applications and systems from various types of attacks and vulnerabilities. A multi-layered approach is often used to ensure comprehensive protection. Here are some key defense strategies:

Software Verification

Software verification is a crucial process in ensuring that software systems meet their specified requirements and function correctly. It involves various techniques and methods to confirm that the software behaves as intended and is free of critical errors.

Software verification Ensure that software requirements are clear, complete, and feasible before development begins. Verify that each requirement is addressed by corresponding design elements and test cases.

Static Code Analysis: Use tools to analyze source code without executing it. This helps in identifying potential issues such as coding standards violations, security vulnerabilities, and other defects.

Dynamic Verification: Test individual components or units of code in isolation to ensure they work correctly. Test the interactions between integrated components or systems to identify issues related to their interaction. Verify the

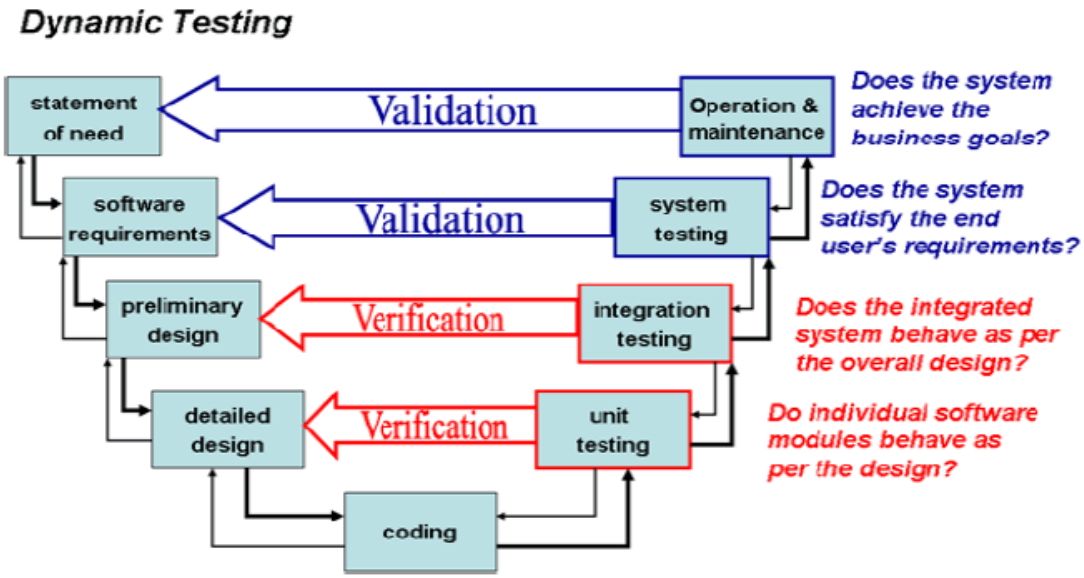
complete and integrated system against the requirements. This includes functional testing, performance testing, and security testing.

Verification Techniques

• **Model-Based Verification**: Create and analyze models of the software to verify that it meets its requirements and behaves correctly.

• **Formal Specification**: Use formal methods to specify and reason about software behavior mathematically.

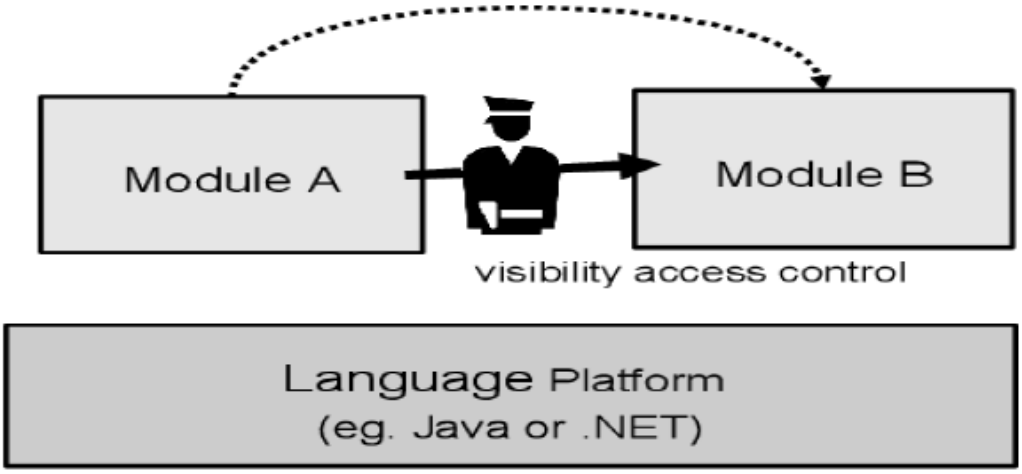
• **Simulation and Emulation**: Test software in a simulated or emulated environment to verify its behavior under controlled conditions.



Language-based Security

Language-based security focuses on using programming languages and formal methods to enhance the security of software systems. This approach involves designing languages and language features that help prevent security vulnerabilities, such as buffer overflows or injection attacks. Here are some key concepts:

1. **Type Systems:** Strong type systems can help catch errors at compile time, preventing many common vulnerabilities. For example, languages like Rust use ownership and borrowing to manage memory safely.
2. **Formal Verification:** This involves mathematically proving that a program adheres to its specification and is free from certain types of errors. Languages designed with formal verification in mind, such as Ada or Coq, can ensure higher levels of reliability and security.
3. **Safe Programming Constructs:** Languages can provide constructs that limit the kinds of operations that can be performed, reducing the risk of errors. For instance, languages like Java and C# provide automatic garbage collection to avoid memory leaks and dangling pointers.
4. **Secure Language Design:** Some languages are designed with security as a primary concern. For example, languages like Haskell and Scala have features that support functional programming, which can make it easier to reason about and ensure the security of code.
5. **Security-focused Libraries and Frameworks:** Many modern languages come with libraries and frameworks designed to prevent common security issues, like SQL injection or cross-site scripting (XSS).



Testing

In the context of software testing, a defense strategy involves incorporating various practices and methodologies to safeguard the software from defects, vulnerabilities, and performance issues.

Manual Testing

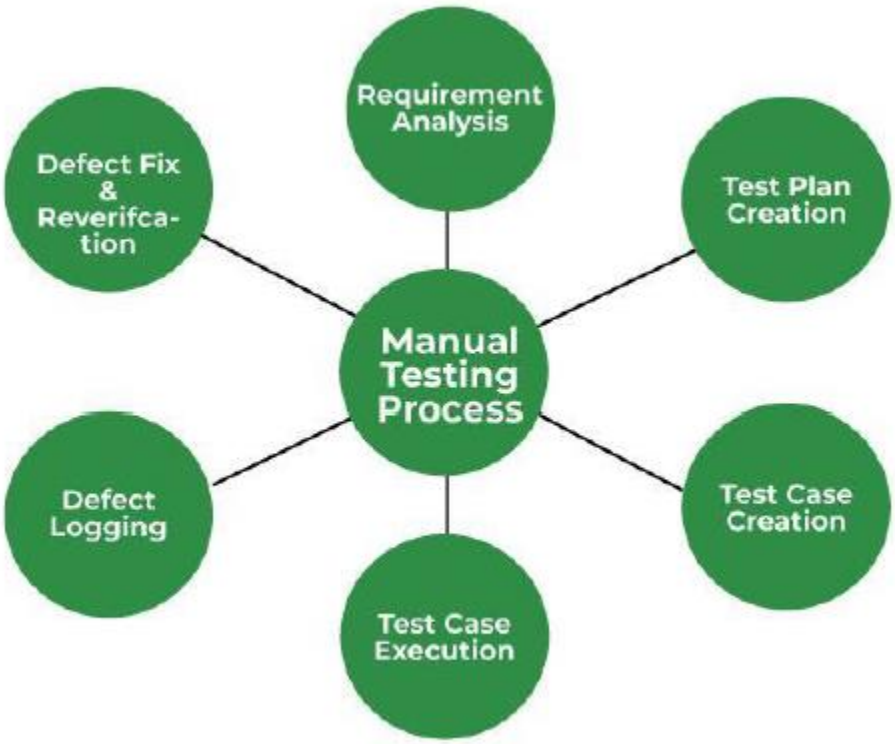
Manual software testing involves a tester manually executing test cases without the use of automated tools. The goal is to identify bugs or issues in the software by simulating the end-user experience. Here's a basic outline of how it typically works:

1. **Test Planning**: Define the scope and objectives of testing. Create a test plan that outlines what to test, how to test, and the resources required.
2. **Test Case Design**: Develop detailed test cases based on the software requirements and specifications. Each test case should have clear steps, expected results, and conditions.

3. **Test Execution**: Manually execute the test cases on the software application. This involves navigating through the application, entering data, and verifying that the software behaves as expected.
4. **Defect Reporting**: When issues or bugs are found, document them in a defect tracking system. Include details such as steps to reproduce, expected vs. actual results, and severity.
5. **Test Closure**: Once testing is complete, review and analyze the results. Ensure all critical issues are resolved and finalize testing documentation.
6. **Regression Testing**: After fixes are applied, retest the affected areas to ensure that the changes didn't introduce new issues.

The below diagram lists the steps in the manual testing process:

1. **Requirement Analysis:** Study the software project documentation, guides, and Application Under Test (AUT). Analyze the requirements from SRS.
2. **Test Plan Creation:** Create a test plan covering all the requirements.
3. **Test Case Creation:** Design the test cases that cover all the requirements described in the documentation.
4. **Test Case Execution:** Review and baseline the test cases with the team lead and client. Execute the test cases on the application under test.
5. **Defect Logging:** Detect the bugs, log and report them to the developers.
6. **Defect Fix and Re-verification:** When bugs are fixed, again execute the failing test cases to verify they pass.



Q/

1. What is the primary goal of a defense strategy in software security?

A) To reduce software complexity
B) To improve software performance
C) To layer defenses for enhanced security
D) To speed up development
E) To reduce hardware costs

2. Which of the following is NOT a key concept in language-based security?

A) Type Systems
B) Formal Verification
C) Secure Language Design
D) Test Case Execution
E) Safe Programming Constructs

3. What does software verification aim to ensure?

A) That the software meets performance expectations
B) That the software is bug-free
C) That the software meets its specified requirements and functions correctly
D) That the software has a user-friendly interface
E) That the software runs faster

4. What is the primary purpose of static code analysis?

A) To analyze source code without execution
B) To execute test cases automatically
C) To enhance network security
D) To analyze performance under stress
E) To review UI/UX design

5. Which of the following is a verification technique?

A) Garbage collection
B) Model-Based Verification
C) SQL injection
D) Web Scraping
E) Debugging

6. What is the main benefit of using strong type systems in programming languages?

A) They improve software execution speed
B) They prevent common vulnerabilities at compile time
C) They make coding more complex
D) They improve graphical user interfaces
E) They help with hardware optimizations

7. Which of the following languages is known for its strong ownership and borrowing system to manage memory safely? A) Python
B) Rust
C) JavaD) JavaScript

E) PHP

8. In software verification, what is dynamic verification?

A) Testing without execution
B) Executing individual software components in isolation
C) Checking requirements before development
D) Analyzing logs post-deployment
E) Reviewing project documentation

9. Why is formal specification important in software security?

A) It allows programmers to skip testing
B) It ensures software behavior is mathematically proven to meet requirements
C) It speeds up the coding process
D) It reduces the need for debugging
E) It improves software aesthetics

10. Which of the following is NOT part of the manual testing process?

A) Test Planning
B) Defect Reporting
C) Garbage Collection
D) Test Execution
E) Regression Testing

11. What is the role of security-focused libraries in modern programming languages?

A) To increase software development costs
B) To prevent security vulnerabilities such as SQL injection and XSS
C) To improve application aesthetics
D) To make programming easier for beginners
E) To enhance internet browsing speed

12. What is the first step in the manual testing process?

A) Test Plan Creation
B) Defect Logging
C) Requirement Analysis

D) Test Execution
E) Regression Testing

13. Why is regression testing important?

A) To speed up the development cycle
B) To ensure that software requirements are well-documented
C) To verify that recent changes did not introduce new bugs
D) To check network security
E) To test only new features

14. What does dynamic verification typically involve?

A) Running code to test interactions and performance
B) Reviewing test documentation
C) Writing test plans before development
D) Running automated scripts without execution
E) Checking UI elements for consistency

15. What is a key advantage of using a multi-layered defense strategy in cybersecurity?

A) It speeds up software development
B) It reduces the number of employees required
C) It enhances protection by addressing multiple attack vectors
D) It simplifies the software design process
E) It reduces system memory usage